Algorithms for Minimum-violation Planning with Formal Specifications

by

Luis I. Reyes Castro

B.Sc., Georgia Institute of Technology (2011)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Author .....
Department of Aeronautics and Astronautics
May 22, 2014

Certified by .....
Emilio Frazzoli
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Algorithms for Minimum-violation Planning with Formal Specifications

by

Luis I. Reyes Castro

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

We consider the problem of control strategy synthesis for robots given a set of complex mission specifications, such as "eventually visit region A and then return to a base", "periodically survery regions A and B" or "do not enter region D". We focus on problem instances where there does not exist a strategy that satisfies all the specifications, and we aim to find strategies that satisfy the most important specifications albeit violating the least important ones. We focus on two particular problem formulations, both of which take as input the mission specifications in the form of Linear Temporal Logic (LTL) formulae. In our first formulation we model the robot as a discrete transition system and each of the specifications has a reward associated with its satisfaction. We propose an algorithm for finding the strategy of maximum cumulative reward which has a significantly better computational complexity than that of a brute-force approach. In our second formulation we model the robot as a continuous dynamical system and the specifications are associated with priorities in such a way that a specification with priority i is infinitely more important than one with priority level j, for any $i < j$. For this purpose, we introduce a functional that quantifies the level of violation of a motion plan and we design an algorithm for asymptotically computing the control strategy of minimum level of violation among all strategies that guide the robot from an initial state to a goal set. For each of our two formulations we demonstrate the usefulness of our algorithms in possible applications through simulations, and in the case of our second formulation we also carry experiments on a real-time autonomous test-bed.

This is joint work with Jana Tumova, Pratik Chaudhari and Sertac Karaman.

Thesis Supervisor: Emilio Frazzoli
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank Prof. Emilio Frazzoli, my research adviser and also the brightest and most hilarious aerospace engineer I have ever met, for his guidance and support throughout my graduate studies here at MIT. Also, I am extremely thankful to Jana Tumova and to Pratik Chaudhari, without whom this thesis could have never become a reality. In addition, I am very thankful to Prof. Panagiotis Tsiotras at Georgia Tech for introducing me to the wonderful field of control systems and robotics.

I would also like to thank Prof. Sertac Karaman for his useful advice. Of course, the people at LIDS also played an essential role in my success, particularly Kyle Treleaven, Kevin Spieser, Josh Bialkowski, Jeong hwan Jong, Phil Root, Noele Norris, Hajir Roozbehani, Ali Faghih, Omer Tanovic, Shreya Saxena, and Swati Gupta.

Last but not least, I am grateful to my parents for their love and support.

Contents

List of Figures

Chapter 1

Introduction

Control strategy synthesis for robotic systems with complex and formally-specified mission specifications has recently gained considerable attention in the robotics literature. For this purpose, a diverse set of techniques, such as triangulation-based cell decomposition and random sampling, have been applied to the construction of discrete models of robotic systems. With these models at hand, a variety of automata and temporal logics have been successfully utilized to specify the type of complex missions that arise in modern robotics. By far the most popular of these languages is the Linear Temporal Logic (LTL), which is the mission specification language in [TP06], [KB08], [KGFP09], [BKV10],[WTM10], [DSBR11], [STBR11], [USDB12], and in many other works. Also employed to encode mission specifications are the Probabilistic Computational Tree Logic (PCTL), which is used in [LWAB10], and the Deterministic μ-Calculus, which is used in [KF09]. Nonetheless, so far the large majority of the work in the literature has focused on the problem of synthesizing a control strategy for the robot that satisfies all the mission specifications, if one exists, and to report failure otherwise.

The usual execution of many robotic systems, however, involves cases where not all of the mission specifications can be satisfied at the same time. Yet, in most such cases, it is desirable to synthesize a control strategy that fulfills the most important mission specifications by temporarily violating some of the less important specifications if necessary. For instance, consider an autonomous passenger vehicle navigating in urban traffic. The vehicle must reach some destination while abiding by the rules of the road, such as always driving on the right lane. However, this robot, as well as any human driver, must sometimes drive on the left lane if it is to eventually reach its destination. In fact the right lane

rule is temporarily violated, for instance, when maneuvering around a parked car in a two-way street. For another example, consider Isaac Asimov's "three laws of robotics", which in Asimov's fictional universe dictate how robots interact with humans. According to these laws, a robot must disregard or violate any order given by a human operator if said order endangers the life of another human being [Asi50].

Additionally, as pointed out in [WTM10], one of the main challenges in current approaches to the synthesis of control strategies subject to formal specifications is the abstraction of continuous dynamical systems into discrete models that preserve the essential properties of the original systems. The issue arises because discrete models are unavoidable in this type of problems, as temporal logics and automata are intrinsically discrete mathematical objects. Unfortunately, given a fixed abstraction of a continuous dynamical system, there is no guarantee that a particular path of the continuous system exists in the discretized version of the system, and so algorithms for kinodynamic planning with logical constraints that rely on fixed abstractions are only complete up to the resolution of the discretization.

In light of these limitations, the motion planning community has recently introduced methods based on the Probabilistic Road Maps (PRM) and Rapidly-Exploring Random Trees (RRT) algorithms which are capable of asymptotically solving the motion planning problem for continuous dynamical systems operating in cluttered environments to true completeness. Specifically, the PRM* and RRT* algorithms, first introduced in [KF11], are computationally efficient counterparts of the PRM and RRT algorithms, respectively, which guarantee almost sure convergence to optimal solutions. Nevertheless, only in the last couple of years these new algorithms have been adapted to solve problems with temporal logic constraints [KF12].

Motivated by these challenges, in this thesis we consider the problem of minimum-violation control strategy synthesis. That is, we study the problem of finding a path or motion plan for a robot that satisfies the most important aspects of the mission specifications, even in the cases where the mission specifications cannot be fulfilled as a whole. In particular, in Chapter 3 we investigate this problem for the case where the robot dynamics are abstracted into a fixed discrete model and the specifications are given as infinite LTL formulae. In turn, because of the completeness issues discussed above, in Chapter 4 we consider this problem for the case where the robot is a continuous-time continuous-space dynamical system and the mission specifications are given as particular type of finite $\text{LTL}_{-\mathsf{X}}$ formulae.

The work in this thesis is related to that in [RKG11] and [RKG12], where the authors study the following problem: Given a set of LTL formulae and a discrete model of a

robot that cannot satisfy all of the formulae, decide whether or not the unsatisfiability of the formulae is limited to the specific robot model at hand. Related literature also includes [CRST08], where the authors study the algorithmic generation of explanations of specification unsatisfiability. Other related works include those in [Fai11] and [KFS12], where the authors aim to find an automaton that accepts the given robot model and that is as close as possible to the original specification automaton. Conversely, in [BGK+11] the authors aim to modify the robot's model, rather than the mission specifications, so that the new robot model can satisfy the specifications. Also related is the work in [Hau14], where the author focuses on finding the smallest set of obstacles that need to be removed from the robot's environment in order to guarantee the existence of a collision-free trajectory from an initial state to a goal set.

This thesis is closest in nature to the work in [DF11], where the authors consider a transition system with its variables partitioned into a set of control inputs and a set of disturbances. The mission specifications are captured as an ordered set of LTL formulae, and the problem is to synthesize a control strategy that satisfies the first m formulae in the set regardless of the disturbances for the largest possible m. Also closely related is the work in [CY90], where the authors consider a Markov decision process and a set of Büchi automata. Here each automaton is associated with a reward, and the problem is to find a feedback policy that maximizes the cumulative reward of the infinite path of the system weighted by the probability that each automaton accepts the path. The authors' solution builds on translating the problem into a linear program, but unfortunately the time complexity of the algorithm proposed is exponential in the size of the specification automata.

Regarding our contributions, the setup in Chapter 3 involves a discrete model of a robotic system along with a set of mission specifications, including tasks that need to be fulfilled and rules that must be obeyed, which are given as a set of infinite LTL formulae. Each formula in the set has a reward associated with its satisfaction, and the problem is to synthesize a control strategy that maximizes the cumulative reward. Of course, this problem can be solved in a straightforward manner by finding, for each subset of the original set, a control strategy that satisfies all the formulae in the subset, if any such strategy exists. In contrast to this brute-force approach, we offer an alternative solution method which enjoys a worst-case computational complexity that is only slightly higher than that of synthesizing a control strategy satisfying all formulae in the original set.

Recalling the completeness issues associated with fixed discretizations, in Chapter 4 we combine ideas from automata-based formal verification and sampling-based motion planning algorithms to solve a similar problem for the case where the robot is a continuous-

time continuous-space dynamical system, which we achieve by incrementally abstracting the robot's dynamics. Here the specifications are given as an ordered set of finite $\text{LTL}_{-\mathsf{X}}$ formulae where each formula has the property that if a particular path violates it then the path cannot be extended into a path that satisfies the formula. For this purpose we construct a weighted automaton which allows us to quantify the 'level' of violation of the trajectories of the continuous system with respect to the specifications. We then propose an algorithm, which we have named MVRRT*, that asymptotically solves the problem of finding a trajectory of the dynamical system that *(i)* achieves the minimum violation among all trajectories that reach the goal region and that *(ii)* achieves the minimum cost among all trajectories that satisfy the first condition. We then argue that this algorithm offers the almost sure asymptotic optimality guarantee and that it is computationally efficient. Finally, we present and discuss the implementation of this algorithm in simulation and in a real-time experimental testbed.

Chapter 2

Background on Formal Verification

This chapter introduces several definitions and facts necessary for the understanding of the formal verification techniques utilized in Chapters 3 and 4 of this thesis.

This and the following chapters make use of the following fundamental notation. We use $\mathbb{N}$ to denote the set of all natural numbers, and we define $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. Furthermore, we let $\mathbb{R}$ denote the set of all natural numbers, and for any $n \in \mathbb{N}$ we let $\mathbb{R}^n$ denote the n-dimensional Euclidean space. Also, we use $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ to denote the set of all non-negative and strictly positive real numbers, respectively, and we let $\mathbb{R}^n_{\geq 0}$ and $\mathbb{R}^n_{>0}$ denote the non-negative and strictly positive orthant or $\mathbb{R}^n$, respectively. Finally, we use $\mathbb{S}$ to denote the unit sphere, *i.e.,* $\mathbb{S} = \left\{ (x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1 \right\}$.

2.1 Alphabets and Words

Any set of *symbols* constitutes and *alphabet*. If Π is a finite alphabet, we denote its cardinality by $|\Pi|$ and its power set, *i.e.,* the set of all of its subsets, by 2^{Π}. If $\pi \in \Pi$ then, with a slight abuse of notation, we let $\pi \subseteq 2^{\Pi}$ denote the set of all subsets of Π that contain symbol π. For instance, if $\Pi = \{ a, b, c\}$ then $a \equiv \{ \{a\}, \{a, b\}, \{a, c\}, \{a, b, c\} \}$. We specify subsets of 2^{Π} by using Propositional Calculus. In particular, if $q \subseteq 2^{\Pi}$ then $\neg\, q$ is defined as $2^{\Pi} \setminus q$, while if $q_1, q_2 \subseteq 2^{\Pi}$ then $q_1 \wedge q_2$ is defined as $q_1 \cap q_2$. As usual, disjuntion is defined so that $q_1 \vee q_2 \equiv \neg\, (\neg\, q_1 \wedge \neg\, q_2\,)$, while implication is defined so that $q_1 \rightarrow q_2 \equiv \neg\, q_1 \vee q_2$. Finally, we define the symbol `true` such that for any $q \subseteq 2^{\Pi}$ we have

that $\texttt{true} \in q$.

Given any alphabet Σ, such as 2^{Π}, for instance, the set of all *finite sequences of symbols* of Σ is denoted by Σ^*, and we refer to any of its elements as a *finite word* over Σ. We explicitly describe a finite word $\overline{\sigma}$ over Σ of length $n \in \mathbb{N}$ either by writing $\overline{\sigma} = \sigma_0, \ldots, \sigma_{n-1}$, or by writing $\overline{\sigma} = \sigma(0), \ldots, \sigma(n-1)$, where both σ_i and $\sigma(i)$ represent the i^{th} *letter* (*i.e.,* position) of word $\overline{\sigma}$. Note that Σ^* includes the empty word, *i.e.,* the zero-length sequence of symbols of Σ, which we will denote as ε. Since it is often useful to exclude the empty word, we define $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$, *i.e.,* Σ^+ is the set of all finite words of Σ of strictly positive length. If $L \subseteq \Sigma^*$, then we say that L is a *finite language* of Σ. Similarly, if $L \subseteq \Sigma^+$, then we say that L is a *positively finite language* of Σ.

The set of all *infinite sequences of symbols* of Σ is denoted by Σ^ω, and we refer to any of its elements as an *infinite word* over Σ. We explicitly describe an infinite word $\overline{\sigma}$ over Σ either by writing $\overline{\sigma} = \sigma_0, \sigma_1, \ldots$, or by writing $\overline{\sigma} = \sigma(0), \sigma(1), \ldots$, where both σ_i and $\sigma(i)$ represent the i^{th} *letter* (*i.e.,* position) of word $\overline{\sigma}$. If $L \subseteq \Sigma^\omega$, then we say that L is an *infinite language* of Σ.

The set of all *finite timed words* over Σ is the set of all finite words over the infinite alphabet $\Sigma \times \mathbb{R}_{\geq 0}$, and it is denoted as $\Sigma^*_{\geq 0}$. The set of all finite timed words over Σ of strictly positive length is denoted by $\Sigma^+_{\geq 0}$. We explicitly describe a finite timed word $\overline{\sigma}_\tau$ over Σ of length $n \in \mathbb{N}$ either by writing $\overline{\sigma}_\tau = (\sigma_0, \tau_0), \ldots, (\sigma_{n-1}, \tau_{n-1})$, or by writing $\overline{\sigma}_\tau = (\sigma(0), \tau(0)), \ldots, (\sigma(n-1), \tau(n-1))$, where both (σ_i, τ_i) and $(\sigma(i), \tau(i))$ represent the i^{th} *letter* (*i.e.,* position) of word $\overline{\sigma}_\tau$. If $L \subseteq \Sigma^*_{\geq 0}$, then we say that L is a *finite timed language* of Σ, while if $L \subseteq \Sigma^+_{\geq 0}$, then we say that L is a *positively finite timed language* of Σ.

Given a finite word σ and a finite or infinite word σ', both over Σ, we let $\sigma \cdot \sigma'$ denote the concatenation of σ and σ'. Concatenation of finite timed words is defined analogously. If $\sigma \in \Sigma^+$ then we let $\sigma^\omega \in \Sigma^\omega$ denote the infinite concatenation of σ with itself, *i.e.,* we define $\sigma^\omega = \sigma \cdot \sigma \cdot \sigma \cdots$.

Given a finite word $\overline{\sigma} = \sigma_0, \ldots, \sigma_{n-1} \in \Sigma^+$, let $\mathcal{I} = \{ i_0, i_1, \ldots, i_k \}$ be a set of indices such that $i_0 = 0$, such that $\sigma(i_j) = \sigma(i_j + 1) = \cdots = \sigma(i_{j+1} - 1) \neq \sigma(i_{j+1})$ for all $0 \leq j \leq k-1$, and such that $\sigma(i_k) = \sigma(i_{k+1}) = \cdots = \sigma(n-1)$. Then, we define the function $\texttt{destutter}\colon \Sigma^+ \mapsto \Sigma^+$ such that $\texttt{destutter}(\overline{\sigma}) = \sigma(i_0), \sigma(i_1), \ldots, \sigma(i_k)$. Intuitively, the `destutter` function replaces each maximal symbol repetition subsequence of word $\overline{\sigma}$ with one instance of the symbol associated with the subsequence. For example, if $\Sigma = \{ \varnothing, \pi \}$

and word $\overline{\sigma} \in \Sigma^+$ is such that

$$\overline{\sigma} = \varnothing,\, \pi,\, \varnothing,\, \varnothing,\, \pi,\, \pi,\, \varnothing,\, \varnothing,\, \varnothing,\, \pi,\, \pi,\, \pi$$

then

$$\texttt{destutter}(\overline{\sigma}) = \varnothing,\, \pi,\, \varnothing,\, \pi,\, \varnothing,\, \pi\,.$$

Similarly, given a finite timed word $\overline{\sigma}_\tau = (\sigma_0, \tau_0), \ldots, (\sigma_{n-1}, \tau_{n-1}) \in \Sigma^+_{\geq 0}$, let $\mathcal{I} = \{\, i_0, i_1, \ldots, i_k\}$ be a set of indices such that $i_0 = 0$, such that $\sigma(i_j) = \sigma(i_j + 1) = \cdots = \sigma(i_{j+1} - 1) \neq \sigma(i_{j+1})$ for all $0 \leq j \leq k-1$, and such that $\sigma(i_k) = \sigma(i_{k+1}) = \cdots = \sigma(n-1)$, and let $T = \{\, t_0, t_1, \ldots, t_k\}$ be a set of indices such that $t_j = \tau(i_j) + \tau(i_j + 1) + \cdots + \tau(i_{j+1} - 1)$ for all $0 \leq j \leq k-1$ and such that $t_k = \tau(i_k) + \tau(i_k + 1) + \cdots + \tau(n-1)$. Then, we define the function $\texttt{destutter}\colon \Sigma^+_{\geq 0} \mapsto \Sigma^+_{\geq 0}$ such that $\texttt{destutter}(\overline{\sigma}_\tau) = (\sigma(i_0), t_0), (\sigma(i_1), t_1), \ldots, (\sigma(i_k), t_k)$. For example, if $\Sigma = \{\, \varnothing,\, \pi\}$ and word $\overline{\sigma}_\tau \in \Sigma^+_{\geq 0}$ is such that

$$\overline{\sigma}_\tau = (\varnothing, 0.5),\, (\pi, 1.5),\, (\varnothing, 0.5),\, (\varnothing, 0.5),\, (\pi, 1.5),\, (\pi, 1.5)$$

then

$$\texttt{destutter}(\overline{\sigma}) = (\varnothing, 0.5),\, (\pi, 1.5),\, (\varnothing, 1.0),\, (\pi, 3.0)\,.$$

2.2 Dynamical System Models

This section presents two models of dynamical systems which facilitate formal verification of their feasible paths. The first system model, introduced in Definition 2.1, is utilized in Chapter 3 to describe the motion capabilities of a discrete-time discrete-space dynamical system, such as a robot operating sequentially within a finite state space.

Definition 2.1. A *Transition System* is a tuple $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi, \mathcal{L})$ where

- S is a finite set of states,
- $s_{init} \in S$ is the initial state,
- $\mathcal{R} \subseteq S \times S$ is a transition relation,
- Π is a finite alphabet, and
- $\mathcal{L}\colon S \mapsto 2^{\Pi}$ is a labeling function.

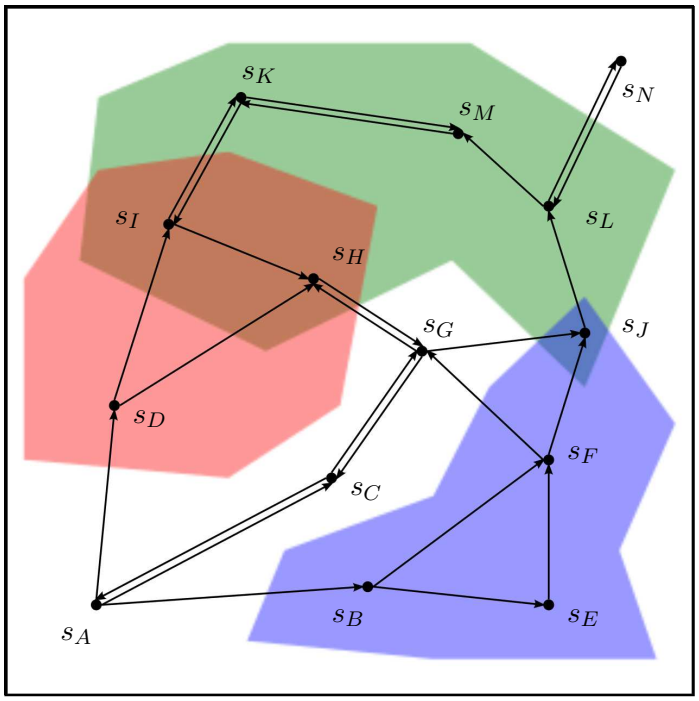


Figure 2-1: Transition system for Example 2.2.

Further, $\mathcal{T}$ is called *non-blocking* if for all states $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in \mathcal{R}$. Otherwise, we say that $\mathcal{T}$ is *blocking*. □

Notice that the triplet $(S, s_{init}, \mathcal{R})$ defines a directed graph with an initial vertex, so a transition system is nothing more than a directed graph extended with a pair $(\Pi, \mathcal{L})$. In particular, the labeling function $\mathcal{L}$ indicates which symbols of the alphabet Π hold at each vertex of the graph.

Example 2.2. Consider the transition system $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$ in Figure 2-1. Here $S = \{s_A, s_B, \ldots, s_N\}$ is a finite subset of the unit square in $\mathbb{R}^2$, initial state s_{init} is chosen to be state s_A, and $\mathcal{R}$ is such that $(s, s') \in \mathcal{R}$ if there is an arrow from s to s'. There are three types of regions: Blue, green and red. Hence $\Pi = \{blue, green, red\}$, and $\mathcal{L}(s)$ contains symbol *blue* if s is belongs to a blue region, symbol *green* if s belongs to a green region, and symbol *red* if s belongs to a red region, for each $s \in S$. Intuitively, $\mathcal{T}$ could serve as a model for a robot operating sequentially, such that the robot is at state s_{init} at the first step, and such that at each successive step if the robot is located in state $s \in S$ then in the next step it can reach any state $s' \in S$ such that $(s, s') \in \mathcal{R}$. □

For any transition system $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$, let $\Sigma = 2^{\Pi}$. Then a *finite path*

$\overline{s} = s_0, \ldots, s_{n-1}$ of $\mathcal{T}$ is a finite sequence of states in S such that if $n \geq 2$ then $(s_i, s_{i+1}) \in \mathcal{R}$ for all $0 \leq i \leq n-2$, and we say that finite path $\overline{s}$ is a *finite trace* of $\mathcal{T}$ if $s_0 = s_{init}$. A non-empty finite path $\overline{s} = s_0, \ldots, s_{n-1}$ of $\mathcal{T}$ *induces* a non-empty finite word $w(\overline{s}) = \ell_0, \ldots, \ell_{n-1} \in \Sigma^+$ such that $\ell_i = \mathcal{L}(s_i)$ for all $0 \leq i \leq n-1$, while the empty path of $\mathcal{T}$ induces the empty word. Similarly, an *infinite path* $\overline{s} = s_0, s_1, \ldots$ of $\mathcal{T}$ is an infinite sequence of states in S such that $(s_i, s_{i+1}) \in \mathcal{R}$ for all $i \in \mathbb{N}_0$, and we say that infinite path $\overline{s}$ is an *infinite trace* of $\mathcal{T}$ if $s_0 = s_{init}$. An infinite path $\overline{s} = s_0, s_1, \ldots$ of $\mathcal{T}$ *induces* an infinite word $w(\overline{s}) = \ell_0, \ell_1, \cdots \in \Sigma^\omega$ such that $\ell_i = \mathcal{L}(s_i)$ for all $i \in \mathbb{N}_0$.

Notice that we can use induced words to describe the behaviors of the paths of a transition system. For instance, in Example 2.2, requesting a trace $\overline{s}^*$ of $\mathcal{T}$ that never intersects a red region and eventually reaches the green region is equivalent to asking for a trace $\overline{s}^*$ such that word $w(\overline{s}^*)$ does not contain symbol *red* in any of its letters and such that word $w(\overline{s}^*)$ contains symbol *green* in at least one of its letters.

The second system model studied in this section, which is presented in Definition 2.3, is utilized in Chapter 4 in order to abstract a continuous-time continuous-space dynamical system, which is achieved by constructing a sequence of incrementally refined discrete-time discrete-space models.

Definition 2.3. A *Durational Transition System* is a tuple $\mathcal{D} = (\mathcal{T}, \Delta)$ where $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$ is a transition system and $\Delta \colon \mathcal{R} \mapsto \mathbb{R}_{\geq 0}$ is a function that maps each transition of $\mathcal{T}$ to a time duration. □

Example 2.4. Recall the robot model considered in Example 2.2. If the real robot has single-integrator dynamics (*i.e.,* it is a velocity-controlled vehicle), then we can build a durational transition system $\mathcal{D} = (\mathcal{T}, \Delta)$ by combining transition system $\mathcal{T}$ from Example 2.2 with the function Δ defined such that $\Delta(s, s')$ is equal to the minimum time required by the robot to travel from state s to state s', for each $(s, s') \in \mathcal{R}$. □

For any durational transition system $\mathcal{D} = (\mathcal{T}, \Delta)$ with $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$, let $\Sigma = 2^\Pi \times 2^\Pi$. Then a *finite path* of $\mathcal{D}$ is a finite path of $\mathcal{T}$, and a *finite trace* of $\mathcal{D}$ is a finite trace of $\mathcal{T}$. A finite path $\overline{s} = s_0, \ldots, s_{n-1}$ of $\mathcal{D}$ with $n \geq 2$ *induces* a timed word $w_\tau(\overline{s}) = (\ell_0, \tau_0), \ldots, (\ell_{n-2}, \tau_{n-2}) \in \Sigma^+_{\geq 0}$ such that $(\ell_i, \tau_i) = ((\mathcal{L}(s_i), \mathcal{L}(s_{i+1})), \Delta(s_i, s_{i+1}))$ for all $0 \leq i \leq n-2$. Notice that both the empty path and any path of $\mathcal{D}$ of length one induce the empty word. In addition, for any finite path $\overline{s} = s_0, \ldots, s_{n-1}$ of $\mathcal{D}$ with $n \geq 2$, we define the *duration* of $\overline{s}$, denoted $\Delta(\overline{s})$, such that $\Delta(\overline{s}) = \sum_{i=0}^{n-2} \Delta(s_i, s_{i+1})$. Note that the empty path and any path of $\mathcal{D}$ of length one have zero duration.

2.3 Non-deterministic Automata

This section studies a particular type of finite-state machines known as non-deterministic automata, which are useful in *verifying properties of paths* of transition systems. Specifically, we can use a non-deterministic automaton to prove or disprove that the word induced by a given path of a transition system belongs to a particular language, *e.g.,* in the context of Example 2.2, the language of all words that do not contain the symbol *red* in any of its letters. We begin by introducing the main concept in its most generic form.

Definition 2.5. A *Non-deterministic Automaton* is a tuple $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$ where

- Q is a finite set of states,
- $q_{init} \in Q$ is the initial state,
- Σ is an input alphabet,
- $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation, and
- AC is an acceptance condition.

Further, $\mathcal{A}$ is called *non-blocking* if for all states $q \in Q$ and all symbols $\sigma \in \Sigma$ there exists a state $q' \in Q$ such that $(q, \sigma, q') \in \delta$. Otherwise, we say that $\mathcal{A}$ is *blocking*. □

Given automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$, a *finite path* of $\mathcal{A}$ over a non-empty finite word $\overline{\sigma} = \sigma_0, \ldots, \sigma_{n-1} \in \Sigma^+$ is a non-empty finite sequence $\overline{q} = q_0, \ldots, q_n$ of states in Q such that $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $0 \leq i \leq n-1$. Note that a finite path of automaton $\mathcal{A}$ over the empty word is a trivial sequence consisting of only one state. Finite path $\overline{q} = q_0, \ldots, q_n$ of automaton $\mathcal{A}$ is said to be *simple* if $q_i = q_{i'}$ implies $i = i'$, for all $0 \leq i \leq n$. Simple finite path $\overline{c} = c_0, \ldots, c_n$ of automaton $\mathcal{A}$ is said to be a *cycle* if $(c_n, c_0) \in \delta$. Note that a finite path of automaton $\mathcal{A}$ over the empty word is always a simple path but never a cycle. Finally, a *finite run* of automaton $\mathcal{A}$ over a non-empty finite word $\overline{\sigma} \in \Sigma^+$ is a finite path of $\mathcal{A}$ over $\overline{\sigma}$ that starts at state q_{init}. Note that a run of automaton $\mathcal{A}$ over the empty word is the trivial sequence of states in Q consisting only of q_{init}.

Similarly, an *infinite path* of automaton $\mathcal{A}$ over an infinite word $\overline{\sigma} = \sigma_0, \sigma_1, \ldots \in \Sigma^\omega$ is an infinite sequence $\overline{q} = q_0, q_1, \ldots$ of states in Q such that $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}_0$. An *infinite run* of automaton $\mathcal{A}$ over an infinite word $\overline{\sigma} \in \Sigma^\omega$ is an infinite path of $\mathcal{A}$ over $\overline{\sigma}$ that starts at state q_{init}.

Remark 2.6. If we consider automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$, we realize that because

transition relation δ is non-deterministic, there may exist finitely many different paths of automaton $\mathcal{A}$ with the same starting state over the same finite word. Likewise, there may exist finitely or infinitely many different paths of automaton $\mathcal{A}$ with the same starting state over the same infinite word. □

Unlike the definition above, the interpretation of the acceptance condition AC of automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$ is not generic, but rather depends on the specific type of automaton under study. The three types of automata which are relevant to this thesis are introduced in the following definitions.

Definition 2.7. A *Non-deterministic Finite Automaton* is a non-deterministic automaton $\mathcal{A}_{\mathcal{F}} = (Q, q_{init}, \Sigma, \delta, AC)$ where $AC \subseteq Q$ is a set of *accepting states*. It takes as input finite words only, and it accepts a finite word $\overline{\sigma} \in \Sigma^*$ if and only if it has a finite run $\overline{q}$ over word $\overline{\sigma}$ such that the last state of $\overline{q}$ lies inside AC. Hence the set of all words accepted by automaton $\mathcal{A}_{\mathcal{F}}$ is a finite language of Σ, which we denote as $L(\mathcal{A}_{\mathcal{F}})$. □

Definition 2.8. A *Non-deterministic Büchi Automaton* is a non-deterministic automaton $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, AC)$ where $AC \subseteq Q$ is a set of *accepting states*. It takes as input infinite words only, and it accepts an infinite word $\overline{\sigma} \in \Sigma^{\omega}$ if and only if it has an infinite run $\overline{q}$ over word $\overline{\sigma}$ that intersects set AC infinitely many times. Hence the set of all words accepted by automaton $\mathcal{B}$ is an infinite language of Σ, which we denote as $L(\mathcal{B})$. □

Definition 2.9. A *Non-deterministic Generalized Büchi Automaton* is a non-deterministic automaton $\mathcal{G} = (Q, q_{init}, \Sigma, \delta, AC)$ where $AC \subseteq 2^Q$ is a set of *accepting sets*. It takes as input infinite words only, and it accepts an infinite word $\overline{\sigma} \in \Sigma^{\omega}$ if and only it has an infinite run $\overline{q}$ over word $\overline{\sigma}$ that intersects every set in AC infinitely many times. Hence the set of all words accepted by automaton $\mathcal{G}$ is an infinite language of Σ, which we denote as $L(\mathcal{G})$. □

Notice that if automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$ is blocking then we can easily construct a non-blocking automaton $\widetilde{\mathcal{A}} = (\widetilde{Q}, q_{init}, \Sigma, \widetilde{\delta}, AC)$ such that $L(\mathcal{A}) = L(\widetilde{A})$. In particular, this is achieved by defining $\widetilde{Q} = Q \cup \{q_{trap}\}$ and

$$\widetilde{\delta} = \delta \cup \left\{ (q, \sigma, q_{trap}) \in \widetilde{Q} \times \Sigma \times \widetilde{Q} : \nexists q' \in Q \text{ s.t. } (q, \sigma, q') \in \delta \right\}.$$

Let $\mathcal{T}$ be a transition system and $\mathcal{A}$ be an automaton with input alphabet $\Sigma = 2^{\Pi}$. If $\mathcal{A}$ is a finite automaton and $\overline{s}$ is a finite path of $\mathcal{T}$, then we say that path $\overline{s}$ is accepted by

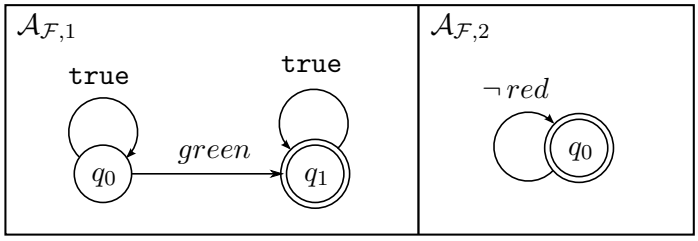


Figure 2-2: Finite automata for Example 2.10.

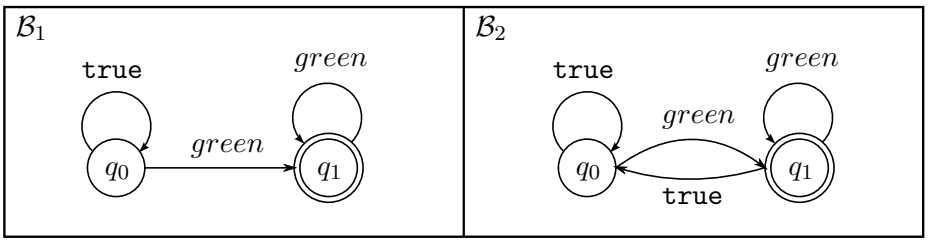


Figure 2-3: Büchi automata for Example 2.10.

automaton $\mathcal{A}$, which we denote as $\overline{s} \models \mathcal{A}$, if $w(\overline{s}) \in L(\mathcal{A})$. Similarly, if $\mathcal{A}$ is a Büchi or generalized Büchi automaton and $\overline{s}$ is an infinite path of $\mathcal{T}$, then we say that path $\overline{s}$ is accepted by automaton $\mathcal{A}$, which we denote as $\overline{s} \models \mathcal{A}$, if $w(\overline{s}) \in L(\mathcal{A})$.

We now provide concrete instances of finite and Büchi automata in the context of Example 2.2. We intentionally omit generalized Büchi automata from our example because as shown in Fact 2.11 they can be algorithmically translated into *language equivalent* Büchi automata.

Example 2.10. Recall transition system $\mathcal{T}$ from Example 2.2, and consider the finite automata $\mathcal{A}_{\mathcal{F},1}$ and $\mathcal{A}_{\mathcal{F},2}$ with input alphabet $\Sigma = 2^{\Pi}$ shown in Figure 2-2. In this figure, as well as in every other graphical representation of finite or Büchi automata in this thesis, we use circles to represent states, double circles to represent states belonging to the acceptance condition, and arcs to represent transitions. Regarding the symbols associated with the arcs, we use the notation established in Section 2.1, *e.g.,* for any $color \in \Pi \cup \{\texttt{true}\}$ we use $color \subseteq \Sigma$ to denote the set of all subsets of Π that contain symbol $color$, and we use propositional calculus to construct more complex sets.

Now, considering word $\overline{\sigma} = \sigma_0, \ldots, \sigma_{n-1} \in \Sigma^+$, we see that automaton $\mathcal{A}_{\mathcal{F},1}$ accepts $\overline{\sigma}$ if and only if $green \in \sigma_k$ for some $0 \leq k \leq n-1$, and we see that automaton $\mathcal{A}_{\mathcal{F},2}$ accepts $\overline{\sigma}$ if and only if $red \notin \sigma_k$ for all $0 \leq k \leq n-1$. (Notice that $\mathcal{A}_{\mathcal{F},2}$ also accepts ε.) Therefore, for any finite trace $\overline{s}$ of transition system $\mathcal{T}$, verifying that trace $\overline{s}$ eventually reaches the

green region is equivalent to checking that $\overline{s} \models \mathcal{A}_{\mathcal{F}_1}$. Likewise, verifying that trace $\overline{s}$ never enters the red region is equivalent to checking that $\overline{s} \models \mathcal{A}_{\mathcal{F},2}$.

Next, consider the Büchi automata $\mathcal{B}_1$ and $\mathcal{B}_2$ with input alphabet $\Sigma = 2^{\Pi}$ shown in Figure 2-3, and consider word $\overline{\sigma} = \sigma_0, \sigma_1, \ldots \in \Sigma^\omega$. Then automaton $\mathcal{B}_1$ accepts word $\overline{\sigma}$ if and only if there exists $j \in \mathbb{N}_0$ such that $green \in \sigma_k$ for all $k \geq j$, while automaton $\mathcal{B}_2$ accepts word $\overline{\sigma}$ if and only if for every $j \in \mathbb{N}_0$ there exist $k \geq j$ such that $green \in \sigma_k$. Consequently, for any infinite trace $\overline{s}$ of transition system $\mathcal{T}$, verifying that trace $\overline{s}$ eventually reaches the green region and then never leaves it is equivalent to checking that $\overline{s} \models \mathcal{B}_1$. Likewise, verifying that trace $\overline{s}$ intersects the green region infinitely many times is equivalent to checking that $\overline{s} \models \mathcal{B}_2$. □

Fact 2.11. Given a non-deterministic generalized Büchi automaton $\mathcal{G} = (Q_\mathcal{G}, q_{init,\mathcal{G}}, \Sigma, \delta_\mathcal{G}, F_\mathcal{G} = \{F_1, \ldots, F_m\})$, we can build a non-deterministic Büchi automaton $\mathcal{B} = (Q_\mathcal{B}, q_{init,\mathcal{B}}, \Sigma, \delta_\mathcal{B}, F_\mathcal{B})$ such that $L(\mathcal{G}) = L(\mathcal{B})$. Specifically, automaton $\mathcal{B}$ is constructed as follows.

- $Q_\mathcal{B} = Q_\mathcal{G} \times \{1, \ldots, m\}$.
- $q_{init,\mathcal{B}} = (q_{init,\mathcal{G}}, 1) \in Q_\mathcal{B}$.
- $((q, j), \sigma, (q', j')) \in \delta_\mathcal{B}$ if and only if $(q, \sigma, q') \in \delta_\mathcal{G}$ and either
  - $q \notin F_j$ and $j' = j$, or
  - $q \in F_j$ and $j' = (j \mod n) + 1$.
- $F_\mathcal{B} = F_1 \times \{1\}$.

□

To understand the intuition behind the construction in Fact 2.11, realize that set $Q_\mathcal{B}$ can be viewed as m copies or 'layers' of set $Q_\mathcal{G}$. Hence $F_\mathcal{B}$ is the 'projection' of F_1 onto the first layer. Thus, for an infinite run $\overline{q}$ of automaton $\mathcal{B}$ to be accepting, $\overline{q}$ needs to reach the projection of F_1 onto the first layer, then 'climb' and reach the projection of F_2 onto the second layer, and so on. Once run $\overline{q}$ has reached the projection of F_m onto the last layer, it must then 'descend' to the first layer and re-start the process that we just described. It follows that for any word $\overline{\sigma} \in \Sigma^\omega$ we have that $\overline{\sigma} \in L(\mathcal{B})$ if and only if there exists an infinite run of automaton $\mathcal{B}$ over word $\overline{\sigma}$ that intersects each of the sets $F_1 \times \{1\}$, $F_2 \times \{2\} \ldots, F_n \times \{n\}$ infinitely many times, which in turn occurs if and only if $\overline{\sigma} \in L(\mathcal{G})$.

Finally, we extend non-deterministic automata by associating weights with their transi-

tions. These constructions are key to the formal verification approaches to be developed in Chapters 3 and 4.

Definition 2.12. A *Weighted Non-deterministic Automaton* is a tuple $\mathcal{A}_\mathcal{W} = (\mathcal{A}, \mathcal{W})$, where $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, AC)$ is a non-deterministic automaton and $\mathcal{W}\colon \delta \mapsto \mathbb{R}_{\geq 0}$ is a function that maps transitions to weights. Further, for any finite path $\overline{q} = q_0, \ldots, q_n$ of automaton $\mathcal{A}_\mathcal{W}$ over word $\overline{\sigma} = \sigma_0, \ldots, \sigma_{n-1} \in \Sigma^+$ with $n \geq 1$, we define the *weight* of path $\overline{q}$, which we denote as $\mathcal{W}(\overline{q})$, such that $\mathcal{W}(\overline{q}) = \sum_{i=0}^{n-1} \mathcal{W}((q_i, \sigma_i, q_{i+1}))$. Note that the weight of any path of automaton $\mathcal{A}_\mathcal{W}$ over the empty word is zero. □

2.4 Formal Verification of Path Properties

Example 2.10 illustrates the usefulness of non-deterministic automata in verifying that a given trace of a transition system satisfies certain property. This motivates the following two *formal verification problems*.

Problem 2.13. Given a transition system $\mathcal{T}$ and a non-deterministic finite automaton $\mathcal{A}_\mathcal{F}$, does there exists a finite trace $\overline{s}$ of $\mathcal{T}$ such that $\overline{s} \models \mathcal{A}_\mathcal{F}$? If the answer is in the affirmative, return true and an example of such a trace. Otherwise, return false. □

Problem 2.14. Given a transition system $\mathcal{T}$ and a non-deterministic Büchi automaton $\mathcal{B}$, does there exists an infinite trace $\overline{s}$ of $\mathcal{T}$ such that $\overline{s} \models \mathcal{B}$? If the answer is in the affirmative, return true and an example of such a trace. Otherwise, return false. □

As it turns out, there exist relatively simple algorithms for solving these two problems. Such algorithms operate on the structure given in Definition 2.15.

Definition 2.15. Given a transition system $\mathcal{T} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L})$ and a non-deterministic finite or Büchi automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, the *product* of $\mathcal{T}$ and $\mathcal{A}$, denoted $\mathcal{T} \otimes \mathcal{A}$, is a non-deterministic finite or Büchi automaton $\mathcal{P} = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \Sigma_\mathcal{P}, \delta_\mathcal{P}, F_\mathcal{P})$, respectively. Specifically, $\mathcal{P}$ is defined as follows.

- $Q_\mathcal{P} = S \times Q$.
- $q_{init,\mathcal{P}} = (s_{init}, q_{init}) \in Q_\mathcal{P}$.
- $\Sigma_\mathcal{P} = \varnothing$.

- $((s, q), (s', q')) \in \delta_{\mathcal{P}}$ if and only if $(s, s') \in \mathcal{R}$ and $(q, \mathcal{L}(s), q') \in \delta$.
- $F_{\mathcal{P}} = S \times F$.

Further, the size of automaton $\mathcal{P}$, *i.e.,* the amount of memory required to store it, is equal to the sum of the cardinalities of $Q_{\mathcal{P}}$ and $\delta_{\mathcal{P}}$. □

Definition 2.16. Given a transition system $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi,\, \mathcal{L}\,)$ and a weighted non-deterministic finite or Büchi automaton $\mathcal{A}_{\mathcal{W}} = (\, \mathcal{A},\, \mathcal{W}\,)$ where $\mathcal{A} = (\, Q,\, q_{init},\, \Sigma,\, \delta,\, F\,)$, the *product* of $\mathcal{T}$ and $\mathcal{A}_{\mathcal{W}}$, denoted $\mathcal{T} \otimes \mathcal{A}_{\mathcal{W}}$, is a weighted non-deterministic finite or Büchi automaton $\mathcal{P}_{\mathcal{W}} = (\, \mathcal{P},\, \mathcal{W}_{\mathcal{P}})$, respectively, where $\mathcal{P} = (\, Q_{\mathcal{P}},\, q_{init,\mathcal{P}},\, \Sigma_{\mathcal{P}},\, \delta_{\mathcal{P}},\, F_{\mathcal{P}}\,)$ is the product of $\mathcal{T}$ and $\mathcal{A}$, and $\mathcal{W}_{\mathcal{P}}$ is a function defined such that if $(s, s') \in \mathcal{R}$ and $(q, \mathcal{L}(s), q') \in \delta$ then $\mathcal{W}_{\mathcal{P}}(\, ((s, q), (s', q'))\,) = \mathcal{W}(\, (q, \mathcal{L}(s), q')\,)$. □

With these definitions at hand, we realize that we can solve Problem 2.13 by searching for an accepting run of finite automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_{\mathcal{F}}$. This task can be executed efficiently using well-known graph search algorithms, such as Breadth-first Search (BFS) and Depth-first Search (DFS). For instance, the worst-case computational complexity of searching automaton $\mathcal{P}$ for an accepting run using either BFS or DFS is linear in the size of $\mathcal{P}$.

We also observe that we can solve Problem 2.14 by searching for an accepting run of Büchi automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$. However, this task is not as simple as the one involved in the previous problem, as it requires searching for a infinite run of automaton $\mathcal{P}$. Nevertheless, as shown in the following proposition, it turns out that automaton $\mathcal{P}$ has an accepting run if and only if it has an accepting run in *prefix-suffix structure*.

Proposition 2.17. Let $\mathcal{P} = (\, Q_{\mathcal{P}},\, q_{init,\mathcal{P}},\, \Sigma_{\mathcal{P}},\, \delta_{\mathcal{P}},\, F_{\mathcal{P}})$ be the product of a transition system and a non-deterministic Büchi automaton. There exists an accepting run of automaton $\mathcal{P}$ if and only if there exist finite paths $\overline{u} = u_0, \ldots, u_{m-1}$ and $\overline{v} = v_0, \ldots, v_{n-1}$ of $\mathcal{P}$ such that the infinite path $u \cdot (\, v^{\omega}\,)$ is an accepting run of $\mathcal{P}$. □

Proof: Assume there exists at least one accepting run of automaton $\mathcal{P}$, and let $\overline{q}$ be any such run. Since $|Q_{\mathcal{P}}|$ is finite and $F_{\mathcal{P}} \subseteq Q_{\mathcal{P}}$, there exists a state $q^* \in F_{\mathcal{P}}$ that appears infinitely many times in run $\overline{q}$. This leads to the following two observations:

- State q^* is reachable from state $q_{init,\mathcal{P}}$, which implies the existence of a path $\overline{u} = u_0, \ldots, u_{m-1}$ of automaton $\mathcal{P}$ such that $u_0 = q_{init,\mathcal{P}}$ and $(\, u_{m-1},\, q^*\,) \in \delta_{\mathcal{P}}$.
- State q^* is reachable from itself, which implies the existence of a cycle $\overline{v} = v_0, \ldots, v_{n-1}$ of automaton $\mathcal{P}$ such that $v_0 = v_{n-1} = q^*$.

Lastly, since $q^* \in F_{\mathcal{P}}$, we conclude that $u \cdot (v^\omega)$ is an accepting run of $\mathcal{P}$. ■

From Proposition 2.17, we realize that solving Problem 2.14 amounts to searching for a 'lasso-shaped' run of automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$ formed by the concatenation of a finite path from state $q_{init,\mathcal{P}}$ to some state $q^* \in F_{\mathcal{P}}$ with a cycle that begins and ends at state q^*. For this purpose, we can use a nested DFS, *i.e.,* a DFS within a DFS, such as the one given in Algorithm 2.1. Note that this algorithm, as well as those in the following chapters, make use of the following definitions.

- Function $successors\colon Q_{\mathcal{P}} \mapsto 2^{Q_{\mathcal{P}}}$, which is defined such that for any state $(s, q) \in Q_{\mathcal{P}}$ the set $successors((s, q))$ contains all states $(s', q') \in Q_{\mathcal{P}}$ such that transition $((s, q), (s', q'))$ belongs to $\delta_{\mathcal{P}}$.
- For any finite path $\overline{q}_{\mathcal{P}} = (s_0, q_0), \ldots, (s_{n-1}, q_{n-1})$ of automaton $\mathcal{P}$, the *projection* of $\overline{q}_{\mathcal{P}}$ onto S is the finite path $\overline{s} = s_0, \ldots, s_{n-1}$ of transition system $\mathcal{T}$.

Regarding guarantees, it is shown in [BK08] that this algorithm is correct and that its worst-case computational complexity is also linear in the size of automaton $\mathcal{P}$.

2.5 Linear Temporal Logic (LTL)

Although non-deterministic automata constitute a verification platform for a rich variety of desired behaviors of a system, it is difficult for humans to construct automata directly from natural language commands. For instance, we saw in Example 2.10 that automaton $\mathcal{A}_{\mathcal{F},1}$ verifies that a given trace of transition system $\mathcal{T}$ eventually enters the green region. Hence the reader may wonder how we came up with this automaton, assuming we started with the English-language command "eventually enter the green region". Motivated by this question, this section introduces finite and infinite Linear Temporal Logic (LTL), a calculus for specifying path properties of systems. This logic provides a user-friendly method for inputting desired behaviors in the form of formulae which can be later algorithmically translated into equivalent finite or Büchi automata.

We begin by introducing the *syntax* of the aforementioned calculus. In other words, the following definition establishes what is and what is not a formula.

Definition 2.18. Given a finite alphabet Π, a *Finite or Infinite Linear Temporal Logic Formula* over Π is defined inductively as follows.

- Symbol `true` is a formula.

Algorithm 2.1: `nested_DFS`

Input : Non-blocking transition system $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi,\, \mathcal{L}\,)$, non-blocking non-deterministic Büchi automaton $\mathcal{B} = (\, Q,\, q_{init},\, \Sigma,\, \delta,\, F\,)$ with $\Sigma = 2^{\Pi}$.

Output: If there exists a trace $\overline{s}$ of $\mathcal{T}$ in prefix-suffix structure such that $\overline{s} \models \mathcal{B}$, then true and an example of such a trace; otherwise, false.

```
1  construct P := T ⊗ B, where P := ( Q_P, q_{init,P}, Σ_P, δ_P, F_P);
2  initialize outer_set and inner_set to empty set;
3  initialize outer_stack and inner_stack to empty stack;
4  cycle_found := false;
5  outer_set := outer_set ∪ {q_{init,P}};
6  outer_stack.push(q_{init,P});
7  repeat
8      (s', q') := outer_stack.top();
9      if successors((s', q')) \ outer_set ≠ ∅ then
10         pick (s'', q'') ∈ successors((s', q')) \ outer_set;
11         outer_set := outer_set ∪ {(s'', q'')};
12         outer_stack.push((s'', q''));
13     else
14         outer_stack.pop();
15         if (s', q') ∈ F_P then cycle_found := cycle_check((s', q'))
16 until ( outer_stack.is_empty() ∨ cycle_found );
17 if cycle_found then
18     return true and the projection onto S of run
       reverse( concatenate( inner_stack, outer_stack) );
19 else
20     return false;
```

Algorithm 2.2: `cycle_check` (Executed within `nested_DFS`.)

Input : State $(s, q) \in F_{\mathcal{P}}$.
Output: True, if automaton $\mathcal{P}$ contains a cycle starting and ending at state (s, q); false, otherwise.

```
1  inner_set := inner_set ∪ {(s, q)};
2  inner_stack.push((s, q));
3  repeat
4      (s', q') := inner_stack.top();
5      if (s, q) ∈ successors((s', q')) then
6          inner_stack.push((s, q));
7          return true;
8      else
9          if successors((s', q')) \ inner_set ≠ ∅ then
10             pick (s'', q'') ∈ successors((s', q')) \ inner_set;
11             inner_set := inner_set ∪ {(s'', q'')};
12             inner_stack.push((s'', q''));
13         else
14             inner_stack.pop();
15 until inner_stack.is_empty();
16 return false;
```

- Symbol π is a formula, for any $\pi \in \Pi$.
- If ϕ is a formula, then $\neg\,\phi$ is a formula.
- If ϕ and ψ are formulae, then $\phi \wedge \psi$ is a formula.
- If ϕ is a formula, then $\mathsf{X}\,\phi$ is a formula.
- If ϕ and ψ are formulae, then $\phi\,\mathsf{U}\,\psi$ is a formula.

Further, we say that a formula is non-trivial if it is not equal to the symbol `true`. Moreover, the *size* or *length* of a finite or infinite LTL formula ϕ, denoted $|\phi|$, is defined inductively as follows.

- Formula `true` has length zero.
- Formula π has length one, for any $\pi \in \Pi$.
- If ϕ is a non-trivial formula, then $|\neg\,\phi| = 1 + |\phi|$.
- If ϕ and ψ are non-trivial formulae with $\phi \neq \psi$, then $|\phi \wedge \psi| = 1 + |\phi| + |\psi|$.
- If ϕ is a non-trivial formula, then $|\mathsf{X}\,\phi| = 1 + |\phi|$.
- If ϕ and ψ are nontrivial formulae with $\phi \neq \psi$, then $|\phi\,\mathsf{U}\,\psi| = 1 + |\phi| + |\psi|$.

□

Letting Π be any finite alphabet, notice that the Linear Temporal Logic over Π includes the propositional logic over Π, as its closed under conjunction and negation. Hence we can define the disjuntion and implication operators in the same way in which they are defined in propositional logic. On the other hand, the LTL over Π also includes two *temporal operators*, namely the X operator, pronounced 'next', and the U operator, pronounced 'until'. These operators allows us to specify properties of finite or infinite words over the alphabet 2^{Π}, and their intuitive interpretation is given in the following two definitions. Before continuing, however, we introduce two derived temporal operators which are quite useful in practice. In particular, the F operator, pronounced 'future', and the G operator, pronounced 'always', are defined as follows.

$$\mathsf{F}\,\phi \equiv \mathtt{true}\,\mathsf{U}\,\phi \tag{2.1}$$

$$\mathsf{G}\,\phi \equiv \neg\,(\,\mathsf{F}\,(\,\neg\,\phi\,)\,) \tag{2.2}$$

We now proceed to introduce the *semantics* of the Linear Temporal Logic. In other words, the following definitions establish what an LTL formula actually means.

Definition 2.19. Given a finite alphabet Π, a *Finite LTL Formula* over Π is equivalent to a finite language over Σ, where $\Sigma = 2^{\Pi}$. A finite LTL formula over Π is *interpreted* inductively as follows, where for any finite word $\sigma = \sigma_0, \ldots, \sigma_{n-1} \in \Sigma^*$ the expression $\sigma \models \phi$, pronounced 'σ satisfies ϕ', means that word $\overline{\sigma}$ belongs to the finite language associated with formula ϕ, which we denote as $L(\phi)$.

- $\sigma \models \texttt{true}$.
- $\sigma \models \pi$ for any $\pi \in \Pi$ if and only if $\pi \in \sigma_0$.
- $\sigma \models \neg\, \phi$ if and only if it is not the case that $\sigma \models \phi$.
- $\sigma \models \phi \wedge \psi$ if and only if $\sigma \models \phi$ and $\sigma \models \psi$.
- $\sigma \models \mathsf{X}\, \phi$ if and only if word $\sigma_1, \ldots, \sigma_{n-1}$ satisfies formula ϕ.
- $\sigma \models \phi\, \mathsf{U}\, \psi$ if and only if one of the following holds.
  - $\sigma \models \psi$.
  - There exists $0 \leq m \leq n-1$ such that for each $0 \leq k \leq m-1$ word $\sigma_k, \ldots, \sigma_{m-1}$ satisfies formula ϕ and such that word $\sigma_m, \ldots, \sigma_{n-1}$ satisfies formula ψ.

In addition, we say that finite timed word $\overline{\sigma}_\tau = (\sigma_0, \tau_0), \ldots, (\sigma_{n-1}, \tau_{n-1}) \in \Sigma^*_{\geq 0}$ satisfies formula ϕ, denoted $\overline{\sigma}_\tau \models \phi$, if and only if word $\sigma_0, \ldots, \sigma_{n-1} \in \Sigma^*$ satisfies ϕ. □

Definition 2.20. Given a finite alphabet Π, an *Infinite LTL Formula* over Π is equivalent to an infinite language over Σ, where $\Sigma = 2^{\Pi}$. An infinite LTL formula over Π is *interpreted* inductively as follows, where for any infinite word $\sigma = \sigma_0, \sigma_1, \ldots \in \Sigma^\omega$ the expression $\sigma \models \phi$, pronounced 'σ satisfies ϕ', means that word $\overline{\sigma}$ belongs to the infinite language associated with formula ϕ, which we denote as $L(\phi)$.

- $\sigma \models \texttt{true}$.
- $\sigma \models \pi$ for any $\pi \in \Pi$ if and only if $\pi \in \sigma_0$.
- $\sigma \models \neg\, \phi$ if and only if it is not the case that $\sigma \models \phi$.
- $\sigma \models \phi \wedge \psi$ if and only if $\sigma \models \phi$ and $\sigma \models \psi$.
- $\sigma \models \mathsf{X}\, \phi$ if and only if word $\sigma_1, \sigma_2, \ldots$ satisfies formula ϕ.
- $\sigma \models \phi\, \mathsf{U}\, \psi$ if and only if one of the following holds.
  - $\sigma \models \psi$.
  - There exists $m \in \mathbb{N}$ such that for each $0 \leq k \leq m-1$ word $\sigma_k, \ldots, \sigma_{m-1}$ satisfies formula ϕ and such that word $\sigma_m, \sigma_{m+1}, \ldots$ satisfies formula ψ.

□

With these definitions at hand, we can return to Expressions 2.1 and 2.2 in order to understand the interpretation of the derived temporal operators F and G.

Fact 2.21. Consider a finite alphabet Π and let $\Sigma = 2^{\Pi}$. If $\overline{\sigma} \in \Sigma^*$ is a finite word and ϕ is a finite LTL formula over Π, then we have the following semantics.

- $\overline{\sigma} \models \mathsf{F}\,\phi$ if and only if there exists $0 \leq k \leq n-1$ such that word $\sigma_k, \ldots, \sigma_{n-1}$ satisfies formula ϕ.
- $\overline{\sigma} \models \mathsf{G}\,\phi$ if and only if for all $0 \leq k \leq n-1$ word $\sigma_k, \ldots, \sigma_{n-1}$ satisfies formula ϕ.

In addition, if $\overline{\sigma} \in \Sigma^{\omega}$ is an infinite word and ϕ is an infinite LTL formula over Π, then we have the following semantics.

- $\overline{\sigma} \models \mathsf{F}\,\phi$ if and only if there exists $k \in \mathbb{N}_0$ such that word $\sigma_k, \sigma_{k+1}, \ldots$ satisfies formula ϕ.
- $\overline{\sigma} \models \mathsf{G}\,\phi$ if and only if for all $k \in \mathbb{N}_0$ word $\sigma_k, \sigma_{k+1}, \ldots$ satisfies formula ϕ.

□

Let $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi,\, \mathcal{L}\,)$ be a transition system. If $\overline{s}$ is a finite trace of $\mathcal{T}$ and ϕ is a finite LTL formula over Π, then with a slight abuse of notation we write $\overline{s} \models \phi$, pronounced '$\overline{s}$ satisfies ϕ', if and only if $w(\overline{s}) \models \phi$. Similarly, if $\overline{s}$ is an infinite trace of $\mathcal{T}$ and ϕ is an infinite LTL formula over Π, then we write $\overline{s} \models \phi$ if and only if $w(\overline{s}) \models \phi$.

Example 2.22. Recall the transition system $\mathcal{T}$ from Example 2.2, and consider the following finite and infinite LTL formulae over Π.

- $\mathsf{X}\,(\,\mathit{blue} \wedge \mathit{green}\,)$, which is satisfied by any trace for which its *second element* lies at the intersection of the blue and green region.
- $\mathsf{F}\,(\,\mathit{blue} \vee \mathit{green}\,)$, which is satisfied by any trace that *eventually reaches* the union of the blue and green regions.
- $\mathsf{F}\,(\,\mathit{blue} \wedge \mathsf{X}\,(\mathit{green})\,)$, which is satisfied by any trace that eventually reaches the blue region and then in the next step reaches the green region.
- $\mathsf{F}\,(\,\mathit{blue} \wedge \mathsf{F}\,(\mathit{green})\,)$, which is satisfied by any trace that eventually reaches the blue region and then eventually reaches the green region, *i.e.,* that eventually executes the ordered sequence consisting of the tasks *(i)* reaching the blue region and *(ii)* reaching the green region.
- $\mathsf{G}\,(\neg\, \mathit{red}\,)$, which is satisfied by any trace that *never enters*, the red region.

- $\mathsf{F}\,(\,\mathsf{G}\,(\mathit{blue})\,)$, which is satisfied by any finite trace that ends in the blue region, and by any infinite trace that eventually reaches the blue region and then always stays inside of the blue region.
- $\mathsf{G}\,(\,\mathsf{F}\,(\mathit{green})\,)$, which is satisfied by any finite trace that ends in the green region, and by any infinite trace that intersects the green region infinitely many times.

□

The examples above illustrate the straightforwardness with which a human user can specify the desired temporal properties of the trajectory of a system. This motivates the following analogs of Problems 2.13 and 2.14.

Problem 2.23. Given a transition system $\mathcal{T}$ and a finite LTL formula ϕ, does there exists a finite trace $\overline{s}$ of $\mathcal{T}$ that satisfies ϕ ? If the answer is in the affirmative, return true and an example of such a trace. Otherwise, return false. □

Problem 2.24. Given a transition system $\mathcal{T}$ and an infinite LTL formula ϕ, does there exists an infinite trace $\overline{s}$ of $\mathcal{T}$ that satisfies ϕ ? If the answer is in the affirmative, return true and an example of such a trace. Otherwise, return false. □

As mentioned before, both finite and infinite LTL formulae can be automatically translated into non-deterministic automata. More precisely, given a finite LTL formula ϕ, there exists algorithms, such as the one given in [GP02], capable of constructing a non-deterministic finite automaton $\mathcal{A}_\phi$ such that $L(\phi) = L(\mathcal{A}_\phi)$. Hence, Problem 2.23 can be readily solved by translating formula ϕ into finite automaton $\mathcal{A}_\phi$ and then solving Problem 2.13 with transition system $\mathcal{T}$ and automaton $\mathcal{A}_\phi$ as inputs.

Example 2.25. Recall the transition system $\mathcal{T}$ from Example 2.2 and the finite automata $\mathcal{A}_{\mathcal{F},1}$ and $\mathcal{A}_{\mathcal{F},2}$ from Example 2.10. Then finite LTL formula $\phi_1 = \mathsf{F}\,(\,\mathit{green}\,)$ is language equivalent to automaton $\mathcal{A}_{\mathcal{F},1}$, *i.e.*, $L(\phi_1) = L(\mathcal{A}_{\mathcal{F},1})$. Similarly, formula $\phi_2 = \mathsf{G}\,(\,\neg\,\mathit{red}\,)$ is language equivalent to automaton $\mathcal{A}_{\mathcal{F},2}$. □

Furthermore, given an infinite LTL formula ϕ, there exists algorithms, such as the ones given in [GPVW95] and [GO01], capable of constructing a non-deterministic generalized Büchi automaton $\mathcal{G}_\phi$ such that $L(\phi) = L(\mathcal{G}_\phi)$. Thus, we can solve Problem 2.24 by means of the following three step procedure. First, we translate formula ϕ into generalized Büchi automaton $\mathcal{G}_\phi$. Second, we apply Fact 2.11 to translate generalized Büchi automaton $\mathcal{G}_\phi$

into Büchi automaton $\mathcal{B}_\phi$. Lastly, we solve Problem 2.14 with transition system $\mathcal{T}$ and automaton $\mathcal{B}_\phi$ as inputs.

Example 2.26. Recall the transition system $\mathcal{T}$ from Example 2.2 and the Büchi automata $\mathcal{B}_1$ and $\mathcal{B}_2$ from Example 2.10. Then infinite LTL formula $\phi_1 = \mathsf{F}\,(\,\mathsf{G}\,(\mathit{green})\,)$ is language equivalent to automaton $\mathcal{B}_1$, *i.e.*, $L(\phi_1) = L(\mathcal{B}_1)$. Similarly, formula $\phi_2 = \mathsf{G}\,(\,\mathsf{F}\,(\mathit{green})\,)$ is language equivalent to automaton $\mathcal{B}_2$. □

Chapter 3

Least-violation Infinite LTL Path Planning

This chapter constitutes the first major contribution of this thesis. It presents an algorithm for the synthesis of least-violating trajectories of a discrete-time discrete-space dynamical system in the case where the specifications are given as infinite LTL formulae. It is organized as follows. Section 3.1 motivates and formally defines the problem of interest. Section 3.2 then describes our solution in detail, which consists combines a particular type of weighted Büchi automaton of our design with a special-purpose formal verification algorithm. Finally, Section 3.3 introduces an application of our problem formulation and then shows the result of applying our solution approach to this problem.

A preliminary version of this work has been published in [TRCK+13].

3.1 Problem Formulation

Let us consider a robot moving in a partitioned environment with its motion capabilities modeled as a transition system $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi,\, \mathcal{L}\,)$. Each region of the environment is modeled as a state of $\mathcal{T}$, and the robot's ability to move from region $s \in S$ to region $s' \in S$ is represented as a transition $(s, s') \in \mathcal{R}$. In case several controlled robots are placed in the environment, the states of the transition system encode the positions of all the robots in the environment. That is, if there are K robots, a state corresponds to a K-tuple of regions,

where the i^{th} element of the tuple is the region in which the i^{th} robot is located. Hence the transitions of $\mathcal{T}$ reflect the simultaneous motion capabilities of all the robots. The labeling function $\mathcal{L}\colon S \mapsto 2^{\Pi}$ maps each state to the set of symbols of finite alphabet Π that hold true at the state, such as, for instance, "vehicle v_i is in a safe region".

There is a set of high-level missions specifications to be satisfied by the robot or robots expressed as a set of infinite LTL formulae $\mathbf{\Phi} = \{\, \phi_1, \ldots, \phi_n \}$ over Π with the priorities of their satisfaction given by a reward function $rew\colon \mathbf{\Phi} \mapsto \mathbb{N}$. Here the value $rew(\phi_i)$ represents the reward earned whenever the trace of $\mathcal{T}$ associated with the motion of the robot satisfies formula ϕ_i. In other words, function rew allows us to compare the 'quality' of different traces of $\mathcal{T}$. This idea is further refined in the following definition.

Definition 3.1. The *reward* of a trace $\overline{s}$ of transition system $\mathcal{T}$, denoted $rew(\overline{s})$, is defined as follows.

$$rew(\overline{s}) = \sum_{\{\, \phi \in \Phi \,:\, \overline{s} \models \phi \,\}} rew(\phi) \tag{3.1}$$

□

We are now ready to formally state the problem of finding the 'best' trace of transition system $\mathcal{T}$, *i.e.,* the most compliant motion of the robot or robots with respect to the mission specifications given by set $\mathbf{\Phi}$ and function rew.

Problem 3.2. Given

- a transition system $\mathcal{T} = (\, S,\, s_{init},\, \mathcal{R},\, \Pi,\, \mathcal{L}\,)$,
- a set of infinite LTL formulas $\mathbf{\Phi} = \{\, \phi_1, \ldots, \phi_n \}$ over Π, and
- a reward function $rew : \mathbf{\Phi} \to \mathbb{N}$,

find a trace $\overline{s}^*$ of $\mathcal{T}$ such that $rew(\overline{s}^*) \geq rew(\overline{s})$ for any trace $\overline{s}$ of $\mathcal{T}$, *i.e.,* a trace $\overline{s}^*$ of $\mathcal{T}$ of maximum reward among all traces. □

Remark 3.3. Without loss of generality, from now on, we assume that $rew(\phi_i) \geq rew(\phi_j)$, for all $1 \leq i \leq j \leq n$. Hence if $rew(\phi_i) = 2^{n-i}$ for each formula $\phi_i \in \mathbf{\Phi}$, then the set $\mathbf{\Phi}$ is in fact ordered according to the Standard Lexicographic Ordering. In other words, for all $1 \leq i \leq n-1$ it is more rewarding (*i.e.,* important) to satisfy formula ϕ_i than to satisfy formula $\phi_{i+1} \wedge \cdots \wedge \phi_n$. □

A straightforward solution to Problem 3.2, which builds on the automata-based approach to

formal verification, is to consider every set $\widetilde{\mathbf{\Phi}} \in 2^{\mathbf{\Phi}}$ and to search for a trace of $\mathcal{T}$ satisfying each formula in set $\widetilde{\mathbf{\Phi}}$, if such a trace exists. Each one of these searches, which as a matter of fact solves Problem 2.14, can be executed using a formal verification algorithm such as nested DFS, *i.e.,* Algorithm 2.1. A trace $\overline{s}^*$ maximizing function rew among the ones found maps to the desired robot path. However, this brute-force solution is not efficient as it requires solving up to 2^n formal verification problems.

In this chapter, we suggest a method to alleviate the high computational demand of this straightforward solution. The main idea also develops on the automata-based approach to formal verification, but it relies on the construction of one weighted Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$ which is language-equivalent to the disjunction of all formulae in set $\mathbf{\Phi}$ and which captures the reward associated with each of the formulae in Φ through its weights. Then, the product $\mathcal{P}^*_{\mathcal{W}} = \mathcal{T} \otimes \mathcal{B}^*_{\mathcal{W}}$ is constructed and an reward-maximizing run of automaton $\mathcal{P}^*_{\mathcal{W}}$ is sought using a modification of the nested DFS algorithm. The proposed search algorithm has a worst-case computational complexity that is only slightly worse than that of the original nested DFS. Thus, instead of carrying up to 2^n nested DFS calls, we perform only one execution of a special-purpose formal verification algorithm.

3.2 Problem Solution

This section describes our solution to Problem 3.2 in detail. First, in Subsection 3.2.1, we present the construction of the weighted Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$ associated with the set of infinite LTL formulae $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$ and the reward function $rew\colon \mathbf{\Phi} \mapsto \mathbb{N}$, and we establish some useful properties of $\mathcal{B}^*_{\mathcal{W}}$ as well as of the automaton $\mathcal{P}^*_{\mathcal{W}} = \mathcal{T} \otimes \mathcal{B}^*_{\mathcal{W}}$. Second, in Subsection 3.2.2 the special-purpose nested DFS is described. Finally, in Subsection 3.2.3 we discuss the correctness and complexity of the proposed algorithm.

3.2.1 Design of the Weighted Automata

As discussed at the end of Section 2.5, each formula $\phi \in \mathbf{\Phi}$ can be automatically translated into a non-deterministic Büchi automaton $\mathcal{B}_\phi$ such that $L(\phi) = L(\mathcal{B}_\phi)$. Therefore, instead of working with set of formulae $\mathbf{\Phi}$ and function $rew\colon \mathbf{\Phi} \mapsto \mathbb{N}$, we consider the set of Büchi automata $\boldsymbol{\mathcal{B}} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ and the function $rew\colon \boldsymbol{\mathcal{B}} \mapsto \mathbb{N}$ such that automaton $\mathcal{B}_i \in \boldsymbol{\mathcal{B}}$ is language equivalent to formula $\phi_i \in \mathbf{\Phi}$ and $rew(\mathcal{B}_i) = rew(\phi_i)$, for each $1 \leq i \leq n$. In light of the fact that blocking automata can be trivially augmented into non-blocking

automata, we assume without loss of generality that each automaton in set $\boldsymbol{\mathcal{B}}$ is non-blocking. We now present the construction of the weighted Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$ associated with the pair $(\boldsymbol{\mathcal{B}}, rew)$, the design of which is inspired by the translation of generalized Büchi automata into Büchi automata given in Fact 2.11.

Definition 3.4. Given a set of non-deterministic Büchi automata $\boldsymbol{\mathcal{B}} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ such that $\mathcal{B}_i = (Q_i, q_{init,i}, \Sigma, \delta_i, F_i)$ for each $1 \leq i \leq n$ and a reward function $rew\colon \boldsymbol{\mathcal{B}} \mapsto \mathbb{N}$, the *weighted product associated with the pair* $(\boldsymbol{\mathcal{B}}, rew)$ is the weighted non-deterministic Büchi automaton $\mathcal{B}^*_{\mathcal{W}} = (\mathcal{B}^*, \mathcal{W})$, where $\mathcal{B}^* = (Q, q_{init}, \Sigma, \delta, F)$ and $\mathcal{W}\colon \delta \mapsto \mathbb{N}_0$. In particular, automaton $\mathcal{B}^*_{\mathcal{W}}$ is defined as follows.

- $Q = Q_1 \times \cdots \times Q_n \times \{0, 1, \ldots, n\}$.
- $q_{init} = (q_{init,1}, \ldots, q_{init,n}, 0) \in Q$.
- $t = ((q_1, \ldots, q_n, j), \sigma, (q'_1, \ldots, q'_n, j')) \in \delta$ if and only if $(q_i, \sigma, q'_i) \in \delta_i$ for each $1 \leq i \leq n$ and one of the following holds.
  - $j = 0$ and $j' = 0$. In this case, $\mathcal{W}(t) = 0$.
  - $j = 0$ and $j' > 0$. In this case, $\mathcal{W}(t) = rew(\mathcal{B}_{j'})$.
  - $j > 0$ and $j' = 0$ and $q_j \in F_j$. In this case, $\mathcal{W}(t) = 0$.
  - $j > 0$ and $j' = j$ and $q_j \notin F_j$. In this case, $\mathcal{W}(t) = 0$.
  - $j > 0$ and $j' > j$ and $q_j \in F_j$. In this case, $\mathcal{W}(t) = rew(\mathcal{B}_{j'})$.
- $F = Q_1 \times Q_2 \times \cdots Q_n \times \{0\}$.

In addition, we define the function $layer\colon Q \mapsto \{0, 1, \ldots, n\}$ such that for every state $q = (q_1, \ldots, q_n, j) \in Q$ we have that $layer(q) = j$. □

Loosely speaking, the set of states of automaton $\mathcal{B}^*_{\mathcal{W}}$ can be viewed as $n+1$ copies or *layers* of the set $Q_1 \times \cdots \times Q_n$. Layer zero, which contains the initial state and is equal to the set of accepting states of the automaton, is connected to itself and to each of the other layers. Any transition from layer zero to itself has zero weight, while any transition from layer zero to layer j' has weight equal to $rew(\mathcal{B}_{j'})$, for each $1 \leq j' \leq n$. Furthermore, for each $1 \leq j \leq n$, layer j is connected to itself through transitions of weight zero, and it is connected to layer zero through transitions of weight zero leaving from $Q_1 \times \cdots \times F_j \times \cdots \times Q_n \times \{j\}$. Also, for each $1 \leq j \leq n-1$, layer j is connected to each layer $j' > j$ through transitions leaving from $Q_1 \times \cdots \times F_j \times \cdots \times Q_n \times \{j\}$ which have weight equal to $rew(\mathcal{B}_{j'})$.

Notice that for all sets $\widetilde{\boldsymbol{\Phi}} \in 2^{\boldsymbol{\Phi}}$ automaton $\mathcal{B}^*_{\mathcal{W}}$ accepts all infinite words that satisfy each

formula in $\widetilde{\mathbf{\Phi}}$. Hence the weights associated with the transitions connecting different layers of automaton $\mathcal{B}^*_{\mathcal{W}}$ determine the 'quality' of an infinite run of $\mathcal{B}^*_{\mathcal{W}}$ over an infinite word, *i.e.,* they capture the reward of the formulae satisfied by the word. More precisely, notice that an infinite word $\overline{\sigma} \in \Sigma^\omega$ satisfies formula ϕ_j if and only if it is accepted by automaton $\mathcal{B}_j$, which occurs if and only if there exists an accepting run of $\mathcal{B}_j$ over $\overline{\sigma}$, which occurs if and only if there exists a run $\overline{q}^*_{\mathcal{W}}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ over $\overline{\sigma}$ that enters and exits the j^{th} layer infinitely many times. In this case, run $\overline{q}^*_{\mathcal{W}}$ contains infinitely many transitions of automaton $\mathcal{B}^*_{\mathcal{W}}$ with weight equal to $rew(\mathcal{B}_j)$.

We provide formal statements and proofs of the previous intuitive arguments. For this purpose, we first introduce two necessary definitions.

Definition 3.5. Let automaton $\mathcal{B}^*_{\mathcal{W}}$ be as in Definition 3.4. Let $\overline{q} = q_0, \ldots, q_n$ be a finite path of automaton $\mathcal{B}_{\mathcal{W}}$ over some finite word $\overline{\sigma}$. Then we say that a sub-path $\overline{q}_{frag} = q_i, \ldots, q_k$ of path $\overline{q}$ is a *fragment* of $\overline{q}$ if $q_i, q_k \in F$ and if $q_j \notin F$ for each $i < j < k$. In other words, a fragment of path $\overline{q}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ is a sub-path of $\overline{q}$ that starts and ends at an accepting state but that does not include any accepting states in between. Similarly, if $\overline{q} = q_0, q_1, \ldots$ is an infinite path of automaton $\mathcal{B}^*_{\mathcal{W}}$ over some infinite word then we say that a finite sub-path $\overline{q}_{frag} = q_i, \ldots, q_k$ of path $\overline{q}$ is a *fragment* of $\overline{q}$ if $q_i, q_k \in F$ and if $q_j \notin F$ for each $i < j < k$. Lastly, for any finite or infinite path $\overline{q}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$, we let $\mathsf{Frag}(\overline{q})$ denote the set of all of its fragments. □

Definition 3.6. Let automaton $\mathcal{B}^*_{\mathcal{W}}$ be as in Definition 3.4. The *reward* of an infinite path $\overline{q}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$, which with a slight abuse of notation we denote as $rew(\overline{q})$, is defined as follows.

$$rew(\overline{q}) = \max\{\, C \in \mathbb{N}_0 : \text{ there exists infinitely many } \overline{u} \in \mathsf{Frag}(\overline{q}) \text{ s.t. } \mathcal{W}(\overline{u}) = C \,\}$$

□

With these definitions at hand, we now present the central result of this subsection.

Proposition 3.7. Consider set $\widetilde{\mathbf{\Phi}} \in 2^{\mathbf{\Phi}}$, and suppose infinite word $\overline{\sigma} \in \Sigma^\omega$ satisfies every formula in set $\widetilde{\mathbf{\Phi}}$ and violates, *i.e.,* does not satisfy, every formula in set $\mathbf{\Phi} \setminus \widetilde{\mathbf{\Phi}}$. There exists an accepting run $\overline{q}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ over word $\overline{\sigma}$ such that

$$rew(\overline{q}) = \sum_{\phi \in \widetilde{\mathbf{\Phi}}} rew(\,\phi\,)\,.$$

Furthermore, for every run $\overline{q}'$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ over word $\overline{\sigma}$ it holds that

$$rew(\overline{q}') \leq \sum_{\phi \in \widetilde{\mathbf{\Phi}}} rew(\,\phi\,)\,.$$

□

Proof: For the first part of the proof, let the ordered index set $\mathcal{I} = \{\, i_1, \ldots, i_j \,\}$ be such that $\widetilde{\mathbf{\Phi}} = \{\, \phi_i \in \mathbf{\Phi} : \; i \in \mathcal{I} \,\}$. Since word $\overline{\sigma} \in \Sigma^\omega$ satisfies formula ϕ_i if and only if $\overline{\sigma} \in L(\mathcal{B}_i)$, then for each $i \in \mathcal{I}$ there exists an accepting run of automaton $\mathcal{B}_i$ over word $\overline{\sigma}$. Recalling the construction of automaton $\mathcal{B}^*_{\mathcal{W}}$ in Definition 3.4, we can convince ourselves of the existence of an accepting run $\overline{q}$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ over word $\overline{\sigma}$ such that every fragment $\overline{u}$ of run $\overline{q}$ has the following structure.

$$\overline{u} = u(0),\, u(\ell_1),\, \ldots, u(\ell'_1),\, u(\ell_2),\, \ldots,\, u(\ell'_{j-1}),\, u(\ell_j),\, \ldots,\, u(\ell'_j),\, u(\ell_0)$$

Here, the ordered set of indices $\{\, \ell_1,\, \ell'_1,\, \ldots,\, \ell_j,\, \ell'_j \,\}$ is such that the following constraints on fragment $\overline{u} \in \mathsf{Frag}(\overline{q})$ apply.

$$\begin{aligned}
layer(u(0)) &= 0 \\
layer(u(\ell_1)) &= \cdots = layer(u(\ell'_1)) = i_1 \\
layer(u(\ell_2)) &= \cdots = layer(u(\ell'_2)) = i_2 \\
&\vdots \\
layer(u(\ell_{j-1})) &= \cdots = layer(u(\ell'_{j-1})) = j-1 \\
layer(u(\ell_j)) &= \cdots = layer(u(\ell'_j)) = j \\
layer(u(\ell_0)) &= 0
\end{aligned}$$

It follows that $\mathcal{W}(\overline{u}) = \sum_{i \in \mathcal{I}} rew(\phi_i)$. In turn, noticing from Definition 3.6 that $\mathcal{W}(\overline{v}) \leq rew(\overline{q})$ for every fragment $\overline{v}$ of run $\overline{q}$ that appears infinitely many times in $\overline{q}$, we realize that $\sum_{i \in \mathcal{I}} rew(\phi_i) \leq rew(\overline{q})$.

For the second part of the proof, assume there exists an accepting run $\overline{q}'$ of automaton $\mathcal{B}^*_{\mathcal{W}}$ over word $\overline{\sigma}$ such that $rew(\overline{q}') > \sum_{i \in \mathcal{I}} rew(\phi_i)$. From Definition 3.6, there exists infinitely many fragments $\overline{u} \in \mathsf{Frag}(\overline{q}')$ such that $\mathcal{W}(\overline{u}) > \sum_{i \in \mathcal{I}} rew(\phi_i)$. Hence there exists $\widehat{i} \notin \mathcal{I}$ such that infinitely many fragments $\overline{u} \in \mathsf{Frag}(\overline{q}')$ enter and exit layer $\widehat{i}$, which from the construction of automaton $\mathcal{B}^*_{\mathcal{W}}$ and our hypothesis on run $\overline{q}'$ means that the sequence of $\hat{i}^{\text{th}}$ components of run $\overline{q}'$ is an accepting run of automaton $\mathcal{B}_{\widehat{i}}$ over word $\overline{\sigma}$. However, word $\overline{\sigma}$ does not satisfy any formula in the set $\mathbf{\Phi} \setminus \widetilde{\mathbf{\Phi}}$. ■

We now study the product of transition system $\mathcal{T}$ and weighted Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$, which is the structure on which we perform the search for the most rewarding trace of $\mathcal{T}$. We begin with two formal arguments.

Proposition 3.8. Let non-deterministic Büchi automaton $\mathcal{P}^*_{\mathcal{W}} = (\mathcal{P}, \mathcal{W}_{\mathcal{P}})$ be the product of the transition system $\mathcal{T}$ and the weighted non-deterministic Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$ from Definition 3.4, where $\mathcal{P} = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$. If $\overline{s}$ is an infinite trace of system $\mathcal{T}$ then there exists an infinite run $\overline{q}$ of automaton $\mathcal{P}$ such that $rew(\overline{s}) = rew(\overline{q})$. Furthermore, if $\overline{q}'$ is any other infinite run of automaton $\mathcal{P}$ such that its projection onto the set of states of system $\mathcal{T}$ is also trace $\overline{s}$, then $rew(\overline{q}') \leq rew(\overline{s})$. □

Proof: For each infinite trace $\overline{s}$ of system $\mathcal{T}$ inducing an infinite word $\overline{\sigma} = \sigma_0, \sigma_1, \ldots \in \Sigma^\omega$ accepted by automaton $\mathcal{B}^*_{\mathcal{W}}$ there exists an infinite run $\overline{q} = q_0, q_1, \ldots$ of $\mathcal{B}^*_{\mathcal{W}}$ over $\overline{\sigma}$ that intersects set F infinitely many times. From Definition 2.16, there exists an infinite run $\overline{u} = u_1, u_2, \ldots$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ such that $\mathcal{W}((u_i, u_{i+1})) = \mathcal{W}((q_i, \sigma_i, q_{i+1}))$ for all $i \in \mathbb{N}_0$ and such that state $u_i \in F_{\mathcal{P}}$ if and only if state $q_i \in F$ for all $i \in \mathbb{N}_0$. The result then follows from Proposition 3.7. ■

Proposition 3.9. Let weighted automaton $\mathcal{P}^*_{\mathcal{W}} = (\mathcal{P}, \mathcal{W}_{\mathcal{P}})$ be as in Proposition 3.8, where $\mathcal{P} = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$, and let $\overline{q}$ be an infinite run of $\mathcal{P}^*_{\mathcal{W}}$. Then there exists an infinite run $\overline{q}'$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ in prefix-suffix structure such that $rew(\overline{q}) = rew(\overline{q}')$. □

Proof: The argument is similar to that given in the proof of Proposition 2.17. Since set $Q_{\mathcal{P}}$ is finite and $\overline{q}$ is an infinite run of automaton $\mathcal{P}^*_{\mathcal{W}}$, there exists a state $q^* \in Q_{\mathcal{P}}$ that appears infinitely many times in run $\overline{q}$. This leads to the following two observations.

- State q^* is reachable from state $q_{init,\mathcal{P}}$, which implies the existence of a path $\overline{u} = u_0, \ldots, u_{m-1}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ such that $u_0 = q_{init,\mathcal{P}}$ and $(u_{m-1}, q^*) \in \delta_{\mathcal{P}}$.
- State q^* is reachable from itself, which implies the existence of
  - a fragment $\overline{v}_{out} \in \mathsf{Frag}(\overline{q})$ beginning at state $q^* \in F_{\mathcal{P}}$, ending at a state $q^{\mathsf{out-end}} \in F_{\mathcal{P}}$, and such that $\mathcal{W}(\overline{v}_{out}) = rew(\overline{q})$, along with
  - a path $\overline{v}_{in}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ beginning at a state $q^{\mathsf{in-start}} \in Q_{\mathcal{P}}$ such that $(q^{\mathsf{out-end}}, q^{\mathsf{in-start}}) \in \delta_{\mathcal{P}}$, ending at a state $q^{\mathsf{in-end}} \in Q_{\mathcal{P}}$ such that $(q^{\mathsf{in-end}}, q^*) \in \delta_{\mathcal{P}}$, and such that $\mathcal{W}(\overline{v}_{in}) \leq \mathcal{W}(\overline{v}_{out})$,

  such that path $\overline{v} = \overline{v}_{out} \cdot \overline{v}_{in}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ is a cycle.

Finally, letting infinite run $\overline{q}' = \overline{u} \cdot (\overline{v}^{\,\omega})$ we conclude that $rew(\overline{q}') = rew(\overline{q})$. ■

The last three propositions provide us with guidance on computing a trace of transition system $\mathcal{T}$ of maximum reward. Specifically, it is enough to compute a run of automaton $\mathcal{P}^*_{\mathcal{W}}$ of maximum reward in prefix-suffix structure and then project this run onto the set of states of system $\mathcal{T}$.

Theorem 3.10. Consider set $\widetilde{\mathbf{\Phi}} \in 2^{\mathbf{\Phi}}$ and let automaton $\mathcal{P}^*_{\mathcal{W}}$ be as in Proposition 3.8. Consider an infinite trace $\overline{s} = s_0, s_1, \ldots$ of transition system $\mathcal{T}$ that satisfies every formula in set $\widetilde{\mathbf{\Phi}}$ and that violates every formula in set $\mathbf{\Phi} \setminus \widetilde{\mathbf{\Phi}}$. Then there exists an infinite accepting run $\overline{q} = (s_0, q_0), (s_1, q_1), \ldots$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ in prefix-suffix structure such that $rew(\overline{q}) = \sum_{\phi \in \widetilde{\mathbf{\Phi}}} rew(\phi)$. □

Proof: Directly from the combination of Propositions 3.7, 3.8 and 3.9. ■

The problem thus reduces to searching for a cycle $\overline{c}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ beginning at a state of set $F_{\mathcal{P}}$ that maximizes the value of $rew(\overline{c})$, which equals to $\max_{\overline{q} \in \mathsf{Frag}(\overline{c})} \mathcal{W}(\overline{q})$, among all reachable cycles. With this in mind, the following proposition helps narrow down the search even further by showing that it is enough to search for a particular type of cycle.

Proposition 3.11. Let automaton $\mathcal{P}^*_{\mathcal{W}}$ be as in Proposition 3.8, and let $\overline{c}$ be a cycle of the automaton. If $\overline{q} = q_i, \ldots, q_k$ is a fragment of cycle $\overline{c}$ then there exists a simple finite path $\overline{q}' = q_k, \ldots, q_i$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ such that $\mathcal{W}(\, q_k, \ldots, q_i \,) = 0$. □

Proof: From Definition 3.4 we see that if there exists a simple finite path of automaton $\mathcal{B}^*_{\mathcal{W}}$ from state $q_{\mathcal{B}} = (\, q_1, \ldots, q_n, 0) \in Q$ to state $q'_{\mathcal{B}} = (\, q'_1, \ldots, q'_n, j) \in Q$ then there exists a simple finite path from state $q_{\mathcal{B}}$ to state $q''_{\mathcal{B}} = (\, q'_1, \ldots, q'_n, 0) \in Q$ such that if state $q \in Q$ belongs to the path then $layer(q) = 0$. The reward of the latter path is zero, and the result then follows by extending this argument to the paths of automaton $\mathcal{P}^*_{\mathcal{W}}$. ■

Thus we conclude that it is enough to search for a reachable cycle $\overline{c}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ in prefix-suffix structure maximizing the value of $rew(\overline{c})$ such that $\mathcal{W}(\overline{u}) > 0$ for some fragment $\overline{u} \in \mathsf{Frag}(\overline{c})$ and $\mathcal{W}(\overline{v}) = 0$ for all other fragments $\overline{v} \in \mathsf{Frag}(\overline{c})$. Hence without loss of generality we can restrict our attention to the type of cycle that has all of its weight concentrated on the first fragment.

Theorem 3.12. Consider set $\widetilde{\mathbf{\Phi}} \in 2^{\mathbf{\Phi}}$ and let automaton $\mathcal{P}^*_{\mathcal{W}}$ be as in Proposition 3.8. Consider an accepting run $\overline{q}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ such that $\overline{q} = \overline{u} \cdot (\, \overline{c}^{\,\omega} \,)$, where $\overline{u} = u_0, \ldots, u_{m-1}$ is a finite run and $\overline{c} = c_0, \ldots, c_{n-1}$ is a cycle with the following properties.

- $(\, u_{m-1},\, c_0\,) \in \delta_{\mathcal{P}}$.
- $c_0 \in F_{\mathcal{P}}$.
- If path $\overline{v} = c_i, \ldots, c_k$ is the first fragment of cycle $\bar{c}$ then $\mathcal{W}(\overline{v}') = 0$ for all other fragments $\overline{v}' \in \mathsf{Frag}(\bar{c}) \setminus \{\overline{v}'\}$.

If $rew(\overline{q}) \geq rew(\overline{q}')$ for any accepting run $\overline{q}'$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ with the properties described above then a solution to Problem 3.2, *i.e.,* a trace of transition system $\mathcal{T}$ of maximum reward, can be obtained as the projection of run $\overline{q}$ onto the set of states of system $\mathcal{T}$. □

3.2.2 Weighted Nested DFS

Accepting runs such as the one described in Theorem 3.12 can be found by adapting the standard nested DFS algorithm, as we show in this subsection. Specifically, we search for the concatenation of a finite run followed by a cycle satisfying the conditions of the aforementioned theorem. The solution to this problem is summarized in Algorithm 3.1.

To understand this algorithm, let us first focus on a solution of the following sub-problem: Given a state $q_f \in F_{\mathcal{P}}$, find a cycle of maximum reward among all cycles that begin and end in state q_f. A modification of the Breadth-first Search (BFS) algorithm described in Algorithm 3.2 can be used to do so $\mathcal{O}(\,|\mathcal{P}^*_{\mathcal{W}}|\,)$ time and space because the 'layers' of automaton $\mathcal{P}^*_{\mathcal{W}}$ connected through positive-weight transitions form a directed acyclic graph. Intuitively, the algorithm systematically searches automaton $\mathcal{P}^*_{\mathcal{W}}$ while maintaining, for each state $q \in Q_{\mathcal{P}}$, an approximation of the maximum reward of any simple path from state q_f to state q. This is the purpose of the *dist* and *pred* member values of each state.

The correctness of the algorithm hinges on the fact that when state $q \in F_{\mathcal{P}}$ is processed in Line 2 of Algorithm 3.3, the value of $q.dist$ is set to the actual maximum reward of any simple path from state q_f to state q. Hence at the end of the first phase of Algorithm 3.2, *i.e.,* Lines 4 - 7, all states in set $F_{\mathcal{P}}$ that are reachable from state q_f are included in the set $search\_from$. The second phase of Algorithm 3.2, *i.e.,* Lines 9 - 19, then checks whether or not state q_f is also reachable from a state in set $search\_from$, considering states in set $search\_from$ according to decreasing *dist* member value.

A cycle satisfying the conditions of Theorem 3.12 can be found by running Algorithm 3.2 from each state $q_f \in F_{\mathcal{P}}$ reachable from the initial state $q_{init,\mathcal{P}} \in Q_{\mathcal{P}}$, potentially traversing the entire automaton $|F_{\mathcal{P}}|$ times. However, leveraging ideas from the 'standard' nested DFS, *i.e.,* Algorithm 2.1, this worst-case computational complexity can be reduced.

Specifically, the idea is to run Algorithm 3.2 from states in set $F_{\mathcal{P}}$ in a particular order which ensures that states visited during previous executions of the algorithm do not need to be visited again; this is why the set of states *visited_ps* is used as an external input to Algorithm 3.4. More precisely, suppose that during a standard nested DFS run the inner search, *i.e.,* Algorithm 2.2, is called from a state $q'_f \in F_{\mathcal{P}}$ that is reachable from another state $q_f \in F_{\mathcal{P}}$. If the call does not return a cycle, then later on when the inner search is called from state q_f the states visited during the inner search called from state q'_f do not need to be considered again. This argument, combined with the following proposition, justifies the correctness of Algorithm 3.1.

Proposition 3.13. Let automaton $\mathcal{P}^*_{\mathcal{W}}$ be as in Proposition 3.8. Suppose state $q'_f \in F_{\mathcal{P}}$ is reachable from state $q \in F_{\mathcal{P}}$, and suppose state $q \in Q_{\mathcal{P}}$ is reachable from both q_f and q'_f. If there exists a cycle $\overline{c}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ starting at state q_f and containing state q, then there also exists a cycle $\overline{c}'$ of $\mathcal{P}^*_{\mathcal{W}}$ starting at state q'_f such that $rew(\overline{c}') \geq rew(\overline{c})$. □

Proof: From the structure of cycle $\overline{c}$, state q_f is reachable from state q. Hence there exists a path from state q to state q'_f, and since state q is reachable from state q'_f it follows that there exists a cycle $\overline{c}'$ starting at state q'_f and passing through state q and then through state q_f. From this construction, we realize that $rew(\overline{c}') \geq rew(\overline{c})$. ■

3.2.3 Algorithm Summary and Analysis

With the results from Subsection 3.2.1 and the algorithms from Subsection 3.2.2 at hand, we are now ready to state the overall solution to the problem of finding the most rewarding trace of transition system $\mathcal{T}$ along with its computational complexity.

Theorem 3.14. The following procedure returns the solution to Problem 3.2.

1. Each formula $\phi \in \mathbf{\Phi}$ is translated into a non-deterministic Büchi automaton $\mathcal{B}_\phi$, and the pair $(\boldsymbol{\mathcal{B}}, rew)$ is defined such that $\boldsymbol{\mathcal{B}} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ and $rew \colon \boldsymbol{\mathcal{B}} \mapsto \mathbb{N}$.
2. The weighted non-deterministic Büchi automaton $\mathcal{B}^*_{\mathcal{W}}$ associated with the pair $(\boldsymbol{\mathcal{B}}, rew)$ is constructed as per Definition 3.4.
3. The weighted non-deterministic Büchi automaton $\mathcal{P}^*_{\mathcal{W}} = \mathcal{T} \otimes \mathcal{B}^*_{\mathcal{W}}$ is constructed as per Definition 2.15.
4. Algorithm 3.1 is run with the automaton $\mathcal{P}^*_{\mathcal{W}}$ as its input.

□

Algorithm 3.1: `weighted_nested_DFS`

Input : Non-blocking weighted non-deterministic Büchi automaton $\mathcal{P}^*_{\mathcal{W}} = \mathcal{T} \otimes \mathcal{B}^*_{\mathcal{W}}$, where $\mathcal{P}^*_{\mathcal{W}} = (\mathcal{P}^*, \mathcal{W})$ and $\mathcal{P}^* = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$.
Output: Trace $\overline{s}$ of transition system $\mathcal{T}$ solving Problem 3.2.

```
1  initialize q.pred to none and q.dist to zero, for all q ∈ Q_P;
2  initialize visited_outer, visited_inner and visited_ps to the empty set;
3  initialize stack_outer, max_prefix and max_cycle to the empty stack;
4  initialize max_weight to zero;
5  visited_outer := visited_outer ∪ {q_init,P};
6  stack_outer.push(q_init,P);
7  repeat
8      q' = stack_outer.top();
9      if successors(q') \ visited_outer ≠ ∅ then
10         pick q'' ∈ successors(q') \ visited_outer;
11         visited_outer := visited_outer ∪ {q''};
12         stack_outer.push(q'');
13     else
14         stack_outer.pop();
15         if q' ∈ F_P then
16             re-initialize q'.pred to none and q'.dist to zero;
17             cycle := longest_cycle_search(q');
18             if q'.dist > max_weight then
19                 max_prefix := reverse(stack_outer); max_cycle := cycle;
20                 max_weight := q'.dist;
21 until ( stack_outer.is_empty() ∨ max_weight = n );
22 run := max_prefix · ( max_cycle^ω );
23 if ¬ run.is_empty() then
24     return projection of run onto set S;
25 else
26     return any trace of T;
```

Algorithm 3.2: `longest_cycle_search` (Executed within `weighted_nested_DFS`.)

Input : State $q_f \in F_{\mathcal{P}}$.
Output: Cycle $\bar{c}$ of automaton $\mathcal{P}^*_{\mathcal{W}}$ of maximum reward, if one exists.

```
1  initialize queue_curr to the queue consisting only of {q_f};
2  initialize queue_all[i] to the empty queue, for all 1 ≤ i ≤ n;
3  initialize search_from to the empty set;
4  propagate( queue_curr, queue_all, search_from);
5  foreach 1 ≤ i ≤ n do
6      queue_curr := queues_all[i];
7      if ¬ queue_current.is_empty() then
           propagate( queue_curr, queue_all, search_from);
8  sort elements in set search_from according to decreasing dist member value;
9  while ¬ search_from.is_empty() do
10     q := search_from.first_and_pop();
11     path_suffix := find_ps( q, q_f, visited_ps);
12     if ¬ path_suffix.is_empty() then
13         q_f.dist := q.dist;
14         initialize path_prefix to the empty path;
15         repeat
16             path_prefix := (q.pred) · (path_prefix);
17             q = q.pred;
18         until q = q_f;
19         return (path_prefix) · (path_suffix);
20 return the empty path;
```

Algorithm 3.3: `propagate` (Executed within `longest_cycle_search`.)

Input : Queue *queue_curr* of states of $Q_{\mathcal{P}}$; array *queue_all* of queues of states of $Q_{\mathcal{P}}$; set $search\_from$ of states of $Q_{\mathcal{P}}$.

```
1  repeat
2      q := queue_curr.front_and_pop();
3      if q ∉ visited_inner then
4          visited_inner := visited_inner ∪ {q};
5          foreach q' ∈ successors(q) do
6              if q'.dist < q.dist + W_P(q, q') then
7                  q'.dist := q.dist + W_P(q, q'); q'.pred := q;
8                  if layer(q') = 0 and q' ∉ search_from then
9                      search_from := search_from ∪ {q'};
10                 if layer(q') = layer(q) then
11                     queue_curr.push(q');
12                 else if there exists i ≥ 1 such that layer(q') = layer(q) + i then
13                     queue_all[i].push(q');
14 until queue_curr.is_empty();
```

Algorithm 3.4: `find_ps` (Executed within `longest_cycle_search`.)

Input : State $q \in F_{\mathcal{P}}$; state $q_f \in F_{\mathcal{P}}$; set *visited_ps* of states of $Q_{\mathcal{P}}$.
Output: Path of automaton $\mathcal{P}^*_{\mathcal{W}}$ in set $F_{\mathcal{P}}$ from state q to state q_f, if one exists.

```
1  initialize stack_ps to the empty stack;
2  visited_ps := visited_ps ∪ {q};
3  stack_ps.push(q);
4  repeat
5      q' := stack_ps.top();
6      if successors(q') \ visited_ps ≠ ∅ then
7          pick q'' ∈ successors(q') \ visited_ps;
8          visited_ps := visited_ps ∪ {q''};
9          stack_ps.push(q'');
10         if q'' = q_f then
11             return reverse(stack_ps);
12     else
13         stack_ps.pop();
14 until stack_ps.is_empty();
15 return the empty path;
```

Proof: Directly from Theorems 3.10 and 3.12 and Proposition 3.13. ■

Fact 3.15. Consider Problem 3.2. Let $|\mathcal{T}|$ denote the size of transition system $\mathcal{T}$, and let $|\mathbf{\Phi}| = \sum_{\phi \in \mathbf{\Phi}} |\phi|$. The worst-case computational complexity of Algorithm 3.1 is in $\mathcal{O}(\,|\mathcal{P}^*_{\mathcal{W}}|\,\log|\mathcal{P}^*_{\mathcal{W}}|\,)$, where the size of automaton $\mathcal{P}^*_{\mathcal{W}}$ is in $2^{\mathcal{O}(\,|\mathbf{\Phi}|\,)}\,|\mathcal{T}|$. □

To understand Fact 3.15, recall that the translation from an LTL formula ϕ into a generalized Büchi automaton can be done in $2^{\mathcal{O}(\,|\phi|\,)}$ time and space. In particular, one of the well-known translation algorithms transforms any formula ϕ into a generalized Büchi automaton with at most $2^{|\phi|}$ states and $|\phi|$ sets of states in its acceptance condition [GO01]. If the automata associated with specifications $\phi_1, \ldots, \phi_n \in \mathbf{\Phi}$ are all non-blocking, the worst-case size of automaton $\mathcal{B}^*_{\mathcal{W}}$ is $2^{(\,|\mathbf{\Phi}|\,)} \cdot (\,1 + |\mathbf{\Phi}|\,)$. On the other hand, if k of the obtained automata are blocking then the worst-case size of automaton $\mathcal{B}^*_{\mathcal{W}}$ is $2^{(\,|\mathbf{\Phi}|+k\,)} \cdot (\,|\mathbf{\Phi}| + k + 1\,)$. Although the size of automaton $\mathcal{B}^*_{\mathcal{W}}$ is exponential with respect to the size of the input specification, the sizes of the individual formulas are usually small and so in most cases of interest the sizes of the associated automata are significantly below the worst-case bound. Many optimization techniques have been developed in the formal verification literature to reduce the sizes of the generalized Büchi automata resulting from these translations.

The size of automaton $\mathcal{P}^*_{\mathcal{W}}$ is $|\mathcal{T}|\,|\mathcal{B}^*_{\mathcal{W}}|$ in the worst case, with at most $2^{(\,|\mathbf{\Phi}|+k\,)}\,(\,|\mathbf{\Phi}| + k\,)\,|\mathcal{T}|$ states in each layer, where k is the number of blocking generalized Büchi automata obtained from the translation of the formulae in $\mathbf{\Phi}$. The cumulative number of steps taken in sorting the set *search_from* in Line 8 of Algorithm 3.2 is in $\mathcal{O}(\,|F_{\mathcal{P}}|\,\log|F_{\mathcal{P}}|\,)$. Altogether, the complexity of Algorithm 3.1 is in $\mathcal{O}(\,|\mathcal{P}| + |F_{\mathcal{P}}|\,\log|F_{\mathcal{P}}|)$. Contrast this with the worst-case computational complexity of the 'brute-force' approach consisting of searching for a trace of transition system $\mathcal{T}$ satisfying every formulae in set $\widetilde{\mathbf{\Phi}}$ for each $\widetilde{\mathbf{\Phi}} \in 2^{\mathbf{\Phi}}$. A Büchi automaton $\tilde{\mathcal{B}}$ that is language equivalent to the conjunction of all formulae in set $\widetilde{\mathbf{\Phi}}$ can be constructed with up to $2^{|\widetilde{\mathbf{\Phi}}|} \cdot |\mathbf{\Phi}|$ states, in the worst case. A nested DFS, *i.e.,* Algorithm 2.1, is then run on automaton $\mathcal{P} = \mathcal{T} \otimes \tilde{B}$ at the expense of a worst-case computational complexity in $\mathcal{O}(\,|\mathcal{P}|\,)$. Hence, the worst-case computational complexity of the brute-force approach is in $\mathcal{O}\big(\,|\mathcal{T}|\,\sum_{\widetilde{\mathbf{\Phi}} \subseteq \mathbf{\Phi}} 2^{|\widetilde{\mathbf{\Phi}}|}\,|\widetilde{\mathbf{\Phi}}|\,\big)$.

The advantage of the approach we propose here, *i.e.,* Algorithm 3.1, with respect to the brute-force solution increases with the number of *non-blocking* automata obtained from the translation of the infinite LTL formulae. Note that for many LTL formulae the smallest equivalent generalized Büchi automaton is non-blocking. For instance, specifications such as $\mathsf{F}\,\phi$, *i.e.,* reachability, $\mathsf{G}\,(\,\mathsf{F}\,\phi\,)$, *i.e.,* surveillance, $\mathsf{G}\,(\,(\,\mathsf{F}\,\phi_1\,) \Rightarrow \mathsf{G}\,(\,\mathsf{F}\,\phi_2\,)\,)$, *i.e.,* reactivity, $\mathsf{G}(\,\phi_1 \Rightarrow (\,\mathsf{F}\,\phi_2\,)\,)$, *i.e.,* response, and $\mathsf{F}(\,\phi_1 \wedge (\,\mathsf{F}\,\phi_2\,)\,)$, *i.e.,* sequencing,

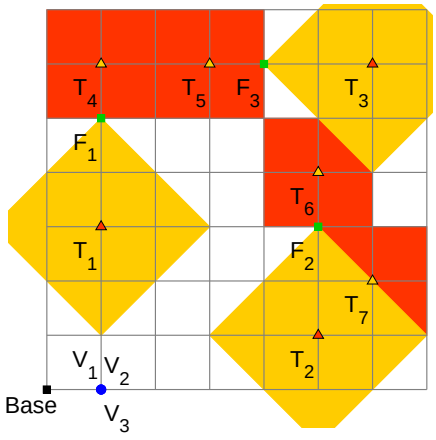


Figure 3-1: Transition system for the rescue mission example.

where ϕ, ϕ_1 and ϕ_2 are propositional logic formulae, belong to this class.

3.3 Rescue Mision Example

To illustrate the usefulness of the methods developed in this chapter we consider an example of a complex military rescue mission involving specifications which cannot be satisfied as a whole. Suppose that we have two unmanned aerial vehicles (UAVs), V_1 and V_2, and an autonomous ground vehicle (AGV), V_3, under our command. Furthermore, suppose that friendly units F_1, F_2 and F_3 have been captured and that they are being held deep within enemy territory. The friendly units are guarded by enemy targets $T_1, \ldots, T_7$, which need to be engaged and destroyed before vehicle V_3 can pick up the captured friendly units and bring them to base.

A particular mission configuration is depicted in Figure 3-1. Here, the captured friendly units F_1, F_2, and F_3 are shown as green squares, while the enemy targets T_1, ..., T_7 appear as triangles. The regions within the firing ranges of targets T_1, T_2 and T_3, and of targets T_4, T_5, T_6 and T_7, are depicted as yellow diamonds and red squares, respectively. The states of vehicles V_1, V_2 and V_3, which can move along the edges of the rectangular grid representing the environment, are shown as blue dots. Vehicles V_1 and V_2 can engage any target that is vulnerable to them target by visiting its location. On the other hand, if any vehicle enters the firing range of a target which the vehicle is vulnerable to then the

vehicle is destroyed. Notice that these rules are captured through irreversible transitions of the underlying transition system $\mathcal{T}$.

From Figure 3-1 we see that friendly unit F_1 can be rescued by engaging targets T_1 and T_4, that friendly unit F_2 can be rescued by engaging targets T_2, T_6 and T_7, and that friendly unit F_3 can be rescued by engaging targets T_3 and T_5. With this in mind, we consider two different scenarios. In our first scenario, the capabilities and weaknesses of the UAVs and of the AGV are as follows.

- Vehicle V_1 can engage and destroy targets T_1 and T_3, and is vulnerable to targets T_2 and T_5. It can also engage and destroy targets T_4, T_6 and T_7, but at the cost of self-destruction.
- Vehicle V_2 can engage and destroy targets T_2 and T_5, and is vulnerable to targets T_1 and T_3. It can also engage and destroy targets T_4, T_6 and T_7, but at the cost of self-destruction.
- Vehicle V_3 can pick-up and transport friendly units F_1, F_2 and F_3, but it is vulnerable to all targets.

In our second scenario the capabilities and weaknesses of the vehicles are as in the first one, except for the following additional capabilities.

- Vehicles V_1 and V_2 can engage and destroy target T_4 at no cost to their integrity, if they engage the target simultaneously.
- Vehicles V_1 and V_2 can engage and destroy target T_6 at no cost to their integrity, if they engage the target simultaneously.

Regarding the high-level mission specifications, the objective is to pick-up friendly units F_1, F_2, and F_3, and to do it while giving priority to friendly units F_1 and F_2 over unit F_3. This prioritization would be desirable, for instance, in the case where F_1 and F_2 are human units and F_3 is a robotic unit. In addition, we would like to be able to achieve these objectives without losing any of the vehicles we command, although sparing the unmanned vehicles from harm is not as important as picking up the friendly units.

To translate this path-and-mission planning problem into our framework, we consider the finite alphabet Π defined as follows.

$$\Pi = \bigcup_{\substack{i=1,2,3 \\ k=1,2,3}} \{\, a_{V_k},\, p_{V_k}^{Base},\, p_{V_k}^{F_i} \,\}$$

Mission Specification	**LTL Formula:** ϕ	$rew(\phi)$
Pick-up F_i and bring it to $Base$, for all $i \in \{1,2,3\}$.	$\mathsf{F}\,(\,p_{V_3}^{F_i} \wedge \mathsf{F}\,(\,p_{V_3}^{Base}\,)\,)$, for all $i \in \{1,2,3\}$.	10
Do not pick up F_3 before picking up F_i, for all $i \in \{1,2\}$.	$\mathsf{G}\,(\,p_{V_3}^{F_3} \Rightarrow \mathsf{G}\,(\,\neg\, p_{V_3}^{F_i}\,)\,)$, for all $i \in \{1,2\}$.	10
Do not lose vehicle V_k and bring V_k to $Base$, for all $k \in \{1,2,3\}$.	$\mathsf{G}\,(\,a_{V_k}\,) \wedge \mathsf{F}\,(\,p_{V_k}^{Base}\,)$, for $k \in \{1,2,3\}$.	1

Table 3.1: Specifications for our rescue mission example.

Here, symbol a_{V_k} holds true at any state where vehicle k is active, *i.e.,* it has not been destroyed because of entering the firing range of a target to which it is vulnerable. Symbol $p_{V_k}^{Base}$ holds true at any state where vehicle k is located at the base. Symbol $p_{V_k}^{F_i}$ holds true at any state where vehicle k is at the same location as friendly unit i. Thus we can precisely define the high-level mission specifications and their associated rewards for both scenarios as shown in Table 3.1. Notice that the reward function matches our objectives, as picking-up the friendly units in accordance to their relative priorities has a higher reward than avoiding the destruction of the vehicles we command.

Now that we have a transition system $\mathcal{T}$, and a pair ($\mathbf{\Phi}$, rew) consisting of the set of infinite LTL formulae $\mathbf{\Phi}$ and the function rew defined in Table 3.1, we can apply Theorem 3.14 to solve our path-and-mission planning problem. For this purpose we developed a C++ implementation of Algorithm 3.1 which takes as input a transition system and a set of Büchi automata $\boldsymbol{\mathcal{B}}$ obtained from set $\mathbf{\Phi}$ with the use of LTL2BA [GO01], an off-the-shelf automatic translation tool available online, along with a reward function.

The trace produced by our C++ implementation for the first scenario is shown in Figure 3-2 and has a reward of 32 units. The resulting trace indicates the following execution of the mission. First, vehicles V_1 and V_2 engage targets T_1 and T_5, respectively, as shown in Figures 3-2(a) and 3-2(b). Then, V_1 destroys target T_3, as seen in Figure 3-2(c), before launching a self-destructive attack on target T_4, as seen in Figure 3-2(d). Later on, vehicle V_2 engages target T_2, as shown in Figure 3-2(e). Finally, vehicle V_3 proceeds to pick-up friendly unit F_1, followed by friendly unit and F_3, as seen in Figure 3-2(f). The mission concludes when the two surviving vehicles return to base.

Because of the correctness of Algorithm 3.1, the trace in Figure 3-2 produced by our program is the *most rewarding mission execution* under the assumptions of the first scenario. This can be intuitively understood by noting that there are only two vehicles capable of

engaging targets (namely, V_1 and V_2), but there are three targets that can only be neutralized by a self-destructive engagement (namely, targets T_4, T_6 and T_7). Therefore no more than two friendly units can be rescued, and at least one vehicle must be 'sacrificed' if either one of friendly units F_1 or F_2 is to be picked-up.

On the other hand, the trace outputted by our implementation for the second scenario is shown in Figure 3-3 and has a reward of 52 units. The resulting trace indicates the following execution of the mission. First, vehicles V_1 and V_2 engage targets T_1 and T_5, respectively, as shown in Figures 3-3(a) and 3-3(b). Then, vehicle V_1 destroys target T_3, as seen in Figure 3-3(c). Next, vehicles V_1 and V_2 simultaneously engage and destroy target T_6 at no cost to their integrity, as shown in Figure 3-3(d). Vehicles V_1 and V_2 then again partner up, this time to engage and destroy target T_4, as seen in Figure 3-3(e). However, vehicle V_1 then proceeds to engage target T_7 at the cost of self-destruction, as shown in Figure 3-3(f). Finally, vehicle V_3 picks-up friendly units F_1, F_2 and F_3, in this order, and the mission concludes when the two remaining vehicles return to base. This result is intuitive, as there exists one target which can only be destroyed at the expense of a vehicle (namely, target T_7). Hence, the best mission execution cannot have a reward in excess of 52 units, which is exactly the reward of the trace produced by our implementation under the assumptions of the second scenario.

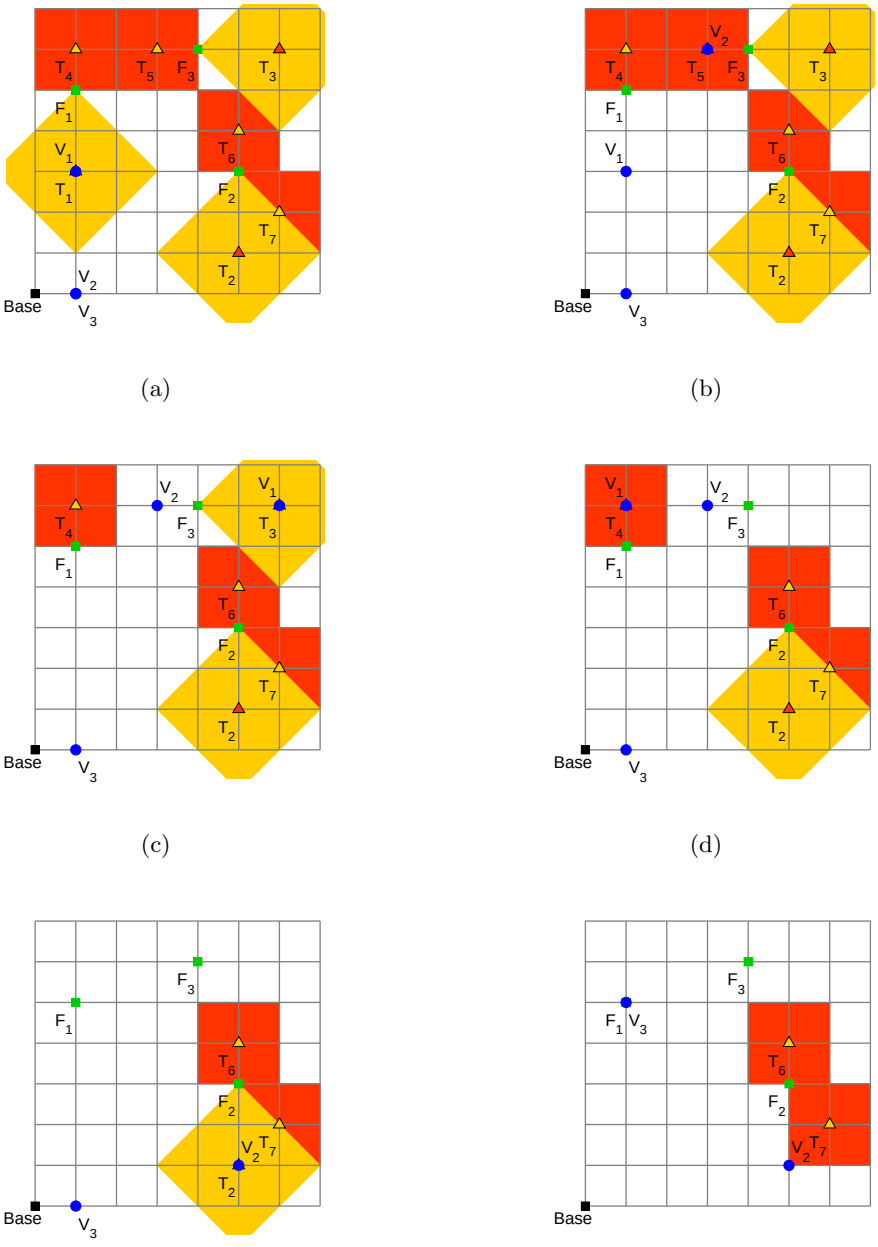


Figure 3-2: Most rewarding rescue mission execution (first scenario).

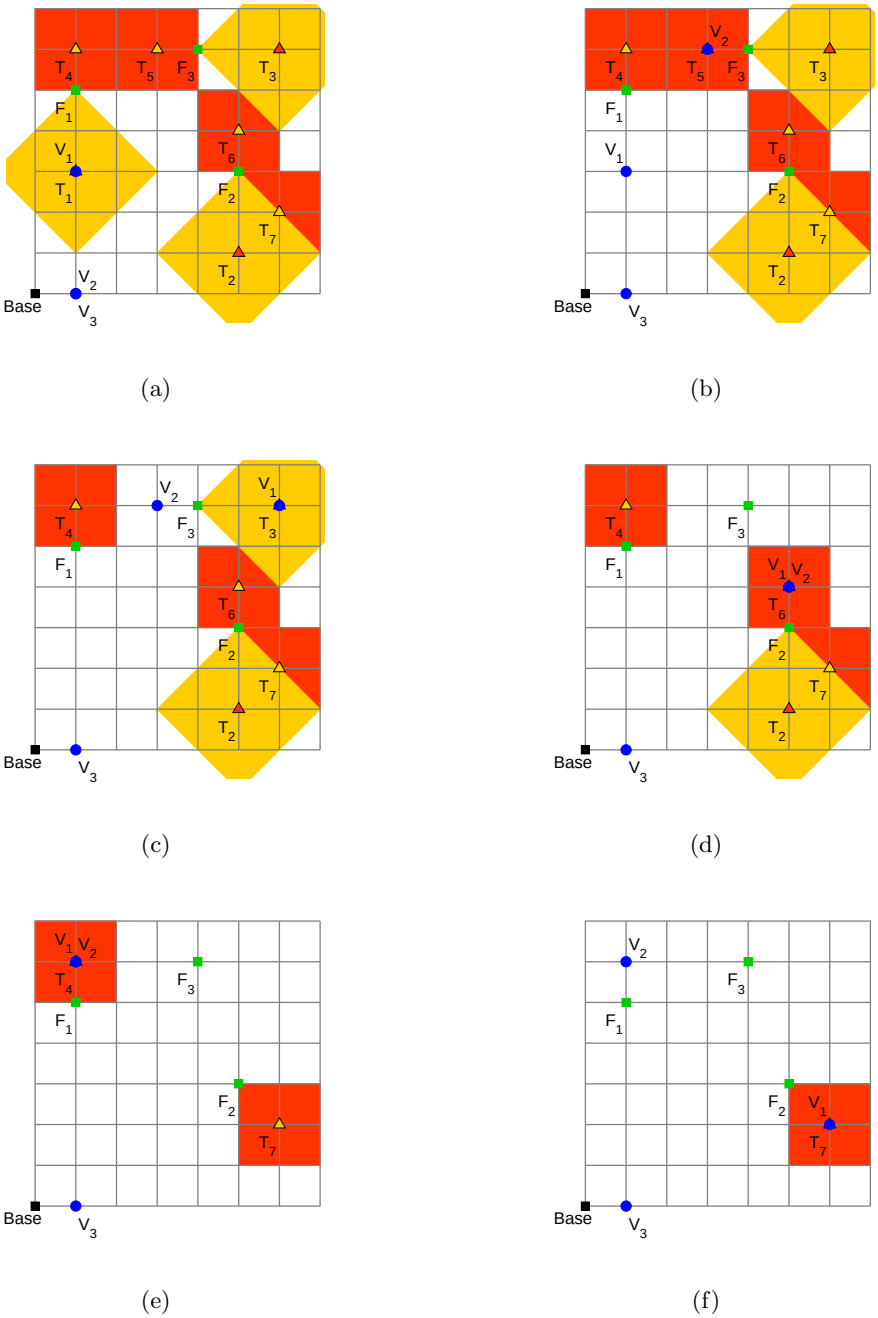


Figure 3-3: Most rewarding rescue mission execution (second scenario).

Chapter 4

Minimum-violation SFLTL$_{-\chi}$ Motion Planning

This chapter constitutes the second major contribution of this thesis. It is structured as follows. Section 4.1 provides several definitions necessary for the understanding of the problem of interest. Section 4.2 then formally states the problem. Next, Section 4.3 describes our solution approach in detail, which combines a special type of finite automaton of our design with an incremental sampling-based formal verification algorithm. Finally, Section 4.4 provides case studies of the use of our solution approach to urban driving problems, both in simulation and in a real-time testbed.

A preliminary version of this work has been published in [RCCT+13].

4.1 Introductory Definitions

This section introduces several definitions, along with a few facts, necessary for the understanding of the motion planning problem that we propose and solve in this chapter.

4.1.1 Dynamical Systems

In contrast with Chapter 3, where we considered models of robots operating sequentially in discrete state spaces, this chapter considers models of robots that operate in continuous time and in continuous state spaces. These models are useful in synthesizing motion plans for vehicles such as cars and airplanes.

Definition 4.1. A *continuous-time continuous-space dynamical system* is a tuple $\mathcal{S} = (X, U, f)$, where X is a d-dimensional manifold, $U \subseteq \mathbb{R}^m$ is a compact set, and $f\colon X \times U \mapsto \mathbb{R}^d$ is a function which is Lipschitz continuous in each of its arguments. Sets X and U are referred to as the *state space* and *control input space* of the system, respectively. A *trajectory* of $\mathcal{S}$ is a continuous function $x\colon [0,T] \mapsto X$ with the property that there exists a control input history $u\colon [0,T] \mapsto U$ such that

$$x(t) = x(0) + \int_0^t f(x(\tau), u(\tau))\, d\tau\,, \quad \text{for all } t \in [0,T]\,.$$

□

Definition 4.2. A partition of the closed interval $[0,1]$ is a finite ordered sequence $\{\tau_0, \ldots, \tau_{k-1}\}$ such that $\tau_0 = 0$, such that $\tau_i < \tau_{i+1}$ for all $0 \leq i \leq k-2$, and such that $\tau_{k-1} = 1$. We let $\mathfrak{P}$ denote the set of all partitions of $[0,1]$. Let dynamical system $\mathcal{S}$ be as in Definition 4.1, and let $x\colon [0,T] \mapsto X$ and $x'\colon [0,T'] \mapsto X$ be two of its trajectories. Further, define functions $\overline{x}, \overline{x}', z\colon [0,1] \mapsto X$ such that $\overline{x}(\tau) = x(T\tau)$, such that $\overline{x}'(\tau) = x'(T'\tau)$, and such that $z(\tau) = \overline{x}(\tau) - \overline{x}'(\tau)$, for all $\tau \in [0,1]$. Notice that $\overline{x}$, $\overline{x}'$ and z are all continuous functions, but they are not necessarily trajectories of $\mathcal{S}$.

With these definitions at hand, the *Total Variation Distance* between trajectories x and x' of dynamical system $\mathcal{S}$, which is denoted as $d_{TV}(x, x')$, is defined as follows.

$$d_{TV}(x,x') = \sup_{\{\tau_0,\ldots,\tau_{k-1}\}\in\mathfrak{P}} \sum_{i=0}^{k-2} \| z(\tau_{i+1}) - z(\tau_i) \|_2$$

□

Given trajectories $x_1\colon [0,T_1] \mapsto X$ and $x_2\colon [0,T_2] \mapsto X$ of the dynamical system $\mathcal{S}$ in Definition 4.1 such that $x_1(T_1) = x_2(0)$, the concatenation of x_1 and x_2, denoted $x_1 \cdot x_2$, is

the trajectory $x_{12}\colon [\,0,\, T_1 + T_2] \mapsto X$ of $\mathcal{S}$ defined as follows.

$$x_{12}(t) = \begin{cases} x_1(t), & \text{if } \ 0 \leq t \leq T_1 \\ x_2(t - T_1), & \text{if } \ T_1 < t \leq T_2 \end{cases}$$

Given a continuous functional c mapping trajectories of dynamical system $\mathcal{S}$ to non-negative costs, we say that c is *monotonically non-decreasing with respect to concatentation* if for any pair of trajectories $x_1\colon [0, T_1] \mapsto X$ and $x_2\colon [0, T_2] \mapsto X$ of $\mathcal{S}$ such that $x_1(T_1) = x_2(0)$, we have that $c(x_1) \leq c(x_1 \cdot x_2)$.

Consider a finite alphabet Π and a labeling function $\mathcal{L}_{cts}\colon X \mapsto 2^{\Pi}$ mapping each state of dynamical system $\mathcal{S}$ to the set of symbols that hold true at that state. Given a trajectory $x\colon [0, T] \mapsto X$ of $\mathcal{S}$, we define the set of label changes of x, denoted $D(x)$, as the union of zero with the set of discontinuities of the function $\mathcal{L}_{cts}(x)$, where we tacitly assume the existence of a metric function $d_{\Pi}\colon 2^{\Pi} \times 2^{\Pi} \mapsto \{0, 1\}$ such that for every $\pi, \pi' \in 2^{\Pi}$ we have that $d(\pi, \pi')$ is zero if $\pi = \pi'$ and is one otherwise. In other words,

$$D(x) = \{0\} \cup \left\{ \, t \in [0, T] \, : \, \mathcal{L}_{cts}(x(t)) \neq \lim_{s \to t^-} \mathcal{L}_{cts}(x(s)) \, \right\}.$$

Furthermore, letting $\Sigma = 2^{\Pi} \times 2^{\Pi}$, we define the *word induced* by trajectory $x\colon [0, T] \mapsto X$ of dynamical system $\mathcal{S}$ as the finite word $w(x) \in \Sigma^*$ such that if $D(x) = \{\, t_0, \ldots, t_{n-1}\}$ and $t_i < t_{i+1}$ for all $0 \leq i \leq n - 2$ then

$$\begin{aligned} w(x) = &(\, \mathcal{L}_{cts}(x(t_0)), \mathcal{L}_{cts}(x(t_0)) \,), \, (\, \mathcal{L}_{cts}(x(t_0)), \mathcal{L}_{cts}(x(t_1)) \,), \ldots, \\ &(\, \mathcal{L}_{cts}(x(t_{n-2})), \mathcal{L}_{cts}(x(t_{n-1})) \,), \, (\, \mathcal{L}_{cts}(x(t_{n-1})), \mathcal{L}_{cts}(x(t_{n-1})) \,). \end{aligned}$$

Moreover, letting $\tau_i = t_{i+1} - t_i$ for all $0 \leq i \leq n - 2$ and $\tau_{n-1} = T - t_{n-1}$, the *timed word induced* by trajectory x is the finite timed word $w_\tau(x) \in \Sigma^*_{\geq 0}$ defined such that

$$\begin{aligned} w_\tau(x) = &(\, (\, \mathcal{L}_{cts}(x(t_0)), \mathcal{L}_{cts}(x(t_0)) \,), \, \tau_0 \,), \, (\, (\, \mathcal{L}_{cts}(x(t_0)), \mathcal{L}_{cts}(x(t_1)) \,), \, 0 \,), \ldots, \\ &(\, (\, \mathcal{L}_{cts}(x(t_{n-2})), \mathcal{L}_{cts}(x(t_{n-1})) \,), \, 0 \,), \, (\, (\, \mathcal{L}_{cts}(x(t_{n-1})), \mathcal{L}_{cts}(x(t_{n-1})) \,), \, \tau_{n-1} \,). \end{aligned}$$

The following definition and fact relate the trajectories of the dynamical system in Definition 4.1 to the durational transition system in Definition 2.3.

Definition 4.3. Consider the continuous-time continuous-space dynamical system $\mathcal{S} = (\, X, \, U, \, f \,)$. A durational transition system $\mathcal{D} = (\, \mathcal{T}, \, \Delta)$, where $\mathcal{T} = (\, S, \, s_{init}, \, \mathcal{R}, \, \Pi, \mathcal{L})$, is

said to be *trace inclusive* with respect to $\mathcal{S}$ if the following two conditions hold.

- $S \subseteq X$.
- If $(s, s') \in \mathcal{R}$ then there exists a trajectory $x\colon [0, T] \mapsto X$ of $\mathcal{S}$ such that $x(0) = s$, such that $x(T) = s'$, such that $T = \Delta(s, s')$, and such that $|D(x)| \leq 2$, *i.e.,* such that function $\mathcal{L}_{cts}(x)$ changes at most once in the interval $(0, T]$.

□

Fact 4.4. Let dynamical system $\mathcal{S}$ and durational transition system $\mathcal{D}$ be as in Definition 4.3. If $\mathcal{D}$ is trace inclusive with respect to $\mathcal{S}$, then for any finite trace $\overline{s}$ of $\mathcal{D}$ of length at least two that induces a finite timed word $w_\tau(\overline{s})$ there exists a trajectory $x\colon [0, T] \mapsto X$ of $\mathcal{S}$ such that $\texttt{destutter}(w_\tau(\overline{s})) = w_\tau(x)$. □

4.1.2 SFLTL$_{-\mathsf{X}}$: Safety Finite Linear Temporal Logic Minus Next

Finite automata can capture a large class of properties that are exhibited by the finite traces of a transition system. However, as discussed at the beginning of Section 2.5, some formal languages with similar expressive power, such as regular expressions and variants of the Linear Temporal Logic (LTL), provide a more user-friendly means to express these properties. See [Sip06] and [BK08] for more arguments on why this is the case. Hence in this chapter we utilize a modification of the finite LTL which features pairs of symbols as its primitives, but which does not allow the X operator (pronounced 'next').

Definition 4.5. A *Safety Finite Linear Temporal Logic Minus Next (SFLTL$_{-\mathsf{X}}$) Formula* over finite alphabet Π is any finite LTL formula ϕ over finite alphabet $\Pi \times \Pi$ such that the following two conditions hold.

- Formula ϕ does not contain the X operator.
- Let $\Sigma = 2^\Pi \times 2^\Pi$. If finite word $\overline{\sigma} \in \Sigma^*$ violates formula ϕ then there does not exist a finite word $\overline{\sigma}' \in \Sigma^*$ such that $\overline{\sigma} \cdot \overline{\sigma}' \models \phi$. That is, a word that violates formula ϕ cannot be extended into a word that satisfies it.

□

Example 4.6. Recall the transition system $\mathcal{T}$ from Example 2.2, where

$\Pi = \{ \mathit{blue}, \mathit{green}, \mathit{red} \}$. Then, finite LTL formulae

$$(\mathit{blue} , \mathit{blue}) \, \mathsf{U} \, (\mathit{blue} , \mathit{green}) \quad \text{and} \quad \mathsf{G} (\neg (\mathit{red} , \mathit{red}))$$

are both SFLTL$_{-\mathsf{X}}$ formulae over Π. On the other hand, finite LTL formulae

$$\mathsf{X} (\mathit{blue} , \mathit{blue}) \quad \text{and} \quad \mathsf{F} (\mathit{blue} , \mathit{green})$$

are not SFLTL$_{-\mathsf{X}}$ formulae over Π. □

The reason why SFLTL$_{-\mathsf{X}}$ is defined in terms of pairs of symbols is that this construction allows us to express properties of the transitions of dynamical systems, such as the crossing of boundaries, for instance. In addition, the reason why SFLTL$_{-\mathsf{X}}$ does not include the X operator of finite LTL is that in continuous time the notion of a 'next step' is not well-defined. Finally, notice that an SFLTL$_{-\mathsf{X}}$ formula can be automatically translated into a finite automaton using the same techniques available for the translation of finite LTL formulae to finite automata, such as the algorithm given in [GPVW95].

4.1.3 Level of Violation

Consider an ordered sequence of non-empty sets of SFLTL$_{-\mathsf{X}}$ formulae $\boldsymbol{\Phi} = \{ \Phi_1, \ldots, \Phi_n \}$ over a finite alphabet Π, where $\Phi_i = \{ \phi_1^{(i)}, \ldots, \phi_{m_i}^{(i)} \}$ for each $1 \leq i \leq n$, along with an ordered sequence of functions $\boldsymbol{\varpi} = \{ \varpi_1, \ldots, \varpi_n \}$ such that $\varpi_i \colon \Phi_i \mapsto \mathbb{R}_{\geq 0}$ for each $1 \leq i \leq n$. The pair $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$, which we refer to as a *set of SFLTL$_{-\mathsf{X}}$ formulae with priorities* over alphabet Π, constitutes the second major parameter of the problem to be proposed and solved in this chapter. Intuitively, the weights ϖ_i allow us to trade-off violations of the different formulae belonging to set Φ_i, for each $1 \leq i \leq n$. With this in mind, the following definitions precisely describe the manner in which we penalize a word or timed word for violating a particular set of SFLTL$_{-\mathsf{X}}$ formulae with priorities.

Definition 4.7. Let Σ be a finite alphabet. For any finite word $\overline{\sigma} = \sigma_0, \ldots, \sigma_{k-1} \in \Sigma^*$ and any index set $\mathcal{I} \subseteq \{0, \ldots, k-1\}$, define $\mathrm{vanish}(\overline{\sigma}, \mathcal{I})$ as the sub-word of $\overline{\sigma}$ obtained by removing all letters σ_i with $i \in \mathcal{I}$. Similarly, for any finite timed word $\overline{\sigma}_\tau = (\sigma_0, \tau_0), \ldots, (\sigma_{k-1}, \tau_{k-1}) \in \Sigma_{\geq 0}^*$ and any index set $\mathcal{I} \subseteq \{0, \ldots, k-1\}$, define $\mathrm{vanish}(\overline{\sigma}_\tau, \mathcal{I})$ as the sub-word of $\overline{\sigma}_\tau$ obtained by removing all pairs (σ_i, τ_i) with $i \in \mathcal{I}$. □

Definition 4.8. Let Π be a finite alphabet, and let $\Sigma = 2^\Pi \times 2^\Pi$. Given a set of

SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ over Π, the *Level of Violation* of finite timed word $\overline{\sigma}_\tau = (\sigma_0, \tau_0), \ldots, (\sigma_{k-1}, \tau_{k-1}) \in \Sigma^*_{\geq 0}$ with respect to the formula $\phi_j^{(i)} \in \Phi_i$, denoted $\lambda(\overline{\sigma}_\tau, \phi_j^{(i)})$, is defined as follows.

$$\lambda(\overline{\sigma}_\tau, \phi_j^{(i)}) = \varpi_i(\phi_j^{(i)}) \left[\min_{\{\, \mathcal{I} \subseteq \{0, \ldots, k-1\} \,:\, \text{vanish}(\overline{\sigma}_\tau, \mathcal{I}) \models \phi_j^{(i)} \,\}} \sum_{i \in \mathcal{I}} \tau_i \right]$$

That is, $\lambda(\overline{\sigma}_\tau, \phi_j^{(i)})$ is the cumulative duration of the letters of timed word $\overline{\sigma}_\tau$, weighted by $\varpi_i(\phi_j^{(i)})$, that need to be removed in order for $\overline{\sigma}_\tau$ to satisfy formula $\phi_j^{(i)}$. □

Remark 4.9. Following the setup of Definition 4.8, we observe that for any trajectory $x : [0, T] \mapsto X$ of the dynamical system $\mathcal{S}$ in Definition 4.1 we have that $\lambda(w_\tau(x), \phi_j^{(i)}) \geq 0$. In addition, from the second condition in the definition of SFLTL$_{-\mathsf{X}}$ formulae, we realize that for any trajectories $x_1 : [0, T_1] \mapsto X$ and $x_2 : [0, T_2] \mapsto X$ of dynamical system $\mathcal{S}$ such that $x_1(T_1) = x_2(0)$ we have that

$$\lambda(w_\tau(x_1), \phi_j^{(i)}) \leq \lambda(w_\tau(x_1 \cdot x_2), \phi_j^{(i)}).$$

Therefore function $\lambda(w_\tau(\cdot), \phi_j^{(i)})$, which maps trajectories of dynamical system $\mathcal{S}$ to their levels of violation with respect to formula $\phi_j^{(i)}$, is a non-negative cost functional that is non-decreasing with respect to concatenation, for each $1 \leq i \leq n$ and each $1 \leq j \leq m_i$. □

Definition 4.10. Given a set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ over finite alphabet Π, let $\Sigma = 2^\Pi \times 2^\Pi$. Then for any finite timed word $\overline{\sigma}_\tau \in \Sigma^*_{\geq 0}$, the Level of Violation of $\overline{\sigma}_\tau$ with respect to the set of formulae Φ_i, denoted $\lambda(\overline{\sigma}_\tau, \Phi_i)$, is defined as follows.

$$\lambda(\overline{\sigma}_\tau, \Phi_i) = \sum_{j=1}^{m_i} \lambda(\overline{\sigma}_\tau, \phi_j^{(i)})$$

□

Definition 4.11. Given a set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ over finite alphabet Π, let $\Sigma = 2^\Pi \times 2^\Pi$. Then for any finite timed word $\overline{\sigma}_\tau \in \Sigma^*_{\geq 0}$, the Level of Violation of $\overline{\sigma}_\tau$ with respect to the pair $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$, denoted $\lambda(\overline{\sigma}_\tau, \boldsymbol{\Phi})$, is defined as the following ordered n-tuple.

$$\lambda(\overline{\sigma}_\tau, \boldsymbol{\Phi}) = (\lambda(\overline{\sigma}_\tau, \Phi_1), \ldots, \lambda(\overline{\sigma}_\tau, \Phi_n))$$

□

We emphasize that the level of violation of a timed word with respect to the set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ is an ordered n-tuple. In fact, we compare the levels of violation of different words by means of the Standard Lexicographical Ordering. That is, if $\lambda = (\lambda_1, \ldots, \lambda_n)$ and $\lambda' = (\lambda'_1, \ldots, \lambda'_n)$ are the levels of violation with respect to the pair $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ of two different words, then we say that $\lambda \leq \lambda'$ if either one of the following two conditions holds.

- $\lambda_i = \lambda'_i$ for all $1 \leq i \leq n$.
- $\lambda_i < \lambda'_i$ for some $1 \leq i \leq n$ and $\lambda_j = \lambda'_j$ for all $1 \leq j < i$.

Therefore the set of all levels of violation with respect to the pair $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ is *totally ordered*, and so under appropriate conditions it makes sense to say that a set of words contains a word of minimum level of violation with respect to the pair $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$.

4.2 Problem Formulation

Armed with the definitions given in the previous section, we can now precisely state the problem we are interested in solving in this chapter. Intuitively, the problem asks that we find a minimum cost trajectory of the dynamical system in Definition 4.1 among all trajectories of minimum level of violation that reach a goal region.

Problem 4.12. Given

- a continuous-time continuous-space dynamical system $\mathcal{S} = (X, U, f)$, along with a finite alphabet Π and a labeling function $\mathcal{L}_{cts} \colon X \mapsto 2^{\Pi}$,
- a compact collision-free space $X_{free} \subseteq X$ of non-zero Lebesgue measure, an initial state $x_{init} \in X_{free}$, and a goal set $X_{goal} \subseteq X_{free}$ of non-zero Lebesgue measure,
- a continuous functional c mapping trajectories of $\mathcal{S}$ into non-negative costs that is monotonically non-decreasing with respect to concatenation, and
- a set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ over Π,

find a trajectory $x^* \colon [0, T^*] \mapsto X_{free}$ with the following properties.

(i) It reaches X_{goal}, *i.e.,* $x^*(T^*) \in X_{goal}$.

(ii) Trajectory x^* achieves the minimum level of violation with respect to $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ among all trajectories of dynamical system $\mathcal{S}$ that reach X_{goal}. That is, if $x \colon [0, T] \mapsto X_{free}$ is a trajectory of $\mathcal{S}$ with $x(T) \in X_{goal}$, then $\lambda(w_\tau(x^*), \boldsymbol{\Phi}) \leq \lambda(w_\tau(x), \boldsymbol{\Phi})$.

(ii) Trajectory x^* achieves the minimum cost among all trajectories of dynamical system $\mathcal{S}$ that reach X_{goal} and that minimize their levels of violation with respect to $(\,\mathbf{\Phi},\,\boldsymbol{\varpi})$ among all trajectories of $\mathcal{S}$ that reach X_{goal}. That is, if $x\colon [0,T] \mapsto X_{free}$ is a trajectory of $\mathcal{S}$ with $x(T) \in X_{goal}$ and such that $\lambda(w_\tau(x), \mathbf{\Phi}) \leq \lambda(w_\tau(x'), \mathbf{\Phi})$ for any trajectory $x'\colon [0,T'] \mapsto X_{free}$ of $\mathcal{S}$ such that $x'(T') \in X_{goal}$, then $c(x^*) \leq c(x)$.

□

For briefness, in this chapter we restrict our attention to *minimum-time* cost functionals, meaning that for any trajectory $x\colon [0,T] \mapsto X$ of $\mathcal{S}$ we have that $c(x) = T$. Nevertheless, the algorithm we propose to solve Problem 4.12 is capable of handling a rich class of cost functionals, including discounted cost functionals and functionals of both the states and the controls, with minor modifications.

In any case, in order to develop an algorithmic solution to the aforementioned problem, we incrementally abstract it into the following discrete path search problem defined on a durational transition system that is trace-inclusive with respect to the dynamical system of interest. To avoid confusion, please be aware that the term 'incremental' is explained in context in the next section.

Problem 4.13. Given

- a durational transition system $\mathcal{D} = (\,\mathcal{T},\,\Delta)$, where $\mathcal{T} = (\,S,\,s_{init},\,\mathcal{R},\,\Pi,\mathcal{L}\,)$,
- a goal set $S_{goal} \subseteq S$, and
- a set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\,\mathbf{\Phi},\,\boldsymbol{\varpi})$ over Π,

find a finite trace $\overline{s}^* = s_0^*, \ldots, s_{k-1}^*$ of $\mathcal{D}$ with the following properties.

(i) It reaches S_{goal}, *i.e.*, $s_{k-1}^* \in S_{goal}$.

(ii) Trace $\overline{s}^*$ achieves the minimum level of violation with respect to $(\,\mathbf{\Phi},\,\boldsymbol{\varpi})$ among all finite traces of $\mathcal{D}$ that reach S_{goal}. That is, if $\overline{s} = s_0, \ldots, s_{\ell-1}$ is a finite trace of $\mathcal{D}$ with $s_{\ell-1} \in S_{goal}$, then $\lambda(w_\tau(\overline{s}^*), \mathbf{\Phi}) \leq \lambda(w_\tau(\overline{s}), \mathbf{\Phi})$.

(iii) Trace $\overline{s}^*$ achieves the minimum cost among all finite traces of $\mathcal{D}$ that reach S_{goal} and that minimize their level of violation with respect to $(\,\mathbf{\Phi},\,\boldsymbol{\varpi})$ among all finite traces of $\mathcal{D}$ that reach S_{goal}. That is, if $\overline{s} = s_0, \ldots, s_{\ell-1}$ is a finite trace of $\mathcal{D}$ with $s_{\ell-1} \in S_{goal}$ and such that $\lambda(w_\tau(\overline{s}), \mathbf{\Phi}) \leq \lambda(w_\tau(\overline{s}'), \mathbf{\Phi})$ for any finite trace $\overline{s}' = s'_0, \ldots, s'_{m-1}$ of $\mathcal{D}$ such that $s'_{m-1} \in S_{goal}$, then $\Delta(\overline{s}^*) \leq \Delta(\overline{s})$.

□

4.3 Problem Solution

This section describes an algorithm for solving Problem 4.12, in an asymptotic sense. In particular, we propose an incremental sampling-based algorithm, based on the RRT* algorithm introduced in [KF11], that constructs, one state at a time, the product of a durational transition system $\mathcal{D}$ and of a weighted finite automaton $\widetilde{\boldsymbol{\mathcal{A}}}$, such that finite automaton $\mathcal{D} \otimes \widetilde{\boldsymbol{\mathcal{A}}}$ captures the priorities of the set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ through the weights of its transitions. Roughly speaking, the algorithm works because the shortest path in automaton $\mathcal{D} \otimes \widetilde{\boldsymbol{\mathcal{A}}}$ with respect to level of violation and duration maps onto a trace of system $\mathcal{D}$ of minimum level of violation and duration. In addition, from the formulation of Problem 4.12, it follows that if there exists a minimum-time trajectory $x^*\colon [0, T] \mapsto X_{free}$ from x_{init} to X_{goal} that satisfies all the SFLTL$_{-\mathsf{X}}$ formulae, then our algorithm returns a sequence of trajectories that approach x^* in the sense of the total variation.

In this section we work with the pair $(\mathbf{A}, \boldsymbol{\varpi})$ associated with the set of SFLTL$_{-\mathsf{X}}$ formulae with priorities $(\boldsymbol{\Phi}, \boldsymbol{\varpi})$ over finite alphabet Π, where $\mathbf{A} = \{\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_n\}$ is an ordered sequence of sets of non-deterministic finite automata with input alphabet $\Sigma = 2^{\Pi} \times 2^{\Pi}$ such that $\boldsymbol{\mathcal{A}}_i = \{\mathcal{A}_1^{(i)}, \ldots, \mathcal{A}_{m_i}^{(i)}\}$ for each $1 \leq i \leq n$ and such that $L(\phi_j^{(i)}) = L(\mathcal{A}_j^{(i)})$ and $\varpi_i(\phi_j^{(i)}) = \varpi_i(\mathcal{A}_j^{(i)})$ for each $1 \leq i \leq n$ and each $1 \leq j \leq m_i$.

4.3.1 Design of the Weighted Automata

In this subsection we describe a four-step procedure for solving Problem 4.13, which in the next subsection we extend to an algorithm for asymptotically solving Problem 4.12. Since this chapter builds on previous work on the part of our collaborator Jana Tumova, Definitions 4.14 and 4.16 and Propositions 4.15 and 4.17 are not original of this chapter, and they were first published in [THK+13]. Nonetheless, these definitions and results are included in this chapter for the purpose of completeness.

The first step of our procedure is to augment finite automaton $\mathcal{A}_j^{(i)}$, for each $1 \leq i \leq n$ and each $1 \leq j \leq m_i$, with new transitions and weights such that the resulting weighted finite automaton *accepts all* words $\overline{\sigma} \in \Sigma^*$, including those that do not satisfy formula $\phi_j^{(i)}$. In particular, the weights of the transitions of automaton $\mathcal{A}_j^{(i)}$ are chosen such that the weight of its shortest run over finite word $\overline{\sigma} \in \Sigma^*$ is equal to the level of violation of $\overline{\sigma}$ with respect to formula $\phi_j^{(i)}$.

Definition 4.14. Given a set of non-deterministic finite automata with priorities ($\mathbf{A}$, $\boldsymbol{\varpi}$), consider the non-deterministic finite automaton $\mathcal{A}_j^{(i)} = (\, Q_j^{(i)},\, q_{init,j}^{(i)},\, \Sigma,\, \delta_j^{(i)}, F_j^{(i)})$, for any $1 \leq i \leq n$ and any $1 \leq j \leq m_i$. The *weighted version* of automaton $\mathcal{A}_j^{(i)}$ is the weighted non-deterministic finite automaton $\widetilde{\mathcal{A}}_j^{(i)} = (\, Q_j^{(i)},\, q_{init,j}^{(i)},\, \Sigma,\, \widetilde{\delta}_j^{(i)}, F_j^{(i)},\, \widetilde{\mathcal{W}}_j^{(i)})$, where transition relation $\widetilde{\delta}_j^{(i)}$ and transition weight function $\widetilde{\mathcal{W}}_j^{(i)}$ are defined as follows.

$$\widetilde{\delta}_j^{(i)} = \delta_j^{(i)} \cup \left\{ \; (q, \sigma, q) \notin \delta_j^{(i)} \; : \; q \in Q_j^{(i)} \text{ and } \sigma \in \Sigma \; \right\}$$

$$\widetilde{\mathcal{W}}_j^{(i)}(t) = \begin{cases} 0, & \text{if } \; t \in \delta_j^{(i)} \\ \varpi_i(\mathcal{A}_j^{(i)}), & \text{if } \; t \in \widetilde{\delta}_j^{(i)} \setminus \delta_j^{(i)} \end{cases}$$

□

Proposition 4.15. Let the pair ($\mathbf{A}$, $\boldsymbol{\varpi}$) and the finite automata $\mathcal{A}_j^{(i)}$ and $\widetilde{\mathcal{A}}_j^{(i)}$ be as in Definition 4.14. Then, automaton $\widetilde{\mathcal{A}}_j^{(i)}$ accepts any finite word over Σ. Furthermore, if $\overline{q}$ is the shortest accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over finite word $\overline{\sigma} \in \Sigma^*$, then $\widetilde{\mathcal{W}}_j^{(i)}(\overline{q}) = \lambda(\,\overline{\sigma},\, \phi_j^{(i)}\,)$. □

Proof: By induction on the level of violation with respect to formula $\phi_j^{(i)}$. For the base case, suppose that finite word $\overline{\sigma} \in \Sigma^*$ is such that $\lambda(\,\overline{\sigma},\, \phi_j^{(i)}\,) = 0$. Then $\overline{\sigma}$ satisfies $\phi_j^{(i)}$, and so there exists an accepting run of automaton $\mathcal{A}_j^{(i)}$ over $\overline{\sigma}$. Consequently, the shortest accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over word $\overline{\sigma}$ has weight equal to zero.

For the inductive hypothesis, suppose that for some $m \in \mathbb{N}_0$ it is true that for any finite word $\overline{\sigma} \in \Sigma^*$ such that $\lambda(\,\overline{\sigma},\, \phi_j^{(i)}\,) = m\, \varpi_i(\,\phi_j^{(i)}\,)$ the shortest accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over word $\overline{\sigma}$ has weight equal to $m\, \varpi_i(\,\phi_j^{(i)}\,)$. For the inductive step, consider any word $\overline{\sigma}' = \sigma'_0, \ldots, \sigma'_{k-1} \in \Sigma^*$ such that $\lambda(\,\overline{\sigma}',\, \phi_j^{(i)}\,) = (m+1)\, \varpi_i(\,\phi_j^{(i)}\,)$. From the definition of level of violation with respect to an SFLTL$_{-\mathsf{X}}$ formula we see that there exists $0 \leq \ell \leq k-1$ such that if $\overline{u} = \sigma'_0, \ldots, \sigma'_{\ell-1}$ and $\overline{v} = \sigma'_{\ell+1}, \ldots, \sigma'_{k-1}$ then $\lambda(\,\overline{u} \cdot \overline{v},\, \phi_j^{(i)}\,) = m\, \varpi_i(\,\phi_j^{(i)}\,)$.

Now, let finite run $\overline{q}^{uv} = q_0^{uv}, \ldots, q_{k-1}^{uv}$ be a shortest accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over word $\overline{u} \cdot \overline{v}$. From the inductive hypothesis, we have that run $\overline{q}^{uv}$ has weight equal to $m\, \varpi_i(\,\phi_j^{(i)}\,)$, and we claim that transition $(q_\ell^{uv}, \sigma'_\ell, q_\ell^{uv}) \in \widetilde{\delta}_j^{(i)}$ must have weight equal to $\varpi_i(\,\phi_j^{(i)}\,)$. To see this, realize that $\overline{q}' = q_0^{uv}, \ldots, q_\ell^{uv}, q_\ell^{uv}, q_{\ell+1}^{uv}, \ldots, q_{k-1}^{uv}$ is a finite accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over word $\overline{\sigma}'$. Therefore, if $\widetilde{\mathcal{W}}_j^{(i)}((q_\ell^{uv}, \sigma'_\ell, q_\ell^{uv})) = 0$ then the weight of run $\overline{q}'$ is equal to the weight of run $\overline{q}^{uv}$. Since only self-transitions of automaton $\widetilde{\mathcal{A}}_j^{(i)}$

have non-zero weight, this implies that there exists a finite word $\overline{\sigma}'' \in \Sigma^*$, obtained from word $\overline{\sigma}'$ by removing exactly m of its letters, such that there exists a finite accepting run of automaton $\mathcal{A}_j^{(i)}$ over word $\overline{\sigma}''$. However, this implies that the level of violation of word $\overline{\sigma}'$ with respect to formula $\phi_j^{(i)}$ is equal to $m\,\varpi_i(\,\phi_j^{(i)}\,)$, which contradicts our original assumption on $\overline{\sigma}'$. It follows that $\widetilde{\mathcal{W}}_j^{(i)}((q_\ell^{uv}, \sigma'_\ell, q_\ell^{uv})) = \varpi_i(\,\phi_j^{(i)}\,)$ and that $\overline{q}'$ is a finite accepting run of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ over word $\overline{\sigma}'$ with weight equal to $(m+1)\,\varpi_i(\,\phi_j^{(i)}\,)$.

Next, assume there exists a finite accepting run $\overline{q}''$ of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ with weight equal to $p\,\varpi_i(\,\phi_j^{(i)}\,)$, where $p < k+1$. Again, because only self-transitions of automaton $\widetilde{\mathcal{A}}_j^{(i)}$ have non-zero weight, this implies that there exists a finite word $\overline{\sigma}''' \in \Sigma^*$, obtained from word $\overline{\sigma}'$ by removing exactly p of its elements, such that there exists a finite accepting run of automaton $\mathcal{A}_j^{(i)}$ over word $\overline{\sigma}'''$. However, this implies that the level of violation of word $\overline{\sigma}'$ with respect to formula $\phi_j^{(i)}$ is equal to $p\,\varpi_i(\,\phi_j^{(i)}\,)$. ■

The second step in our procedure is to combine all the weighted versions of the automata associated with the pair $(\,\mathbf{A},\,\boldsymbol{\varpi}\,)$ into a single weighted finite automaton $\widetilde{\boldsymbol{\mathcal{A}}}$ such that the weights of its transitions capture the level of violation of a word with respect to the set of finite automata with priorities $(\,\mathbf{A},\,\boldsymbol{\varpi}\,)$.

Definition 4.16. Given a set of non-deterministic finite automata with priorities $(\,\mathbf{A},\,\boldsymbol{\varpi}\,)$, the *weighted product* associated with the pair $(\,\mathbf{A},\,\boldsymbol{\varpi}\,)$ is the weighted non-deterministic finite automaton $\widetilde{\boldsymbol{\mathcal{A}}} = (\,\widetilde{Q},\,\widetilde{q}_{init},\,\Sigma,\,\widetilde{\delta},\,\widetilde{F},\,\widetilde{\mathcal{W}})$ defined as follows, where automata $\mathcal{A}_j^{(i)}$ and $\widetilde{\mathcal{A}}_j^{(i)}$ are as in Definition 4.14 for each $1 \le i \le n$ and each $1 \le j \le m_i$, and weight function $\widetilde{\mathcal{W}}\colon \widetilde{\delta} \mapsto \mathbb{R}^n_{\ge 0}$ maps transitions to non-negative n-tuples.

- $\widetilde{Q} = \widetilde{Q}_1^{(1)} \times \cdots \times \widetilde{Q}_{m_1}^{(1)} \times \cdots \times \widetilde{Q}_1^{(n)} \times \cdots \times \widetilde{Q}_{m_n}^{(n)}$.
- $\widetilde{q}_{init} = (\,q_{init,1}^{(1)},\,\ldots,\,q_{init,m_i}^{(1)},\,\ldots,\,q_{init,1}^{(n)},\,\ldots,\,q_{init,m_n}^{(n)}\,)$.
- $(q, \sigma, r) \in \widetilde{\delta}$ if and only if $(\,q_j^{(i)},\,\sigma,\,r_j^{(i)}\,) \in \widetilde{\delta}_j^{(i)}$ for each $1 \le i \le n$ and each $1 \le j \le m_i$.
- $\widetilde{F} = F_1^{(1)} \times \cdots \times F_{m_1}^{(1)} \times \cdots \times F_1^{(n)} \times \cdots \times F_{m_n}^{(n)}$.
- $\widetilde{\mathcal{W}}(q, \sigma, r) = (\,w_1,\,\ldots,\,w_n\,)$ for each $(q, \sigma, r) \in \widetilde{\delta}$, where

$$w_i = \sum_{j=1}^{m_i} \left[\, \widetilde{\mathcal{W}}_j^{(i)}(\,q_j^{(i)},\,\sigma,\,r_j^{(i)}\,) \,\right], \qquad \text{for each } 1 \le i \le n\,.$$

Further, the weight of a run of automaton $\widetilde{\boldsymbol{\mathcal{A}}}$ over a finite run over Σ is the component-wise sum of the weights of the transitions taken along the run. □

Proposition 4.17. Let the pair $(\mathbf{A}, \boldsymbol{\varpi})$ and the finite automaton $\widetilde{\boldsymbol{\mathcal{A}}}$ be as in Definition 4.16. Then, automaton $\widetilde{\boldsymbol{\mathcal{A}}}$ accepts any finite word over Σ. Furthermore, if $\overline{q}$ is a shortest accepting run of automaton $\widetilde{\boldsymbol{\mathcal{A}}}$ over finite word $\overline{\sigma} \in \Sigma^*$ with respect to the Standard Lexicographical Ordering then $\widetilde{\mathcal{W}}(\overline{q}) = \lambda(\overline{\sigma}, \boldsymbol{\Phi})$. □

Proof: Directly from the definition of the Standard Lexicographical Ordering and Proposition 4.15. ■

The third step in our procedure is to build a special-purpose weighted product of the durational transition system $\mathcal{D}$ and the weighted automaton $\widetilde{\boldsymbol{\mathcal{A}}}$. In particular, the following definition is the analog of Definition 2.15 for the case where the weights of the transitions of the automaton are non-negative n-tuples.

Definition 4.18. Given a durational transition system $\mathcal{D} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)$ along with a goal set $S_{goal} \subseteq S$, and the automaton $\widetilde{\boldsymbol{\mathcal{A}}} = (\widetilde{Q}, \widetilde{q}_{init}, \Sigma, \widetilde{\delta}, \widetilde{F}, \widetilde{\mathcal{W}})$ associated with the pair $(\mathbf{A}, \boldsymbol{\varpi})$, the *product* of $\mathcal{D}$ and $\widetilde{\boldsymbol{\mathcal{A}}}$, denoted $\mathcal{D} \otimes \widetilde{\boldsymbol{\mathcal{A}}}$, is the weighted non-deterministic finite automaton $\widetilde{\mathcal{P}}_{\mathcal{W}} = (\widetilde{\mathcal{P}}, \widetilde{\mathcal{W}}_{\mathcal{P}})$, where $\widetilde{\mathcal{P}} = (\widetilde{Q}_{\mathcal{P}}, \widetilde{q}_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \widetilde{\delta}_{\mathcal{P}}, \widetilde{F}_{\mathcal{P}})$ is a non-deterministic finite automaton and $\widetilde{\mathcal{W}}_{\mathcal{P}} \colon \widetilde{\delta}_{\mathcal{P}} \mapsto \mathbb{R}^n_{\geq 0}$ is a function mapping transitions to non-negative $(n+1)$-tuples. Specifically, automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ is defined as follows.

- $\widetilde{Q}_{\mathcal{P}} = S \times \widetilde{Q}$.
- $\widetilde{q}_{init,\mathcal{P}} = (s_{init}, \widetilde{q}_{init}) \in Q_{\mathcal{P}}$.
- $\Sigma_{\mathcal{P}} = \varnothing$.
- $((s,q),(s',q')) \in \widetilde{\delta}_{\mathcal{P}}$ if and only if $(s,s') \in \mathcal{R}$ and $(q,(\mathcal{L}(s),\mathcal{L}(s')),q') \in \widetilde{\delta}$.
- $\widetilde{F}_{\mathcal{P}} = S_{goal} \times \widetilde{F}$.
- $\widetilde{\mathcal{W}}_{\mathcal{P}}(((s,q),(s',q'))) = (\Delta(s,s')\, w_1, \ldots, \Delta(s,s')\, w_n, \Delta(s,s'))$ for each $((s,q),(s',q')) \in \widetilde{\delta}_{\mathcal{P}}$, where $(w_1, \ldots, w_n) = \widetilde{\mathcal{W}}((q,(\mathcal{L}(s),\mathcal{L}(s')),q'))$.

Further, the weight of a finite run of automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ is the component-wise sum of the weights of the transitions taken along the run. □

Proposition 4.19. Let system $\mathcal{D}$ and automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ be as in Definition 4.18. If $\overline{q}$ is a shortest accepting run of automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ with respect to the Standard Lexicographical Ordering then the projection of run $\overline{q}$ onto set S solves Problem 4.13. □

Proof: Directly from Proposition 4.17 and the definitions of set of accepting states $\widetilde{F}_{\mathcal{P}}$ and transition weight function $\widetilde{\mathcal{W}}_{\mathcal{P}}$. ■

The fourth and final step of in procedure is to run Dijkstra's Shortest Path Algorithm on automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ in order to obtain the shortest accepting run of the automaton, and to then project said run onto a trace of the durational transition system $\mathcal{D}$. The reader is referred to [CLR+09] for a detailed exposition of the aforementioned algorithm. Regarding guarantees, combining Proposition 4.19 with the correctness of Dijkstra's Shortest Path Algorithm, we realize that this procedure does in fact solve Problem 4.13 and that it has a worst-case computational complexity in $\mathcal{O}(\,|\widetilde{\mathcal{P}}_{\mathcal{W}}|\log|\widetilde{\mathcal{P}}_{\mathcal{W}}|\,)$.

4.3.2 Incremental Construction of Automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$

In this subsection, we explain how to construct the weighted finite automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ given in Definition 4.18 one state at a time. That is, we present an algorithm, which we have named Minimum-violation RRT*(MVRRT*), that builds the aforementioned automaton *incrementally* via random sampling of the collision-free space X_{free} of dynamical system $\mathcal{S}$. The method, which we present in Algorithm 4.1, uses the following definitions.

- Function `sample`: $\mathbb{N} \mapsto X_{free}$ returns independent and identically distributed states of the collision-free space X_{free} drawn from the uniform distribution.
- Constant $\gamma_{RRT^*} \in \mathbb{R}_{>0}$ satisfies

$$\gamma_{RRT^*} > 2\left[\left(2+\frac{1}{D}\right)\frac{\mu(X_{free})}{V_X}\right]^{1/D}$$

  where constant D is the Hausdorff dimension associated with dynamical system $\mathcal{S}$, constant V_X is the volume of the ball of unit radius in X, and function μ is the Lebesgue measure. The reasoning behind these choices is best explained in [KF11], [KF12] and [KF13], where the RRT*algorithm for holonomic and non-holonomic dynamical systems was first introduced.
- Function `min_time`: $X_{free} \times X_{free} \mapsto \mathbb{R}_{\geq 0}$ returns the duration of a minimum-time trajectory of the dynamical system $\mathcal{S}$ through state space X starting and ending at the given states. Note that it is not necessarily the case that a time-optimal trajectory $z^*\colon [0,T^*] \mapsto X$ from a state to another satisfies $z^*(t) \in X_{free}$ for all $t \in [0,T^*]$, *i.e.,* trajectories associated with `min_time` need not be feasible.
- Function $\overleftarrow{\texttt{Box}}$: $X_{free} \times \mathbb{N} \mapsto 2^X$ returns, for any state x and any iteration N, a scaled box of states in X aligned with vector $f(x,0)$ of volume equal to $\gamma_{RRT^*}\log(N)\,/\,N$.

Furthermore, $\overleftarrow{\texttt{Box}}(x, N)$ obeys the condition

$$\lim_{N\to\infty} \left\{ \max_{z\in \texttt{Box}(x,N)} \texttt{min\_time}(x,z) \right\} = 0\,, \quad \forall\, x \in X_{free}\,, \tag{4.1}$$

meaning that it approximates the sequence of sets of states reachable from state x in increasingly shorter times. More precise information on how the box returned by this function should be scaled and oriented is available in [KF13].

- Function $\overrightarrow{\texttt{Box}}\colon X_{free} \times \mathbb{N} \mapsto 2^X$ is identical to function $\overleftarrow{\texttt{Box}}$, except that instead of obeying Equation 4.1, it satisfies the condition

$$\lim_{N\to\infty} \left\{ \max_{z\in \texttt{Box}(x,N)} \texttt{min\_time}(z,x) \right\} = 0\,, \quad \forall\, x \in X_{free}\,,$$

meaning that it approximates the sequence of sets of states from which state x can be reached in increasingly shorter times.

- Function $\overleftarrow{\texttt{reach}}\colon X_{free} \times \mathbb{N} \mapsto 2^S$ returns the subset of S, the set of states of dura-tional transition system $\mathcal{D}$ constructed inside Algorithm 4.1, such that for any state $x \in X_{free}$ and any iteration N, state $s \in S$ belongs to set $\overleftarrow{\texttt{reach}}(x, N)$ if and only if $s \in \overleftarrow{\texttt{Box}}(x, N)$ and the time-optimal trajectory $z^*\colon [0, T^*] \mapsto X$ from state x to state s satisfies $z^*(t) \in X_{free}$ for all $t \in [0, T^*]$ and $|D(z^*)| \leq 2$.
- Function $\overrightarrow{\texttt{reach}}\colon X_{free} \times \mathbb{N} \mapsto 2^S$ returns the subset of S such that for any state $x \in X_{free}$ and any iteration N, state $s \in S$ belongs to set $\overrightarrow{\texttt{reach}}(x, N)$ if and only if $s \in \overrightarrow{\texttt{Box}}(x, N)$ and the time-optimal trajectory $z^*\colon [0, T^*] \mapsto X$ from state s to state x satisfies $z^*(t) \in X_{free}$ for all $t \in [0, T^*]$ and $|D(z^*)| \leq 2$.

Since Algorithm 4.1 is an extension of the RRT*Algorithm, it offers Asymptotic Optimality, as well as an expected logarithmic computational complexity per iteration. Here, the first guarantee states that almost surely, *i.e.,* with probability one, the sequence of trajectories $x_1^*, x_2^*, \ldots$ of dynamical system $\mathcal{S}$ returned by MVRRT*after $1, 2, \ldots$ iterations converges to a solution of Problem 4.12 in the sense of the total variation. The second guarantee states that the expected amount of computation involved with MVRRT*is within a constant factor of that required to find a shortest path in a graph.

Proposition 4.20. Consider Problem 4.12, and suppose it has a non-empty solution set. Let $\mathcal{X}^*$ denote the set of all solutions at which the functional $\lambda(\, w_\tau(\,\cdot\,), \phi_j^{(i)}\,)$ is continuous for each $1 \leq i \leq n$ and each $1 \leq j \leq m_i$, and let $x_N\colon [0, T_N] \mapsto X_{free}$ be the trajectory returned by Algorithm 4.1 after N iterations. Then, the following convergence takes place

Algorithm 4.1: MVRRT*

Input : Continuous-time continuous-space dynamical system $\mathcal{S} = (X, U, f)$, along with collision-free space $X_{free} \subseteq X$, initial state $x_{init} \in X_{free}$, goal set X_{goal}, finite alphabet Π, and labeling function $\mathcal{L}_{cts}\colon X \mapsto 2^{\Pi}$; non-deterministic finite automaton $\widetilde{\boldsymbol{\mathcal{A}}} = (\widetilde{Q}, \widetilde{q}_{init}, \Sigma, \widetilde{\delta}, \widetilde{F}, \widetilde{\mathcal{W}})$ as in Definition 4.16, where $\Sigma = 2^{\Pi \times \Pi}$; desired number of iterations N.

Output: If a solution to Problem 4.12 exists, then a trajectory $x^*\colon [0, T] \mapsto X_{free}$ of dynamical system $\mathcal{S}$ which approaches a solution to Problem 4.12 as $n \to \infty$; otherwise, nothing.

$s_{init} = x_{init}$; $S = \{ s_{init} \}$; $\mathcal{R} = \varnothing$; $\mathcal{L}$ is $\mathcal{L}_{cts}$ restricted to S; $\Delta\colon \mathcal{R} \mapsto \mathbb{R}_{\geq 0}$;

construct durational transition system $\mathcal{D} = (S, s_{init}, \mathcal{R}, \Pi, \mathcal{L}, \Delta)$;

construct automaton $\widetilde{\mathcal{P}}_{\mathcal{W}} := \mathcal{D} \otimes \widetilde{\boldsymbol{\mathcal{A}}}$, where $\widetilde{\mathcal{P}}_{\mathcal{W}} := (\widetilde{Q}_{\mathcal{P}}, \widetilde{q}_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \widetilde{\delta}_{\mathcal{P}}, \widetilde{F}_{\mathcal{P}}, \widetilde{\mathcal{W}}_{\mathcal{P}})$;

define $\texttt{parent}\colon \widetilde{Q}_{\mathcal{P}} \mapsto \{\perp\} \cup \widetilde{Q}_{\mathcal{P}}$, and let $\texttt{parent}(\widetilde{q}_{init,\mathcal{P}}) = \widetilde{q}_{init,\mathcal{P}}$;

define $J\colon \widetilde{Q}_{\mathcal{P}} \mapsto \mathbb{R}^{n+1}_{\geq 0}$, and let $J(\widetilde{q}_{init,\mathcal{P}}) := (0, \ldots, 0, 0)$;

foreach $1 \leq i \leq N$ **do**

  $s_{new} := \texttt{sample\_state}(i)$;

  $candidates := \varnothing$;

  foreach $s \in \overrightarrow{\texttt{reach}}(s_{new}, |S|)$ **do**

    $\texttt{connect}(s, s_{new})$;

    $candidates := candidates \cup \{ s \}$;

  $\texttt{update\_parents}(s_{new}, candidates)$;

  foreach $s \in \overleftarrow{\texttt{reach}}(s_{new}, |S|)$ **do**

    $\texttt{connect}(s_{new}, s)$;

    $\texttt{update\_parents}(s, \{\texttt{Parent}(s), s_{new}\})$;

compute $\overline{q}^*$, the shortest accepting run of automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$, by running Dijkstra's Shortest Path Algorithm from the initial state;

compute $\overline{s}^* = s^*_0, \ldots, s^*_{k-1}$, the projection of run $\overline{q}^*$ onto set S;

compute $x^*_N\colon [0, T^*] \mapsto X_{free}$, the trajectory of dynamical system $\mathcal{S}$ obtained by sequentially concatenating trajectories x^*_0 through x^*_{k-2}, where $x^*_j\colon [0, T^*_j] \mapsto X_{free}$ is a minimum-time trajectory with initial state $s_j \in S$ and final state $s_{j+1} \in S$;

return trajectory x^*_N;

Algorithm 4.2: `connect` (Executed within `MVRRT*`.)

Input : State $s \in S$ of durational transition system $\mathcal{D}$; state $s' \in X$ of dynamical system $\mathcal{S}$.

if $s' \notin S$ **then**
- $S := S \cup \{s'\}$;
- $\widetilde{Q}_{new,\mathcal{P}} := \{s'\} \times \widetilde{Q}$;
- **foreach** $\widetilde{q}_{\mathcal{P}} \in \widetilde{Q}_{new,\mathcal{P}}$ **do**
  - $\texttt{parent}(\widetilde{q}_{\mathcal{P}}) = \perp$ and $J(\widetilde{q}_{\mathcal{P}}) = (\infty, \ldots, \infty, \infty)$;
- $\widetilde{Q}_{\mathcal{P}} := \widetilde{Q}_{\mathcal{P}} \cup \widetilde{Q}_{new,\mathcal{P}}$;
- **if** $s' \in X_{goal}$ **then** $\widetilde{F}_{\mathcal{P}} := \widetilde{F}_{\mathcal{P}} \cup \widetilde{Q}_{new,\mathcal{P}}$;

$\mathcal{R} := \mathcal{R} \cup \{(s, s')\}$;

$\Delta(s, s') := \texttt{min\_time}(s, s')$;

foreach $q, q' \in \widetilde{Q}$ such that $(s, q) \in \widetilde{Q}_{\mathcal{P}}$ and $(q, (\mathcal{L}(s), \mathcal{L}(s')), q') \in \widetilde{\delta}$ **do**
- $\widetilde{\delta}_{\mathcal{P}} := \widetilde{\delta}_{\mathcal{P}} \cup \{((s, q), (s', q'))\}$;
- $\widetilde{\mathcal{W}}_{\mathcal{P}}(((s, q), (s', q'))) := (\Delta(s, s')\, w_1, \ldots, \Delta(s, s')\, w_n, \Delta(s, s'))$, where $(w_1, \ldots, w_n) = \widetilde{\mathcal{W}}((q, (\mathcal{L}(s), \mathcal{L}(s')), q'))$;

Algorithm 4.3: `update_parent` (Executed within `MVRRT*`.)

Input : State $s \in S$ of durational transition system $\mathcal{D}$; set of states $S_{update} \subseteq S$ of durational transition system $\mathcal{D}$.

foreach $q \in \widetilde{Q}$ **do**
- $\widetilde{Q}_{par,\mathcal{P}} :=$ $\{(s', q') \in \widetilde{Q}_{\mathcal{P}} \colon s' \in S_{update}, \texttt{Parent}((s', q')) \neq \perp, ((s', q'), (s, q)) \in \widetilde{\delta}_{\mathcal{P}}\}$;
- $\texttt{Parent}((s, q)) := \underset{(s', q') \in \widetilde{Q}_{par,\mathcal{P}}}{\arg\min} J((s', q')) + \widetilde{\mathcal{W}}_{\mathcal{P}}(((s', q'), (s, q)))$;
- $J((s, q)) := J(\texttt{Parent}(s, q)) + \widetilde{\mathcal{W}}_{\mathcal{P}}((\texttt{Parent}(s, q), (s, q)))$;

almost surely, *i.e.,* with probability one.

$$\limsup_{N\to\infty} \left\{ \inf_{z^*\in\mathcal{X}^*} d_{TV}(x^*_N, z^*) \right\} = 0$$

Further, if $\lambda^* = \lambda(\, x^*,\, \mathbf{\Phi})$ and $c^* = c(x^*)$ for any $x^* \in \mathcal{X}^*$, then, almost surely,

$$\limsup_{N\to\infty}\ \lambda(\, x^*_N,\, \mathbf{\Phi}) = \lambda^* \qquad \text{and} \qquad \limsup_{N\to\infty}\ c(x^*_N) = c^*\ .$$

□

Proof Sketch: The result is a straightforward extension of the proofs of the convergence to optimal trajectories of the RRT*Algorithm for dynamical systems, which were first published in [KF11], [KF12] and [KF13]. Roughly speaking, the proof of this proposition can be obtained by referring to the proof of Theorem 38 in [KF11] and applying the following modifications to the arguments.

- Replace the graph $G = (\, V,\, E\,)$ with the automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$.
- For any iteration N, instead of covering optimal trajectory $x^*\colon [0, T^*] \mapsto X_{free}$ with Euclidean-norm balls of volume proportional to $\log(N)\,/\,N$, we cover x^* with the sets returned by the functions $\overrightarrow{\texttt{Box}}$ and $\overleftarrow{\texttt{Box}}$ bloated by a constant factor. More precisely, our covering of trajectory x^* at iteration N, which we denote as $\texttt{CoverBoxes}(\, x^*,\, N)$, is such that for any state z of x^* at least one of the following inequalities holds.

$$\mu\left(\overrightarrow{\texttt{Box}}(\, z,\, N) \cap \texttt{CoverBoxes}(\, x^*,\, N)\right) > \gamma_{RRT^*}\,\frac{\log(N)}{N}$$
$$\mu\left(\overleftarrow{\texttt{Box}}(\, z,\, N) \cap \texttt{CoverBoxes}(\, x^*,\, N)\right) > \gamma_{RRT^*}\,\frac{\log(N)}{N}$$

Finally, the last two statements follow from our assumptions on the continuity at trajectory x^* of the level of violation and cost functionals. ■

Proposition 4.21. Consider Algorithm 4.1. The expected computational complexity of executing this algorithm up to iteration N is in $\mathcal{O}(\, |\widetilde{Q}|^2\, N\, \log(N)\,)$, where $\widetilde{Q}$ is the set of states of automaton $\widetilde{\boldsymbol{\mathcal{A}}}$. □

Proof: Assume the usage of a K-D Tree to store automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$, and recall that we sample uniformly and independently from X_{free}. Because of our choice for the volume of the sets returned by function $\overrightarrow{\texttt{Box}}$ it is the case that at any iteration both the expected computational complexity of a call to function $\overrightarrow{\texttt{reach}}$ and the expected size of the set returned this

function are in $\mathcal{O}(\log(N))$. The same argument applies to function $\overleftarrow{\texttt{reach}}$. In addition, notice that procedure `connect` adds at most $|\widetilde{Q}|^2$ transitions to set $\widetilde{\delta}_{\mathcal{P}}$ per call, and that the computational complexity of a call to procedure `update_parent` is in $\mathcal{O}(|\widetilde{Q}|\,|S_{update}|)$. Combining these facts, we observe that the expected computational complexity per iteration incurred in executing Lines 6-15 of our algorithm is in $\mathcal{O}(|\widetilde{Q}|^2 \log(N))$. The result then follows by recalling the worst-case computational complexity of running Dijkstra's Shortest Path Algorithm on a tree with $\mathcal{O}(|\widetilde{Q}|\,N)$ nodes. ■

4.4 Simulations and Experiments

In this section we consider the problem of guiding an autonomous passenger car traveling in an urban environment subject to road safety rules. We use this application to evaluate the performance of Algorithm 4.1 (`MVRRT*`) in several different simulation scenarios. In addition, we implement the algorithm in a real-time experimental testbed.

4.4.1 Dynamical System, Alphabet and Labeling Function

We model our robot as a Dubins Car, which is a robot that moves at constant speed subject to a minimum turning radius constraint. More precisely, we consider the continuous-time continuous-space dynamical system $\mathcal{S} = (X, U, f)$ with state space $X = \mathbb{R}^2 \times \mathbb{S}$, input space $U = [-1, 1]$ and dynamics of the form

$$f((x, y, \theta), u) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ u \end{bmatrix}$$

where $v \in \mathbb{R}_{>0}$ is the speed of the car. As first shown in [Dub57], time-optimal trajectories of this particular dynamical system can be easily computed. Specifically, the desired trajectory is obtained by calculating the duration of six candidate trajectories and then choosing the trajectory with the shortest duration. Out of these six trajectories, only four of them occur with non-zero probability in the case where the endpoints are sampled uniformly and independently from the collision-free space X_{free}. Each of these four trajectories consists of the concatenation of a circular arc, a straight-line segment, and another circular arc, and so its duration can be calculated in closed form. For more information on Dubins Cars

the reader is referred to [LaV06], which also covers the computation of time-optimal trajectories for other types of dynamical systems.

The generic setup for our simulations is shown in Figure 4-1. We partition the robot's state space X into the obstacle space X_{obs} and the collision-free space X_{free}. The set X_{obs} is represented in Figures 4-1a and 4-1b as the union of all the red and dark red regions with the space that lies outside the bold black rectangle, while the set $X_{free} = X \setminus X_{free}$. Further, we partition X_{free} into the sets X_{SW}, X_{DIR}, X_{SL} and X_{DL}. Here, set X_{SW} is associated with the sidewalks, which are represented in Figures 4-1a and 4-1b as the union of all the dark grey regions. Sets X_{SL} and X_{DL} contain the states in X_{free} for which the footprint of the car intersects any one of the single or double lane lines, respectively. Set X_{DIR} corresponds to all states for which the direction of the car's velocity vector matches the lane on which the center of the car's footprint lies. Notice that in our simulations, the direction of the lanes is understood in the sense of the American driving conventions, whereas in our experiments the direction of the lanes is understood in the sense of the British driving conventions.

The alphabet associated with our problem is $\Pi = \{ sw, sl, dl, dir\}$, and the corresponding labeling function $\mathcal{L}_{cts}\colon X \mapsto 2^{\Pi}$ is defined such that $sw \in \mathcal{L}_{cts}(x)$ if $x \in X_{SW}$, such that $sl \in \mathcal{L}_{cts}(x)$ if $x \in X_{SL}$, such that $dl \in \mathcal{L}_{cts}(x)$ if $x \in X_{DL}$, and such that $dir \in \mathcal{L}_{cts}(x)$ if $x \in X_{DIR}$. For notational convenience, for any $\pi_1, \pi_2 \in \Pi \cup \{\texttt{true}\}$, we let $(\pi_1, \pi_2) \subseteq 2^{\Pi} \times 2^{\Pi}$ denote the set of all pairs of subsets of Π that contain symbol π_1 in its first entry and symbol π_2 in its second entry. Finally, notice that we do not associate any symbol with the obstacle space X_{obs}, since we do not want Algorithm 4.1 to maintain or produce any trajectory that collides with an obstacle.

4.4.2 Road Safety Rules

We require our vehicle to travel through the collision-free space X_{free} from starting state x_{init} to goal set X_{goal}, which are represented by the white car and the green region in Figures 4-1a and 4-1b, respectively. With this setup in mind, we are interested in finding a trajectory that obeys the following road safety rules as much as possible among all collision-free trajectories from state x_{init} to set X_{goal}.

(i) Do not drive on sidewalks.

(ii) If driving in the direction of the lane, do not cross the double-lane line.

(iii)-(a) Do not cross the single lane line.

(iii)-(b) Always drive in the direction of the lane.

The reasoning behind this choice of rules and priorities is quite simple, and so illustrates one of the advantages of our contribution. In particular, rule *(i)* is given the highest priority because driving on the sidewalk could endanger pedestrians if the vehicle's sensors fail to detect them. Similarly, since double lane lines are usually associated with high-speed driveways or segments of the road with low forward visibility, failure of the vehicle's sensors to detect incoming cars results in much more serious consequences when crossing a double line than when crossing a single line. Hence rule *(ii)* is given the second highest priority, and rules *(iii)-(a)* and *(iii)-(b)* are given the third highest priority. Notice that the second rule penalizes a transition that crosses the double lane line only when the transition starts inside set X_{DIR}, whereas rule *(iii)-(a)* penalizes a transition that crosses the single lane line independently of the transition's starting state. This distinction thus penalizes trajectories that cross the single lane line too frequently, as this behavior could endanger the driver traveling behind our vehicle, if any. As shown in Figure 4-5b, excessively frequent single lane line crossing could occur when maneuvering around parked car, for instance.

We model our four road safety rules as the following SFLTL$_{-\mathsf{X}}$ formulae.

$$\begin{aligned}
\phi_{SW} &= \mathsf{G}\,(\,\neg\,(\,(\texttt{true}, sw) \vee (sw, sw) \vee (sw, \texttt{true})\,)\,) \\
\phi_{DL} &= \mathsf{G}\,(\,\neg\,(dir, dl)\,) \\
\phi_{SL} &= \mathsf{G}\,(\,\neg\,(\,(\texttt{true}, sl) \vee (sl, sl) \vee (sl, \texttt{true})\,)\,) \\
\phi_{DIR} &= \mathsf{G}\,(dir, dir)
\end{aligned}$$

Here formula ϕ_{SW} is associated with rule *(i)* and states that no transition should either start or end at a state of set X_{SW}. Formula ϕ_{DL} corresponds to rule *(ii)* and states that no transition should start in a state of set X_{DIR} and then end in a state of set X_{DL}. Formula ϕ_{SL} is associated with rule *(iii)-(a)* and states that no transition should either start or end at a state of set X_{SL}. Formula ϕ_{DIR} is associated with rule *(iii)-(b)* and states that every transition should both start and end at a state of set X_{DIR}.

Combining these four safety specifications we can construct the set of SFLTL$_{-\mathsf{X}}$ formalae with priorities ($\mathbf{\Phi}$, $\boldsymbol{\varpi}$) defined such that

$$\mathbf{\Phi} = \{\,\{\phi_{SW}\},\, \{\phi_{DL}\},\, \{\,\phi_{SL},\, \phi_{DIR}\,\}\,\}$$

and such that $\varpi_1(\phi_{SW}) = \varpi_2(\phi_{DL}) = \varpi_3(\phi_{DIR}) = 1$ and $\varpi_3(\phi_{SL}) = 10$. Notice that we have chosen to penalize rules *(iii)-(a)* and *(iii)-(b)* in such a way that one second of

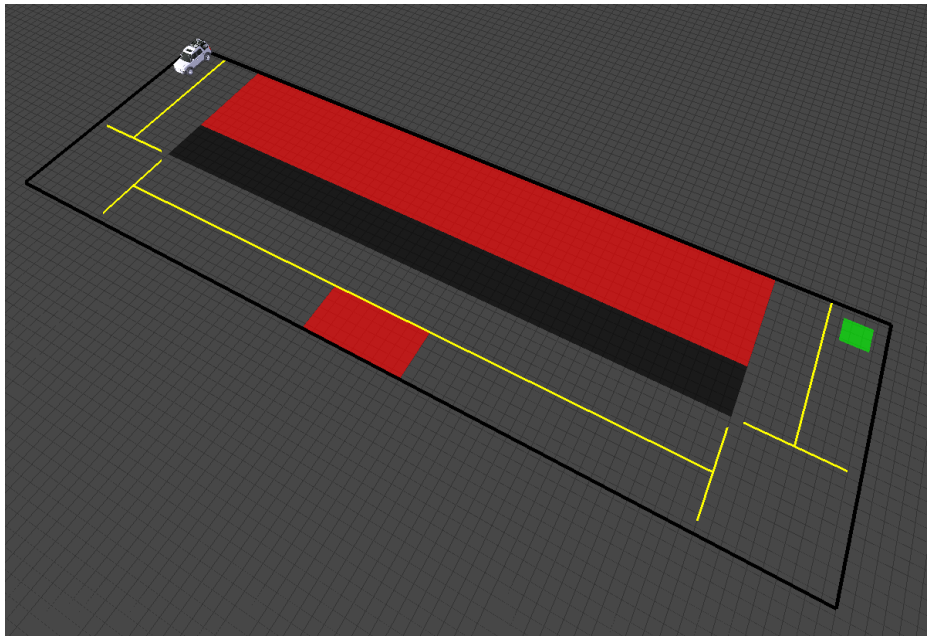
violation of rule *(iii)-(a)* is equivalent to ten seconds of violation of rule *(iii)-(b)*.

Remark 4.22. Since we model our vehicle as a rigid body, it cannot cross from one side of the single lane line to the other in less than L/v time units, were $L \in \mathbb{R}_{>0}$ is the longitudinal length of the vehicle. Combining this fact with our assumption that durational transition system $\mathcal{D}$ is trace-inclusive with respect to dynamical system $\mathcal{S}$, we realize that no transition of the automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ constructed in Algorithm 4.1 can cross the single lane line without the crossing being reflected in the level of violation of the transition with respect to formula ϕ_{SL}. A similar argument applies for the case of the double lane line. □

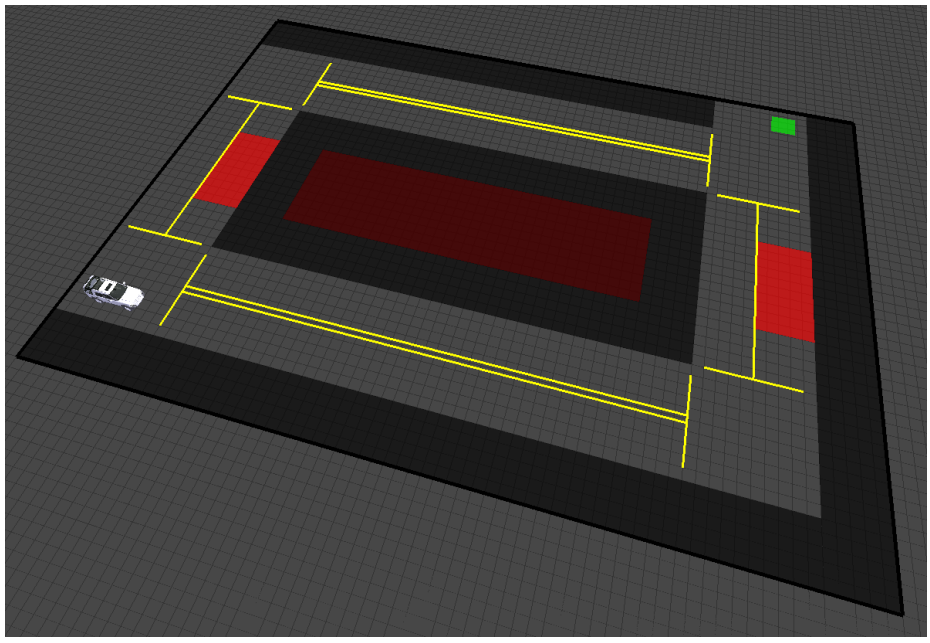
4.4.3 Simulation Results

In order to solve Problem 4.12 for this autonomous urban driving application, we implement Algorithm 4.1 in C++ on a 3.10 GHz Intel® i5-2400 processor with 4 GB of RAM. We now describe our simulations and discuss the results, which are as follows.

- **Case 0:** We run the generic setup in Figure 4-1a with no obstacles on the road. Figure 4-2a shows the states and transitions of the durational transition system $\mathcal{D}$ constructed during the first 500 iterations of the algorithm. After 10 seconds of computation the algorithm returns a trajectory of zero level of violation which is close to the solution, as seen in Figure 4-2b. Notice that in this case the solution is not the minimum-time trajectory, since the minimum-time trajectory involves driving over the sidewalk.
- **Case 1-A:** We run the generic setup in Figure 4-1a. As seen in Figure 4-3, after 10 seconds of computation the algorithm returns a trajectory close to the solution, which in this case involves violating formulae ϕ_{SL} and ϕ_{DIR}.
- **Case 1-B:** Now the obstacle blocks the entire road. As seen in Figure 4-4, after 10 seconds of computation the algorithm returns a trajectory close to the solution, which in this case involves violating formulae ϕ_{SW}, ϕ_{SL} and ϕ_{DIR}.
- **Case 1-C:** Now there are two obstacles sitting on the right lane but separated by some distance. As seen in Figure 4-5a, after 30 seconds of computation the algorithm returns a trajectory close to the solution, which in this case involves violating formulae ϕ_{SL} and ϕ_{DIR}. For comparison, notice that if we remove formula ϕ_{SL}, *i.e.,* rule *(iii)-(a)*, then the solution involves exiting and entering the right lane twice, which could confuse a human driver following our autonomous vehicle. Figure 4-5b shows the trajectory returned by the algorithm in this case, after 30 seconds of

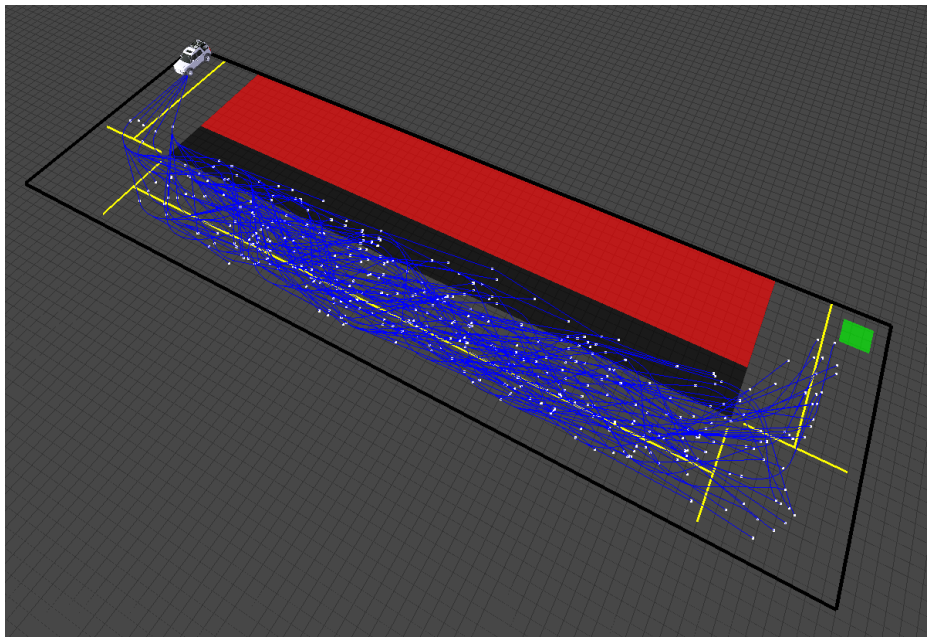


(a) Setup for Cases 0, 1-A, 1-B and 1-C.

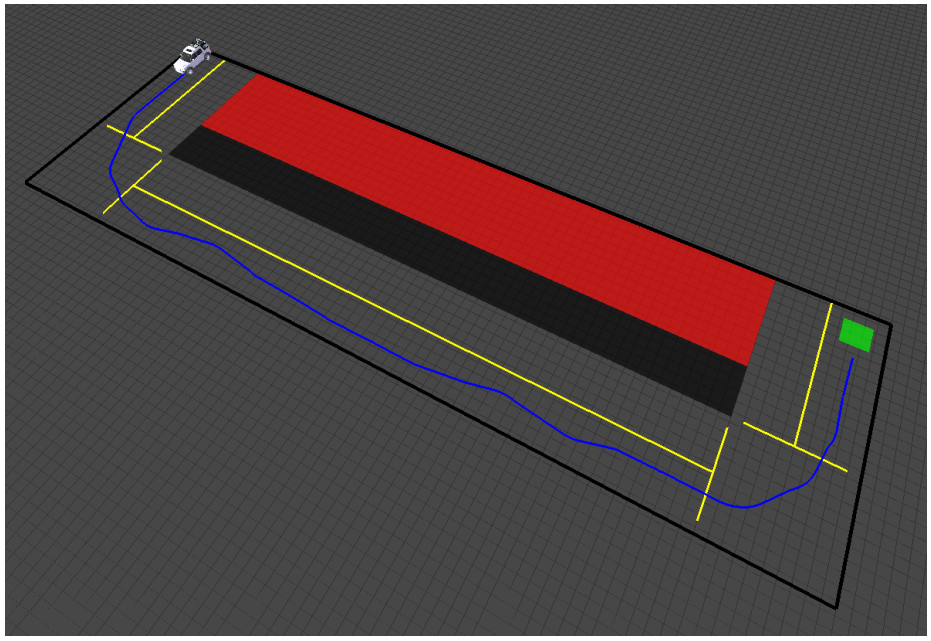


(b) Setup for Cases 2-A and 2-B.

Figure 4-1: Generic setups for our simulations.



(a)



(b)

Figure 4-2: MVRRT* simulation results: Case 0.

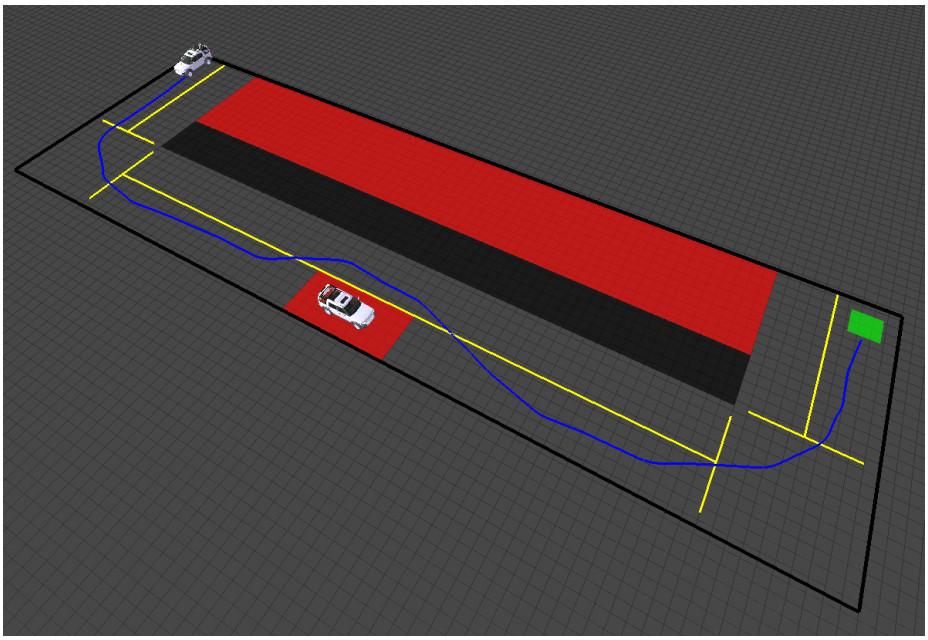


Figure 4-3: MVRRT*simulation results: Case 1-A.

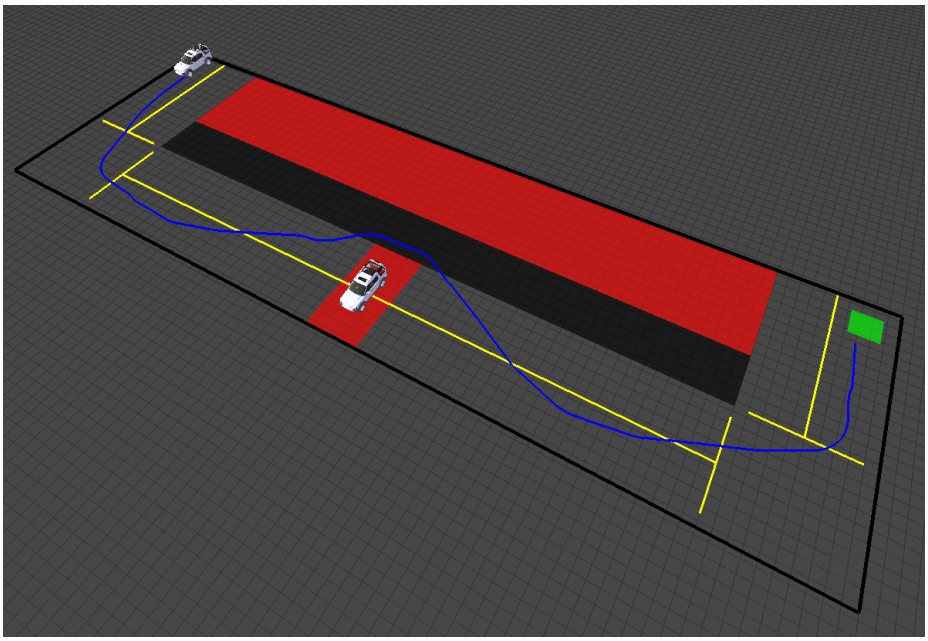


Figure 4-4: MVRRT*simulation results: Case 1-B.

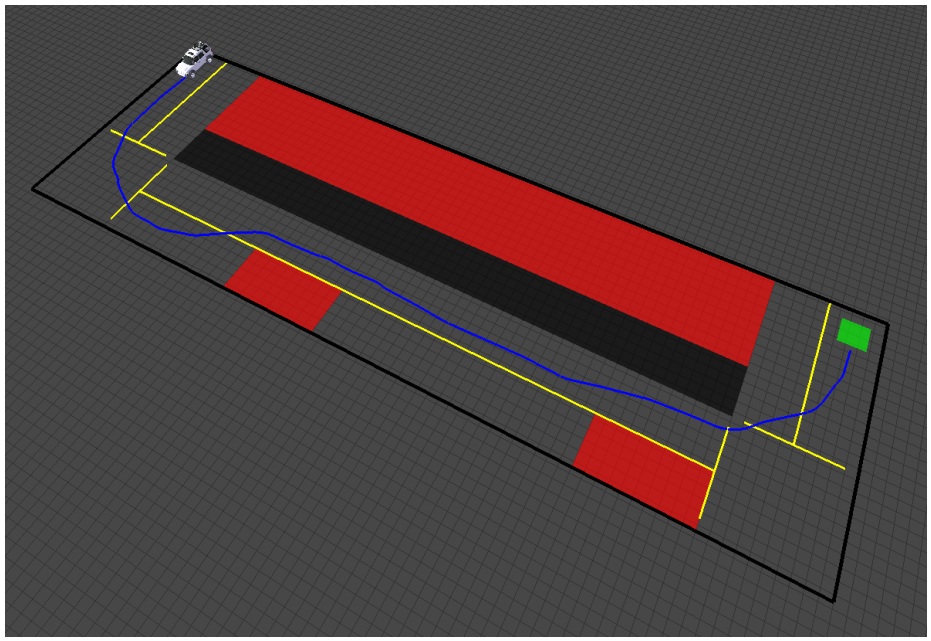
computation.

- **Case 2-A**: We run the generic setup in Figure 4-1b. As seen in Figure 4-6, after 30 seconds of computation the algorithm returns a trajectory close to the solution, which in this case involves taking the South road and then taking the East road, thus violating formulae ϕ_{SL} and ϕ_{DIR}. Note that it may very well be the case that there also exists a solution to the problem which involves taking the West road and then the North road, as this route would also violate formulae ϕ_{SL} and ϕ_{DIR} for a short time interval.
- **Case 2-B:** Now the obstacle which used to sit on the East road has been placed on the South road. Thus if our vehicle takes the South road and then the East road it would need to violate formula ϕ_{DL} in order to sort the obstacle on the South road, which is undesirable. However, after 60 seconds of computation the algorithm returns a trajectory close to the solution, which in this case involves taking the West road and then the North road, therefore violating formulae ϕ_{SL} and ϕ_{DIR} when sorting the obstacle on the West road but not formula ϕ_{DL}.

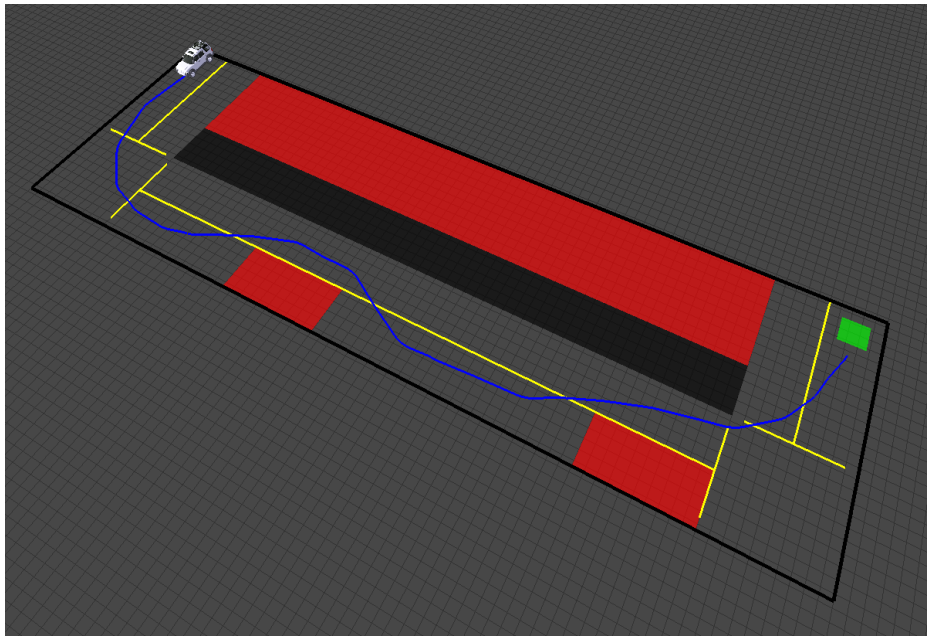
Implementation on a Real-Time Experimental Testbed

We also implemented Algorithm 4.1 on the autonomous golf cart shown in Figure 4-8a and carried real-time autonomous driving experiments on the campus of the National University of Singapore (NUS). The golf cart has drive-by-wire capability and is instrumented with two front-facing SICK LMS-200 laser range finders. Our algorithm interfaced with the vehicle's sensing and closed-loop tracking control modules through the Robot Operating System, which is an open source software standard curated by Willow Garage®.

We made some slight modifications to Algorithm 4.1 so that it could be run in real time. In particular, we extended procedure `update_parent` such that whenever an existing state's parent was changed to the newly sampled state the levels of violation and costs of all the states in the sub-tree rooted at the existing node were updated. These updates allowed us to keep track of the accepting state of automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ with the current best level of violation and cost. By simply backtracking from the current best state we were able to maintain the current best trajectory, and thus we had the ability to terminate our algorithm at any iteration without needing to execute Line 16. Of course, these updates come at the price of extra computation, although empirical evidence suggests that they only increase the computational complexity of our algorithm by a constant factor.



(a)



(b)

Figure 4-5: MVRRT*simulation results: Case 1-C.

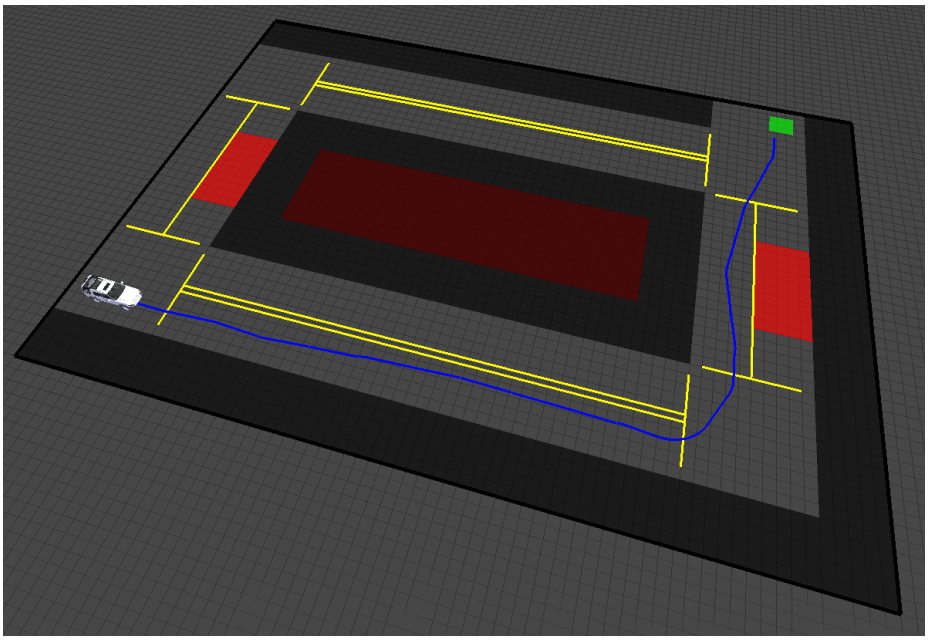


Figure 4-6: MVRRT* simulation results: Case 2-A.

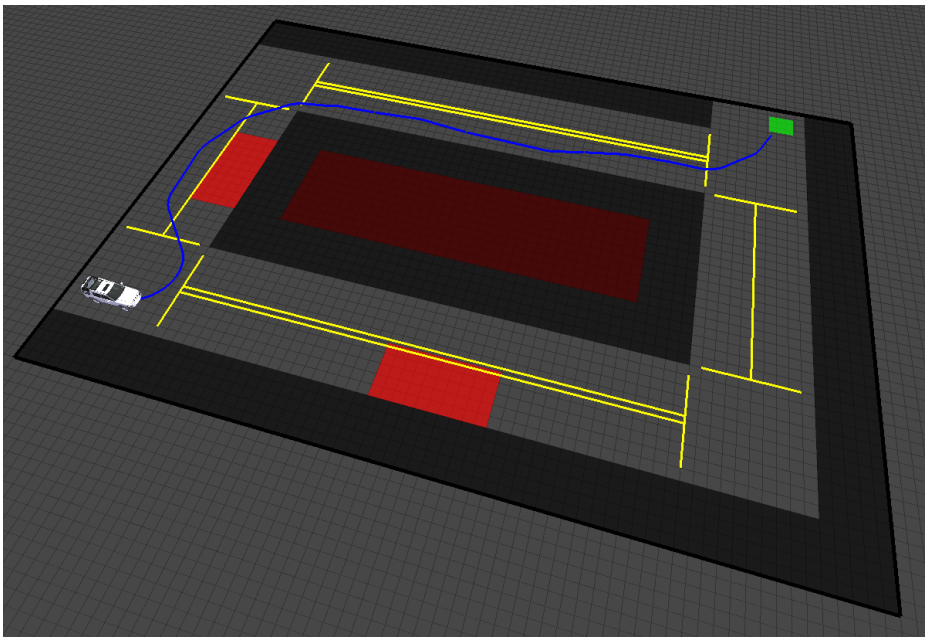


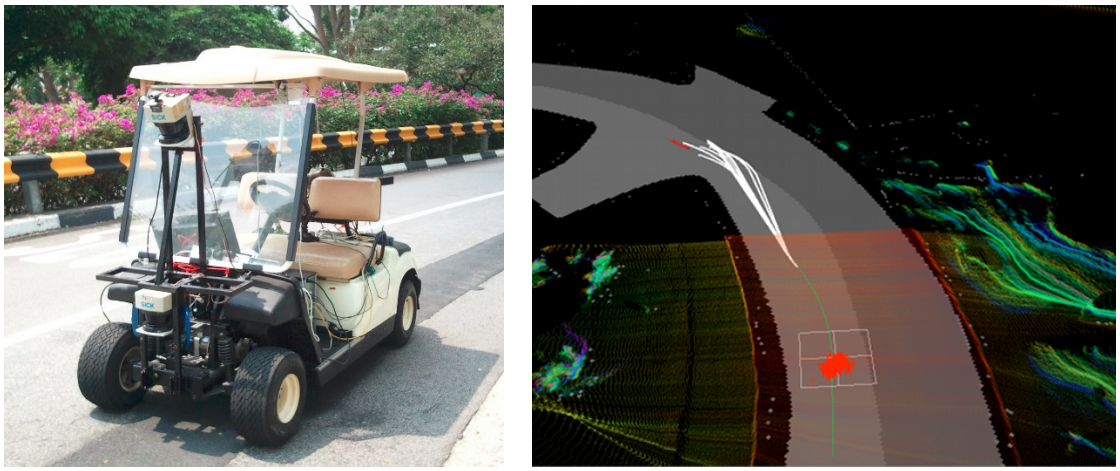
Figure 4-7: MVRRT* simulation results: Case 2-B.

Additionally, in order to speed-up computation we also introduced both branch-and-bound and biased sampling into our algorithm. Branch-and-bound allowed us to connect to automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ only sampled states which could contribute to the improvement of the current best level of violation and current best cost. Biased sampling consisted in drawing states from a distribution that is uniform over the planar states of the vehicle, *i.e.,* the (x, y)-coordinates, and that is Gaussian over the angular state, *i.e.,* the θ-coordinate. The mean of the angular state distribution coincided with the direction of the initial state of the automaton, and the variance of the distribution was made large enough so that the vehicle could find trajectories involving turns whenever necessary. This heuristic allowed the golf cart to safely travel at speeds of up to 10 km/h while in autonomous mode.

In our experiments, such as the one shown in the sequence in Figure 4-9, we only considered rules *(iii)-(a)* and *(iii)-(b)*, as the golf cart is too small to climb sidewalks and all the roads on which we were allowed to experiment were two-way streets. Traffic lane lines, sidewalks and fixed obstacles are detected using pre-generated maps of the campus roads, as seen in Figures 4-8b and 4-8c. Temporary and dynamic obstacles such as pedestrians and other cars are detected in real time using the laser range finders onboard the vehicle.

During operation, the vehicle tracked the current segment of the trajectory while running Algorithm 4.1 to plan for the next segment. If no unexpected obstacles appeared, then our algorithm was allowed to run up until the moment when the vehicle arrived to the last state of the current segment, which was the initial state of the tree maintained within the algorithm. At that point we passed the trajectory produced by our algorithm to the closed-loop tracking controller and then restarted our algorithm with the last state of the new trajectory being tracked playing the role of the initial state. Notice that here we exploited the *anytime property* of our algorithm, *i.e.,* its ability to find feasible trajectories and progressively refine them until it runs out of computation time.

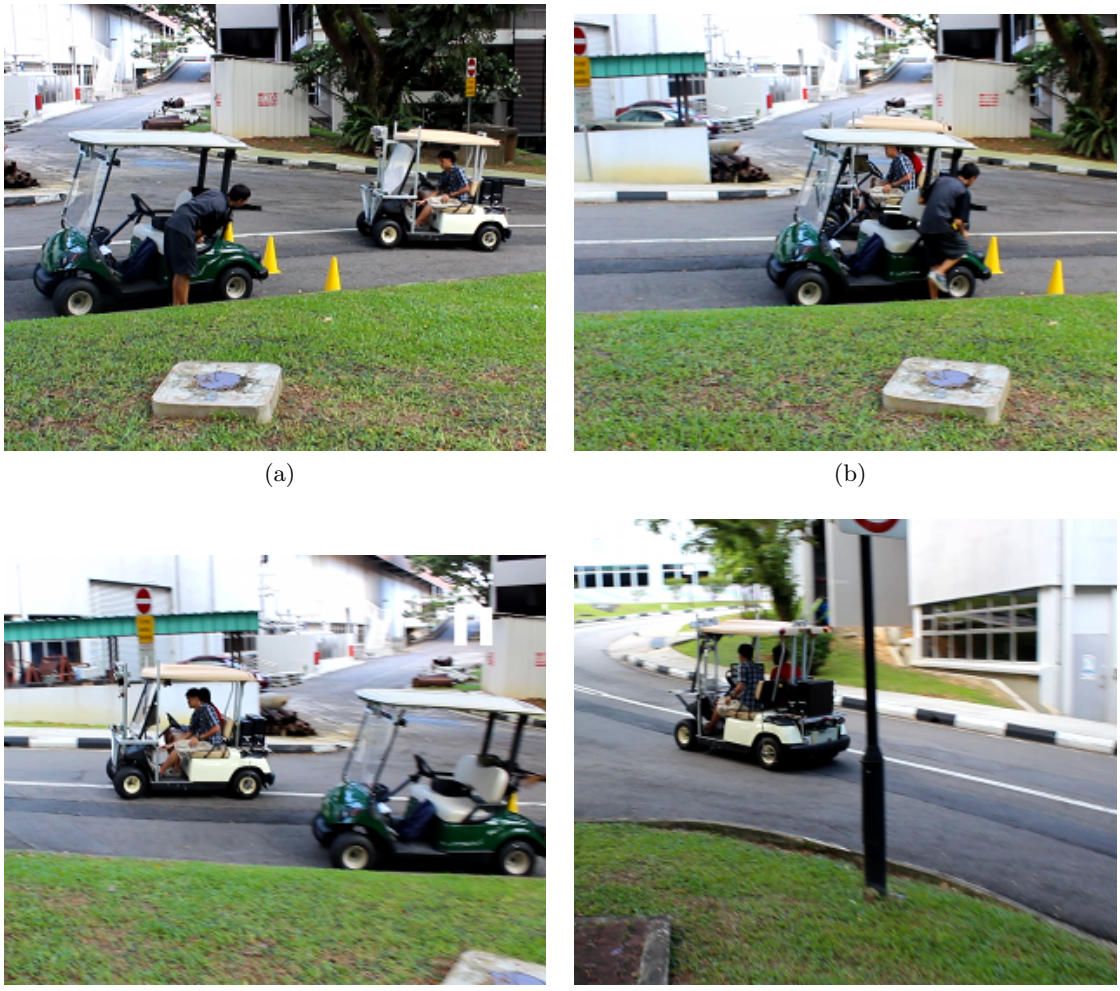
On the other hand, if while planning the next segment of the trajectory an unexpected obstacle materialized and rendered the current best trajectory infeasible, then we simply restarted our algorithm and allowed the vehicle to continue tracking the current segment of the trajectory. Also, if our algorithm had not found a feasible trajectory by the time the vehicle arrived to the last state of the current segment, then the vehicles's closed loop tracking controllers automatically applied the brakes. Finally, as expected, if for any reason the current segment of the trajectory being tracked became infeasible then we immediately ordered the tracking controllers to stop the car, regardless of whether or not the current best trajectory for the next segment was feasible or not.



(a) (b)

(c)

Figure 4-8: Our implementation of Algorithm 4.1 on an experimental testbed. Figure 4-8a shows the Yamaha golf cart which we used in our experiments and which is equipped with laser range-finders and cameras. Figure 4-8b shows a snapshot of the output of the Robot Operating System's console during an experiment. In this picture, red particles depict the estimate of the current position of the golf cart, which is obtained using laser range-finder data and adaptive Monte-Carlo localization on the map shown in Figure 4-8c. Additionally, the green line indicates the current segment of the trajectory being tracked, while the white lines represent the tree of states of automaton $\widetilde{\mathcal{P}}_{\mathcal{W}}$ maintained by our algorithm while planning for the next segment of the the trajectory.



(a) (b)

(c) (d)

Figure 4-9: A few snapshots of the experiments we carried at the NUS campus on May 2013. In this sequence we observe our autonomous golf cart returning to the left lane after maneuvering around a parked vehicle, in spite of the road curving to the right. Note that Singaporean driving conventions require drivers to maintain their left, and that the solution to this problem without road safety rules, *i.e.,* the time-optimal trajectory, would cut through the incoming lane to reach the goal region.

Chapter 5

Conclusion

In Chapter 1 we presented and motivated the problem of interest, and in Chapter 2 we provided background material on formal verification techniques for those readers in engineering who are not very familiar with the field.

In Chapter 3 we have studied the problem of synthesizing the least-violating path, *i.e.,* the most rewarding trace, of a fixed discrete robot model. Given a transition system, a set of infinite LTL formulae encoding the mission specifications, and a function associating formulae with the reward for their satisfaction, we have been able to find a path that satisfies the subset of the specifications with the largest overall reward in the case where there does not exist a path that satisfies all the specifications. In particular, by constructing a specialized automaton that captures the rewards of all the formulae through its weights, we have designed an algorithm for solving this problem, the computational complexity of which is substantially better to that of a brute-force solution. Additionally, we have proved the correctness and efficiency of the proposed algorithm, and we have demonstrated the utility of the algorithm by means of an illustrative example.

In Chapter 4 we have considered the problem of synthesizing a minimum-violation control strategy for a continuous-time continuous-space dynamical system that must travel from an initial state to a goal region while abiding by a set of safety specifications . We have focused on the case where it is impossible for the robot to reach the goal region without temporarily violating some of the specifications. By combining ideas from automata-based verification and techniques from sampling-based motion planning such as the RRT*algorithm, we have constructed a specialized automaton that captures the priorities of the different

specifications, which in turn has allowed us to design an algorithm for solving the problem. This algorithm has the almost sure asymptotic optimality guarantee, and it enjoys the same expected computational complexity per iteration of the RRT*algorithm. Finally, we have implemented this algorithm in both simulations and experiments.

References

[Asi50] Isaac Asimov, *I, Robot*, Gnome Press, 1950. 12

[BGK+11] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, CR Ramakrishnan, and Scott A Smolka, *Model Repair for Probabilistic Systems*, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2011, pp. 326–340. 13

[BK08] Christel Baier and Joost-Pieter Katoen, *Principles of Model Checking*, vol. 26202649, MIT Press Cambridge, 2008. 26, 58

[BKV10] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi, *Sampling-based Motion Planning with Temporal Goals*, Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE, 2010, pp. 2689–2696. 11

[CLR+09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al., *Introduction to Algorithms*, third ed., MIT press Cambridge, 2009. 67

[CRST08] Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Andrei Tchaltsev, *Diagnostic Information for Realizability*, Verification, Model Checking, and Abstract Interpretation, Springer, 2008, pp. 52–67. 13

[CY90] Costas Courcoubetis and Mihalis Yannakakis, *Markov Decision Processes and Regular Events*, Automata, Languages and Programming, Springer, 1990, pp. 336–349. 13

[DF11] Werner Damm and Bernd Finkbeiner, *Does It Pay to Extend the Perimeter of a World Model?*, FM 2011: Formal Methods, Springer, 2011, pp. 12–26. 13

[DSBR11] Xu Chu Ding, Stephen L Smith, Calin Belta, and Daniela Rus, *MDP Optimal Control under Temporal Logic Constraints*, Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on, IEEE, 2011, pp. 532–538. 11

[Dub57] Lester E Dubins, *On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents*, American Journal of mathematics (1957), 497–516. 72

[Fai11] Georgios E Fainekos, *Revising Temporal Logic Specifications for Motion Planning*, Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 40–45. 13

[GO01] Paul Gastin and Denis Oddoux, *Fast LTL to Büchi Automata Translation*, Computer Aided Verification, Springer, 2001, pp. 53–65. 32, 48, 51

[GP02] Elsa Gunter and Doron Peled, *Temporal Debugging for Concurrent Systems*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Springer, 2002, pp. 431–444. 32

[GPVW95] Rob Gerth, Doron Peled, Moshe Y Vardi, and Pierre Wolper, *Simple On-the-fly Automatic Verification of Linear Temporal Logic*, Proceedings of the Fifteenth IFIP WG6. 1 International Symposium on Protocol Specification, Testing and Verification, IFIP, 1995. 32, 59

[Hau14] Kris Hauser, *The Minimum Constraint Removal Problem with Three Robotics Applications*, The International Journal of Robotics Research **33** (2014), no. 1, 5–17. 13

[KB08] Marius Kloetzer and Calin Belta, *A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications*, Automatic Control, IEEE Transactions on **53** (2008), no. 1, 287–297. 11

[KF09] Sertac Karaman and Emilio Frazzoli, *Sampling-based Motion Planning with Deterministic μ-Calculus Specifications*, Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, IEEE, 2009, pp. 2222–2229. 11

[KF11] ———, *Sampling-based Algorithms for Optimal Motion Planning*, The International Journal of Robotics Research **30** (2011), no. 7, 846–894. 12, 63, 67, 71

[KF12] ———, *Sampling-based Algorithms for Optimal Motion Planning with Deterministic μ-Calculus Specifications*, American Control Conference (ACC), 2012, IEEE, 2012, pp. 735–742. 12, 67, 71

[KF13] ———, *Sampling-based Optimal Motion Planning for Non-holonomic Dynamical Systems*, Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE, 2013, pp. 5041–5047. 67, 68, 71

[KFS12] Kangjin Kim, Georgios E Fainekos, and Sriram Sankaranarayanan, *On the Revision Problem of Specification Automata*, Robotics and Automation (ICRA), 2012 IEEE International Conference on, IEEE, 2012, pp. 5171–5176. 13

[KGFP09] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas, *Temporal-logic-based Reactive Mission and Motion Planning*, Robotics, IEEE Transactions on **25** (2009), no. 6, 1370–1381. 11

[LaV06] Steven Michael LaValle, *Planning Algorithms*, Cambridge University Press, 2006. 73

[LWAB10] Morteza Lahijanian, Joseph Wasniewski, Sean B Andersson, and Calin Belta, *Motion Planning and Control from Temporal Logic Specifications with Probabilistic Satisfaction Guarantees*, Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE, 2010, pp. 3227–3232. 11

[RCCT+13] Luis I Reyes Castro, Pratik Chaudhari, Jana Tumova, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Incremental Sampling-based Algorithm for Minimum-violation Motion Planning*, Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on, IEEE, 2013, pp. 3217–3224. 55

[RKG11] Vasumathi Raman and Hadas Kress-Gazit, *Analyzing Unsynthesizable Specifications for High-level Robot Behavior using LTLMoP*, Computer Aided Verification, Springer, 2011, pp. 663–668. 12

[RKG12] ———, *Automated Feedback for Unachievable High-level Robot Behaviors*, Robotics and Automation (ICRA), 2012 IEEE International Conference on, IEEE, 2012, pp. 5156–5162. 12

[Sip06] Michael Sipser, *Introduction to the Theory of Computation*, Cengage Learning, 2006. 58

[STBR11] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus, *Optimal Path Planning for Surveillance with Temporal-logic Constraints*, The International Journal of Robotics Research **30** (2011), no. 14, 1695–1708. 11

[THK+13] Jana Tumova, Gavin C Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Least-violating Control Strategy Synthesis with Safety Rules*, Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, ACM, 2013, pp. 1–10. 63

[TP06] Paulo Tabuada and George J Pappas, *Linear Time Logic Control of Discrete-time Linear Systems*, Automatic Control, IEEE Transactions on **51** (2006), no. 12, 1862–1877. 11

[TRCK+13] Jana Tumova, Luis Reyes-Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Minimum-violation LTL Planning with Conflicting Specifications*, American Control Conference (ACC), 2013, IEEE, 2013, pp. 200–205. 35

[USDB12] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, and Calin Belta, *Robust Multi-robot Optimal Path Planning with Temporal Logic Constraints*, Robotics and Automation (ICRA), 2012 IEEE International Conference on, IEEE, 2012, pp. 4693–4698. 11

[WTM10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray, *Receding Horizon Control for Temporal Logic Specifications*, Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, ACM, 2010, pp. 101–110. 11, 12