Algorithms for Autonomous Urban Navigation with Formal Specifications

by

Pratik Chaudhari

S. M., Massachusetts Institute of Technology (2012)
B. Tech., Indian Institute of Technology Bombay (2010)

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

Engineer 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

Certified by .....
Sertac Karaman
Assistant Professor of Aeronautics and Astronautics
Thesis Committee Member

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

Algorithms for Autonomous Urban Navigation with Formal Specifications

by

Pratik Chaudhari

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2014, in partial fulfilment of the
requirements for the degree of
Engineer in Aeronautics and Astronautics

Abstract

This thesis addresses problems in planning and control of autonomous agents. The central theme of this work is that integration of "low-level control synthesis" and "high-level decision making" is essential to devise robust algorithms with provable guarantees on performance.

We pursue two main directions here. The first part considers planning and control algorithms that satisfy temporal specifications expressed using formal languages. We focus on task specifications that become feasible only if some of the specifications are violated and compute a control law that minimizes the *level of unsafety* of the system while guaranteeing that it still satisfies the task specification. Examples in this domain are motivated from an autonomous car navigating an urban landscape while following road safety rules such as "always travel in the left lane" and "do not change lanes frequently" or an electric vehicle in a mobility-on-demand scenario.

The second part of the thesis focuses on multi-agent control synthesis, where agents are modeled as dynamical systems and they interact with each other while sharing the same road infrastructure — all the while respecting the same road driving rules expressed as LTL specifications. We discuss algorithms that identify well-defined notions in the game theory literature such as Stackelberg equilibria and non-cooperative Nash equilibria under various information structures.

This work builds upon ideas from three different fields, viz., sampling-based motion-planning algorithms to construct efficient *concretizations* of general, continuous time dynamical systems, model checking for formal specifications that helps guarantee the safety of a system under all scenarios, and game theory to model the interaction between different agents trying to perform possibly conflicting tasks.

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

Thesis Committee Member: Sertac Karaman
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

I would like to express my gratitude to my advisor, Emilio Frazzoli. I have benefited immensely from his unwavering belief in his students along with his vision and amazing ability to identify important problems. He has been invaluable as a mentor and I consider myself extremely lucky to have been able to work closely with him. I would like to thank Sertac Karaman who is also my committee member on this thesis. Long conversations with him have had a huge impact in shaping my thinking as a young graduate student and he continues to be an enormous source of inspiration.

I have been lucky to be taught by some truly legendary teachers here at MIT. Their style and depth of knowledge is something I aspire to emulate. I dream of becoming a professor one day and I could not have wished for better role models. Being at LIDS has been instrumental in channeling my research interests. It has expanded my knowledge and broadened my horizons.

I have been lucky to work with a number of wonderful collaborators in the past few years. In particular, I would like to thank David Hsu at the National University of Singapore, Nok Wongpiromsarn at the Singapore-MIT Alliance for Research and Technology and Jana Tumova at the Royal Institute of Technology. Many a time, they have humored me and lent a patient ear to my half-baked ideas to help me evolve them into the material presented here. A special thanks goes to Jana who first introduced me to some of the most beautiful material in computer science that forms the crux of this thesis. She is a rare person and I have greatly enjoyed working with her. Luis Reyes-Castro and Valerio Varricchio with whom I worked closely in Singapore, have become fast friends.

I am fortunate to have several friends who have made this journey enjoyable and memorable. I would like to thank my lab mates at ARES and SMART for the amazing times we have shared. Here's to all the Dungeons and Dragons and the endless nights spent in the by-lanes of Singapore! I would also like to thank all the friends I have made at Cambridge and MIT, quite a few of them are old friends whom I have known for almost a decade. Graduate school so far away from home would not have been as enriching or fulfilling without them. I will cherish these memories.

Lastly, I would like to thank my parents and my brother. No words can do justice to all the sacrifices they have made for me. They have been a constant source of strength and encouragement, without which this thesis would have been impossible.

Contents

CHAPTER 1

Introduction

From avoiding traffic jams in busy cities to helping the disabled and elderly on their daily commute, autonomous vehicles promise to revolutionize transportation. The DARPA Grand Challenges [LHT+08, BIS09, TMD+06] were a gigantic leap towards this goal when they demonstrated that self-driving cars can not only traverse unknown terrain but can also do well in an urban scenario, complete with traffic rules. However, as autonomous cars begin their transition from these experimental projects into viable means of urban transportation, we are faced with a myriad of challenges, e.g., traffic management for autonomous cars, interaction of these agents with other drivers, sharing road infrastructure with pedestrians etc. — and most importantly, doing so while maintaining the same standards of safety that human drivers are subject to.

Figure 1.1: MIT's DARPA Urban Challenge car

To motivate the problems considered in this thesis, let us look at the DARPA Urban Challenge of 2007. It required competing teams to traverse an urban landscape in a completely autonomous manner. Doing so involved performing challenging tasks such as parking maneuvers, U-turns and negotiating cross-roads — all the while obeying traffic regulations. Fig. 1.1 shows Talos, MIT's entry to this competition, which is an LR3 Landrover outfitted with 5 cameras, 12 LIDARs, 16 radars and one 3D laser (Velodyne) sensor. The computing system on the other hand consists of a few hundred processes running in parallel on 40 processors. Team MIT was placed fourth in this competition and it completed an approximately 60 mile route in about 6 hours.

Fig. 1.2 shows a picture of the sensory and control data when Talos encounters Cornell University's car (Alice). Alice unfortunately has a bug in the software which causes it to stall in the middle of the road. *After waiting for some time* for Alice to make progress, Talos plans an alternate trajectory around it. However, as Talos gets closer to Alice, a *fail-safe* mechanism in Alice kicks in which says that it should not be very close to another vehicle and Alice moves forward. By this time however, both cars are committed to their respec-

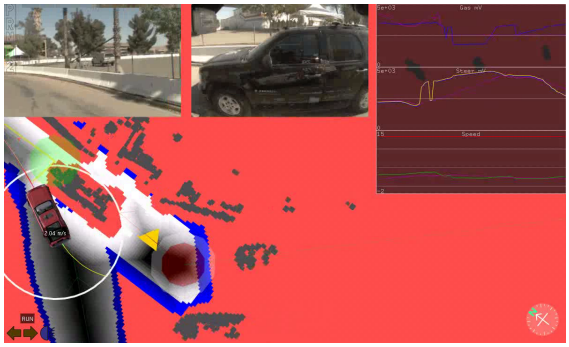


Figure 1.2: Cornell-MIT crash

tive *fail-safe* trajectories and they end up colliding. This incident — and a number of similar ones happened to almost all teams — provides the main motivation for this thesis. Specifically, we identify the following key points that resulted in the collision [FTO+08]:

1. An alternate trajectory that was classified as "safe" by Talos transitioned from the left-lane into the right-lane, which is a potentially dangerous maneuver, in other words, high-level routines that reason about logical safety of a trajectory were completely decoupled from motion-planning.
2. Alice had a number of *conflicting fail-safe* rules whose priorities were not clearly defined with respect to each other. In particular, even though, each of them guaranteed reasonable behavior individually, together, they were not able to guarantee the vehicle's safety.
3. Talos did not correctly judge the *intent* of the other vehicle, i.e., sensory information and perception algorithms were insufficient to classify Alice as "another car that is likely to move".

Let us elaborate further on these points. In the next section, we will identify some fundamental challenges in traditional engineering methodologies for such systems.

1.1 Motivation and Contributions

1.1.1 *Integrated Decision Making and Control*

One of the main challenges of reasoning about "high-level decision making" and implementing it using "low-level controllers" is that the computational models at these two levels are incompatible. For example, logical rules like "do not go into the left lane" that can be easily verified for discrete state-trajectories are hard to verify (or even formalize) for dynamical systems with differential constraints. This has led to a hierarchy of design methodologies wherein, tools from computer science and control theory respectively, are used *independently* of each other. While this is both reasonable and effective, one of the main challenges of this approach is that evaluating the performance or proving the safety of the closed-loop system with respect to its specifications, is difficult. In fact, if the abstracted finite transition system is too coarse, there is no guarantee that the optimal controller will be found even if one exists [WTM10], i.e., these methods are not complete and cannot be applied to, for example, dynamically changing environments.

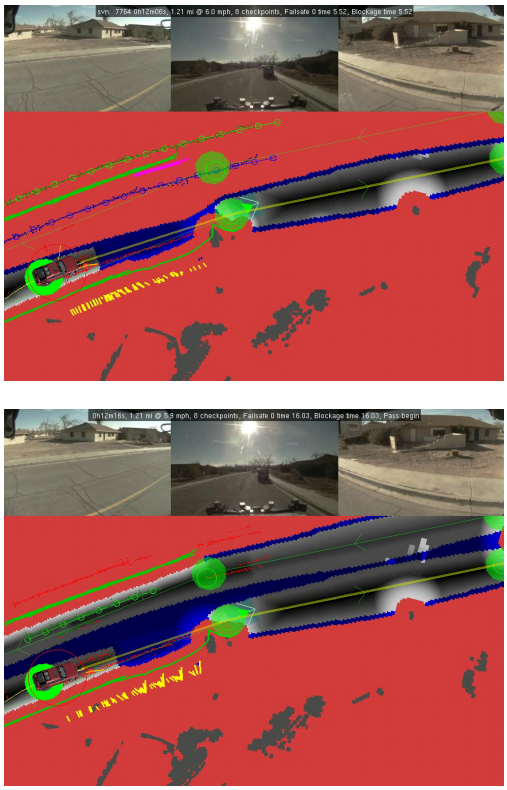


Figure 1.3: The first picture shows the left lane as a "virtual obstacle". If no progress is made for 15 secs., as shown in the second picture, the software clears the left-lane and allows the car to overtake a parked car.

In this work, we propose an integrated architecture for decision making and control. We

construct on line sequences of *concretizations* of general, dynamical systems known as Kripke structures that also maintain low-level control laws. These Kripke structures are finite models of the continuous dynamical system and algorithms from model checking in the computer science literature can be readily used to evaluate temporal properties such as safety, fairness, liveness and their combinations. Also, we are interested in refining these Kripke structures to obtain asymptotic-optimality of the resulting algorithms. In order to differentiate this from *finite abstractions* that are popular in literature, we call these, *concretizations of dynamical systems*.

This thesis proposes a way to construct concretizations of dynamical systems with differential constraints. We use ideas from sampling-based motion-planning algorithms such as Probabilistic Road Maps (PRMs) and Rapidly-exploring Random Trees (RRTs) to create iteratively refined Kripke structures. In particular, recent versions of these algorithms such as PRM* and RRT* [KF11b] help us ensure two important characteristics that differentiate this work from contemporary results, (i) probabilistic completeness, i.e., the algorithm finds a Kripke structure that satisfies the specifications if one exists with high probability, and (ii) probabilistic asymptotic-optimality, i.e., almost surely, the algorithm returns a *continuous* trajectory that not only satisfies all specifications but also minimizes a given cost function while doing so.

1.1.2 *Minimum-violation Planning*

Logical specifications can be succinctly expressed using formal languages such as Linear Temporal Logic (LTL) [KB06, KGFP07], Computational Tree Logic (CTL, CTL*) [LWAB10], modal μ-calculus [KF12a] etc. The general problem of finding optimal controllers that satisfy these specifications has been studied in a number of recent works such as [DSBR11, TP06, STBR11, USDB12]. Given these specifications, if the underlying task is feasible, using methods discussed in the previous section, we can compute an asymptotically-optimal controller for a given dynamical system that satisfies them. On the other hand, in many situations, for example when a single-lane road is blocked, one might have to violate some specifications, e.g., "do not go into the left lane" in order to satisfy the task, e.g., "reach a goal region". In fact, (cf. Fig. 1.3), MIT's DARPA Grand Challenge team had a series of rules in their code to demarcate the left-lane as a virtual "obstacle"; if no progress was made for 15 secs. due to a blocked right lane, the algorithm would selectively remove these "obstacles" until a reasonable motion-plan was found. A consequence of this is that a number of such conflicting rules and pre-specified parameters exist in the framework which makes it very hard to design and debug. It is also hard to guarantee the performance and safety of the overall system.

This thesis discusses a principled approach to tackle such problems. In particular, in our work, we quantify the *level of unsafety* of a trajectory that breaks a subset of specifications in order to satisfy a task. These specifications are expressed using the finite fragment of LTL, e.g., FLTL$_{-X}$, and are converted into a weighted finite automaton such that traces of the Kripke structure on this automaton have a weight that is exactly equal to the level of unsafety of the corresponding trajectory. Using ideas from local model-checking and sampling-based motion-planning, we can then construct an algorithm that identifies the trace in the Kripke structure (and effectively, the continuous trajectory) that minimizes this level of unsafety.

Our work here can be seen in context of recent literature on control synthesis for an unsatisfiable set of specifications. For example, works such as [CRST08] reveal unsatisfiable fragments of given specification, while others such as [Fai11, KFS12, KF12b] try to synthesize controllers with minimal changes to the original specifications, e.g., by modifying the transition system. Two

lines of work that are closest to ideas proposed in this thesis are — [Hau12] which focuses on finding the least set of constraints, the removal of which results in satisfaction of specifications and [DF11, CY98] which quantify the *least-violating controller* according to some proposed metric for a given transition system. Let us note that similar approaches exist for probabilistic transition systems, e.g., [BEGL99, BGK+11].

1.1.3 *Multi-agent Systems*

On the other hand, consider problems such as merging into lanes on freeways, negotiating cross-road junctions, round-abouts etc. In order to enable autonomous vehicles to reason about these situations effectively, we have to develop algorithms for interaction of these agents with the external environment. This thesis takes a look at game theoretic approaches to these problems in conjunction with concepts from model checking and sampling-based algorithms.

Differential games [Isa99, BO95] are popular models for problems such as multi-agent collision avoidance [MBT05], path planning in adversarial environments [VBJP10] and pursuit-evasion problems [GLL+99]. However, analytical solutions for differential games exist for only specific problems, e.g., the "lady in the lake" game, or Linear Quadratic Regulator (LQR) games [BO95]. For problems involving more complex dynamics or other kinds of cost functions, solutions are hard to characterize in closed form. Numerical techniques are based on converting the problem to a finite dimensional optimization problem [Rai01] or solving the corresponding partial differential equations using shooting methods [BPG93, BCD97, BFS99, Sou99], level set methods [Set96], viability theory [ABSP11, CQSP99] etc.

Similarly, there are few rigorous results for game theoretic controller synthesis for multi-robot planning, e.g., [LH98]. However, a number of works solve related problems, e.g., [ZM13] solves optimal sensor deployment while [ASF09] considers vehicle routing problems. These papers mainly reply on centralized planning [SL02, XA08], decoupled synthesis [KZ86, SLL02] or ad-hoc priority-based planning, e.g., [Buc89, ELP87].

This thesis proposes a formulation to compute equilibria for two-player differential games where players try to accomplish a task specification while satisfying safety rules expressed using temporal logic. We formulate the interaction between an autonomous agent and its environment as a non-zero sum differential game; both the robot and the environment minimize the *level of unsafety* of a trajectory with respect to safety rules expressed using LTL formulas. We describe an algorithm to compute the open-loop Stackelberg equilibrium (OLS) of this game. We also consider generic motion-planning tasks for multi-robot systems and devise an algorithm that converges to the non-cooperative Nash equilibrium of the differential game in the limit. Throughout, we employ techniques from sampling-based algorithm to construct concretizations of dynamical systems (cf. Sec. 1.1.1) and model checking techniques on these Kripke structures.

1.2 Organization

This document is organized as follows. In Chap. 1, we develop some preliminary concepts that are used in the remainder of the thesis. We construct durational Kripke structures for efficient model checking of dynamical systems and provide an overview of Linear Temporal Logic (LTL) along with automata based approaches for model checking LTL. This chapter also introduces process algebras and develops various preliminary concepts for differential games.

Chap. 3 formulates the problem of "minimum-violation motion-planning" and proposes an algorithm that minimizes the level of unsafety of a trajectory of the dynamical system with respect to LTL specifications. It also provides a number of results from computational experiments along with an implementation on an experimental full-size autonomous platform. Chap. 4 develops these concepts further, but takes a different route. It considers specifications which can be expressed using simple languages such as process algebras and finds a number of applications related to mobility-on-demand that can be solved using these.

Chap. 5 delves into multi-agent motion-planning problems. It first formulates a two-player, non-zero sum differential game between the robot and external agents and proposes an algorithm that converges to the Stackelberg equilibrium asymptotically. The central theme of this chapter is that we incorporate "minimum-violation" cost functions into game theoretic motion-planning. The later part of his chapter uses ideas from sampling-based algorithms to look at n-player non-cooperative Nash equilibria for generic motion-planning tasks.

Finally, we take a holistic view of urban motion-planning using formal specifications and comment on future directions for research.

1.3 Published Results

Parts of this thesis have been published in the following research articles —

1. Luis I. Reyes Castro, Pratik Chaudhari, Jana Tumova, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Incremental Sampling-based Algorithm for Minimum-violation Motion Planning. In *Proc. of IEEE International Conference on Decision and Control (CDC)*, 2013.

2. Valerio Varricchio, Pratik Chaudhari, and Emilio Frazzoli. Sampling-based Algorithms for Optimal Motion Planning using Process Algebra Specifications. In *Proc. of IEEE Conf. on Robotics and Automation*, 2014.

3. Minghui Zhu, Michael Otte, Pratik Chaudhari, and Emilio Frazzoli. Game theoretic controller synthesis for multi-robot motion planning Part I : Trajectory based algorithms. In *Proc. of IEEE Conf. on Robotics and Automation*, 2014.

4. Pratik Chaudhari, Tichakorn Wongpiromsarn, and Emilio Frazzoli. Incremental Synthesis of Minimum-Violation Control Strategies for Robots Interacting with External Agents. In *Proc. of American Control Conference*, 2014.

CHAPTER 2

Background Material

This chapter introduces some preliminary material on dynamical systems with differential constraints and Kripke structures which are efficient data-structures to talk about temporal properties of trajectories of dynamical systems. It also introduces formal languages such as Linear Temporal Logic and Process Algebras and model checking algorithms for these languages that are used in this work. Lastly, we provide some discussion on game theoretic notions of equilibria such as the Nash equilibrium and Stackelberg equilibrium under various information structures.

Notation: For a finite set S, denote the cardinality and the powerset of S as $|S|$ and 2^S, respectively. Let S^* be the set of all finite sequences of elements of S and for $u, v \in S^*$, denote the concatenation of u and v as $u \cdot v$. Let S^ω denote the set of all infinite strings of elements of S and given some $u \in S^*$, we denote by $u^\omega \in S^\omega$, its infinite concatenation with itself.

2.1 Durational Kripke Structures for Dynamical Systems

Consider a dynamical system given by

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_{init}, \tag{2.1}$$

where $x(t) \in \mathcal{X} \subset \mathbb{R}^d$ and $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ with $\mathcal{X}, \mathcal{U}$ being compact sets and x_{init} is the initial state. Trajectories of states and control are maps $x : [0, T] \to \mathcal{X}$ and $u : [0, T] \to \mathcal{U}$ respectively for some $T \in \mathbb{R}_{\geq 0}$. We will tacitly assume in all this work that the above equation has a unique solution for every x_{init}; in order to do so, we assume that $f(\cdot, \cdot)$ is Lipschitz continuous in both its arguments and u is Lebesgue measurable.

Let Π be a finite set of "atomic propositions" and $\mathcal{L}_c : \mathcal{X} \to 2^\Pi$ map each state to atomic propositions that are true at that state. We can reason about the atomic propositions satisfied by a trajectory of the dynamical system as follows. For a trajectory x, let $D(x) = \{t \mid \mathcal{L}_c(x(t) \neq \lim_{s \to t^-} \mathcal{L}_c(x(s)), 0 < t \leq T\}$ be the set of all discontinuities of $\mathcal{L}_c(x(\cdot))$. In this work, we assume that $D(x)$ is finite for all trajectories. This is true for various different $\mathcal{L}_c$s considered here, one can show that the set of trajectories that do not satisfy this condition has measure zero. A trajectory $x : [0, T] \to \mathcal{X}$ with $D(x) = \{t_1, \ldots, t_n\}$ produces a *finite timed word*

$$w_t = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_n, d_n), \quad \text{where}$$

(i) $\ell_k = \mathcal{L}_c(x(t_k))$ for all $0 \leq k \leq n$ and $t_0 = 0$, $d_k = t_{k+1} - t_k$ and

(ii) $\ell_n = \mathcal{L}_c(x(t_n))$ and $d_n = T - t_n$.

The *finite word* produced by this trajectory is

$$w(x) = \ell_0, \ell_1, \ldots, \ell_n.$$

We are now ready to define a special type of Kripke structure that allows us to reason about all timed words produced by a dynamical system.

Definition 2.1 (Durational Kripke structure). A durational Kripke structure is defined by a tuple $K = (S, s_{init}, R, \Pi, L, \Delta)$, where S is a finite set of states, $s_{init} \in S$ is the initial state, $R \subseteq S \times S$ is a deterministic transition relation, Π is the set of atomic propositions, $L : S \to 2^{\Pi}$ is a state labeling function, and $\Delta : R \to \mathbb{R}_{\geq 0}$ is a function that assigns a time duration to each transition.

A *trace* of K is a finite sequence of states $\rho = s_0, s_1, \ldots, s_n$ such that $s_0 = s_{init}$ and $(s_k, s_{k+1}) \in R$ for all $0 \leq k < n$. Corresponding to it is the finite timed word that it produces, i.e., $\rho_t = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_n, d_n)$ where $(\ell_k, d_k) = (L(s_k), \Delta(s_k, s_{k+1}))$ for all $0 \leq k < n$ and $(\ell_n, d_n) = (L(s_n), 0)$. The word produced by this trace is $w(\rho) = \ell_0, \ell_1, \ldots, \ell_n$. Note that as defined until now, a word may have multiple consecutive ℓ_ks that are the same, e.g., the trajectory spends a lot of time in each labeled region. Such repeated atomic propositions are not useful to us for reasoning about temporal properties of the trajectories and hence we remove them using the destutter operator as shown below.

Given $w(\rho)$, let $I = \{i_0, i_1, \ldots, i_k\}$ be the unique set of indices such that $i_0 = 0$ and

$$\ell_{i_j} = \ell_{i_j+1} = \ldots = \ell_{i_{j+1}-1} \neq \ell_{i_{j+1}} \quad \text{for all} \quad 0 \leq j \leq k-1$$

and similarly $\ell_k = \ell_{k+1} = \ldots = \ell_n$. Define the "destutter" operator to remove these repeated consecutive elements of a word as

$$\text{destutter}(w(\rho)) = \ell_{i_0}, \ell_{i_1}, \ldots, \ell_{i_{k-1}}, \ell_{i_k}.$$

For convenience we also denote the duration of a trace by $\langle \rho \rangle$, i.e., $\langle \rho \rangle = \sum_{i=0}^{n} d_i$. Let us now define a Kripke structure that explicitly encodes the properties of the dynamical system under consideration.

Definition 2.2 (Trace-inclusive Kripke structure). A Kripke structure $K = (S, s_{init}, R, \Pi, L, \Delta)$ is called trace-inclusive with respect to the dynamical system in Eqn. (2.1) if

- $S \subset \mathcal{X}$, $s_{init} = x_{init}$,
- for all $(s_1, s_2) \in R$ there exists a trajectory $x : [0, T] \to \mathcal{X}$ such that $x(0) = s_1$ and $x(T) = s_2$ with $T = \Delta(s_1, s_2)$ and $|D(x)| \leq 1$ i.e., $\mathcal{L}_c(x(\cdot))$ changes its value at most once while transitioning from s_1 to s_2.

The following lemma now easily follows, it relates trajectories of the dynamical system to traces of a trace-inclusive Kripke structure.

Lemma 2.3. *For any trace ρ of a trace-inclusive Kripke structure, there exists a trajectory of the dynamical system, say $x : [0, T] \to \mathcal{X}$ such that*

$$\textit{destutter}(w(\rho)) = w(x).$$

2.2 Linear Temporal Logic (LTL)

We start by defining a finite automaton. For a more thorough and leisurely introduction to the theory of computation and temporal logics, one may refer to a number of excellent texts on the subject, e.g., [Sip12, BK+08].

Definition 2.4 (NFA). A non-deterministic finite automaton (NFA) is a tuple $A = (Q, q_{init}, \Sigma, \delta, F)$ where Q is a finite set of states, $q_{init} \in Q$ is the initial state, Σ is a finite set called the input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation and $F \subseteq Q$ is a set of accepting states.

The semantics of finite automata are defined over Σ^*, in particular traces of Kripke structures. A run ρ of a finite automaton for the input $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$ is the sequence $q_0, q_1, \ldots, q_n$ such that $q_0 = q_{init}$, $(q_{k-1}, \sigma_k, q_k) \in \delta$ for all $k \leq n$. We say that the input σ is accepted if $q_n \in F$ and rejected otherwise. The set of all finite strings accepted by an NFA is called its language.

If the transition relation is *deterministic*, i.e., each state in the automaton can transition to only one other state, we say that A is a deterministic finite automaton (DFA). It is surprising that we do not gain any expressive power by non-determinism in finite automata, in other words, the set of all languages that can be expressed using NFAs — also called regular languages — is exactly the same as the set of all languages expressed using DFAs. Consequently, any NFA can be converted into a DFA (possibly, with exponentially many states) having the same language.

In the context of this work, NFAs will have an input alphabet of 2-tuples, consecutive elements of the word produced by a trace of the Kripke structure will form the input alphabet. Thus, if $\sigma = (\ell_k, \ell_{k+1})$ for some k, we have $\Sigma = 2^{\Pi} \times 2^{\Pi}$. Using this, we can talk about NFAs which "detect the change" in the logical properties of a trajectory. As a technicality, we will also require NFAs to be *non-blocking*, i.e., for all states $q \in Q$, and $\sigma \in \Sigma$, there exists a transition $(q, \sigma, q') \in \delta$ for some $q' \in Q$. Note that any blocking automaton can be trivially made non-blocking by adding transitions to a new state $q_{new} \notin F$.

Definition 2.5 (ω-automaton). An ω-automaton is a tuple $A = (Q, q_{init}, \Sigma, \delta, Acc)$ where the elements Q, q_{init}, Σ and δ are the same as an NFA and Acc is an "acceptance condition".

The semantics of ω-automata are defined similarly as NFAs, except that they are over infinite input words, i.e., elments of Σ^ω. We thus need to re-define what it means for an ω-automaton to "accept" a word, which brings us to the acceptance condition. Contrary to NFAs and DFAs, a number of different accepting conditions give rise to ω-automata of varying power. For a Büchi automaton (BA), Acc is "for a given set $F \subseteq Q$, a run ρ is accepted if it intersects F infinitely many times". For a generalized Büchi automaton (GBA), Acc is "given $\{F_1, F_2 \ldots, F_m\} \subseteq Q^m$, a run ρ is accepting if it intersects each F_i infinitely many times". In addition to this, there are a number of other ω-automata with different accepting conditions such as Muller, Rabin, Streett and parity automata. We note here that in the world of ω-regular languages, i.e., languages of ω-automata, non-determinism is powerful. Non-deterministic BAs, even deterministic Rabin and Muller automata can all be shown to express the whole set of ω-regular languages; on the other hand deterministic BAs only express a strict subset of ω-regular languages.

We will not be using ω-automata in this work. We however motivate model-checking for Linear Temporal Logic, which is a strict subset of ω-regular languages using these concepts.

2.2.1 *Syntax and Semantics of LTL*

Temporal properties can be expressed using ω-automata as we saw earlier. However, there are more concise representations using languages that are also easy to read for humans. Propositional LTL is one such language — it appends propositional logic with temporal operators. LTL is defined inductively and the syntax can be given by the following grammar:

$$\phi ::= \mathsf{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\,\phi \mid \phi_1 \,\mathsf{U}\, \phi_2$$

where $a \in \Pi$. Using these we can define temporal modalities like F (eventually) and G (always) as follows:

$$\mathsf{F}\,\phi = \mathsf{true}\,\mathsf{U}\,\phi \qquad \mathsf{G}\,\phi = \neg(\,\mathsf{F}\,\neg\phi)$$

i.e., if it never happens that $\neg\phi$ is true eventually, then ϕ always holds, equivalently ϕ holds from now on forever. The semantics of LTL can be defined as follows. Given $\sigma = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$, we denote the (infinite) substring from element j as $\sigma_{[j:]} = \sigma_j\sigma_{j+1}\ldots$. Then

$$\begin{aligned}
&\sigma \models \mathsf{true} \\
&\sigma \models a && \text{iff} \quad \sigma_0 \models a \\
&\sigma \models \phi_1 \wedge \phi_2 && \text{iff} \quad \sigma \models \phi_1 \text{and } \sigma \models \phi_2 \\
&\sigma \models \neg\phi && \text{iff} \quad \sigma_0 \not\models \phi \\
&\sigma \models \mathsf{X}\,\phi && \text{iff} \quad \sigma_{[1:]} \models \phi \\
&\sigma \models \phi_1 \,\mathsf{U}\, \phi_2 && \text{iff} \quad \exists\, j \geq 0,\ \sigma_{[j:]} \models \phi_2 \text{and } \sigma_{[i:]} \models \phi_1 \quad \forall\, 0 \leq i \leq j
\end{aligned}$$

LTL can be used to succinctly express various temporal properties, for example,

- liveness: $\mathsf{G}\ \mathsf{F}\,\phi$ (ϕ holds infinitely often),
- safety: $\mathsf{G}\,\neg\phi$ (avoid ϕ),
- reactivity: $\mathsf{G}\ \mathsf{F}\,\phi_1 \Rightarrow\ \mathsf{G}\ \mathsf{F}\,\phi_2$ (if ϕ_1 holds infinitely often, so must ϕ_2) etc.

In order to verify if traces of a Kripke structure satisfy the temporal properties denoted by a given LTL expression ϕ, we use automata-based model checking, i.e., the LTL expression is first converted into an equivalent ω-regular automaton. Using algorithms discussed in Sec. 2.2.2, we can then use simple search algorithms to verify ϕ. Here, we briefly discuss the conversion of LTL to Büchi automata [VW94]. Assume that the formula is in *negated normal form*. We construct the closure of ϕ, denoted as $\mathrm{cl}(\phi)$, which is the set of all sub-formulas and their negations. Not surprisingly, this forms the alphabet the Büchi. The states of the Büchi are the maximal subsets of $\mathrm{cl}(\phi)$, i.e., largest sets of formulas that are consistent. Roughly, for a state $q \subset M \subset \mathrm{cl}(\phi)$ of the GBA, there exists an accepting run of the automaton that satisfies every formula in M and does not satisfy any formula in $\mathrm{cl}(\phi) \setminus M$. The transition function is easy to construct, we just don't have to violate any temporal or Boolean properties of LTL while connecting two states. The accepting states are F_ψ for every sub-formula of the form $\psi = \phi_1 \,\mathsf{U}\, \phi_2$ of ϕ. We ensure that for a run $B_0B_1\ldots$, if $\psi \in B_0$, we have that $\phi_2 \in B_j$ for some $j \geq 0$ and $\phi_1 \in B_i$ for all $i < j$. The acceptance condition is thus a GBA condition. This automaton can be easily de-generalized into an NBA using standard algorithms [BK+08].

As an interesting aside, let us note that since deterministic Büchi automata cannot exhaust ω-regular properties, we cannot use automata minimizing algorithms to convert NBAs to DBAs.

This is particularly important because the size of NBA constructed above is exponential in the size of ϕ and the size of model checking is proportional to the size of the NBA.

Finite Linear Temporal Logic (FLTL)

We focus here on temporal properties expressed using the finite fragment of LTL. Finite LTL [MP95, GP02] has the same syntax as LTL with the semantics defined over finite runs of the underlying transition system. FLTL is defined using two kinds of X operators, the strong next operator, denoted using the usual X says that a property ϕ is true in the next time instant. Of course, since FLTL deals with finite strings, there is no way this property can be true at the last instant. Hence, we define a weak next operator, denoted as $\overline{\mathsf{X}}$ which says that ϕ is true in the next instant if one exists. This also gives rise to a corresponding weak until operator.

Let us now present the construction of NFAs from FLTL specifications. Please refer to [GP02] for a more elaborate discussion. Because there does not exist an equivalent "generalized NFA", we use a slightly modified version of the LTL conversion algorithm as shown here [GPVW95]. For a negated normal formula ϕ, construct a graph where each vertex v holds the following data:

- id: a unique identifier,
- `incoming`: the set of edges that point to v,
- `new`: sub-formulas which hold below v but are not yet processed,
- `old`: sub-formulas which hold below v that are processed,
- `next`: sub-formulas that have to hold for all successors of v,
- `strong`: a flag to signal that v cannot be the last state in a sequence.

The algorithm then constructs a graph G starting from a vertex v, which has `incoming` from a dummy vertex `init`, with `new` $= \{\phi\}$ and `old`, `next` $= \emptyset$. Initialize an empty list called `completed_nodes`. Recursively, for a node v not yet in `completed_nodes`, move a sub-formula η from `new` to `old`. Split v into left and right copies according to Tab. 2.1. `old`, `incoming` retain their values. Roughly, the `strong` indicates that the formulas in `next` are to be upgraded from weak next to strong next.

Formula	left `new`	left `next`	strong	right `new`	right `next`
$\mu \mathsf{U} \nu$	μ	$\mu \mathsf{U} \nu$	1	ν	$\emptyset$
$\mu \overline{\mathsf{U}} \nu$	ν	$\mu \overline{\mathsf{U}} \nu$		μ, ν	$\emptyset$
$\mu \vee \nu$	μ	$\emptyset$		ν	$\emptyset$
$\mu \wedge \nu$	μ, ν	$\emptyset$		-	-
$\overline{\mathsf{X}} \mu$	$\emptyset$	μ		-	-
$\mathsf{X} \mu$	$\emptyset$	μ	1	-	-
$p, \neg p$	$\emptyset$	$\emptyset$		-	-

Figure 2.1: Table for branching in conversion of FLTL to NFA

If there are no more sub-formulas in `new` of v, compare v to `completed_nodes` to check if there is some node u that has the same `old` and `next` and add `incoming` of v to `incoming` of u, i.e., there are multiple ways of arriving at the same u. Else, add v to `completed_nodes` and start a new node's recursion with an incoming edge from v and its `new` is the `next` of v.

The states of the NFA are then all the nodes in the list of completed nodes. The initial nodes are the ones with incoming edges from `init`. The transition relation is easily obtained from `incoming`.

To get the accept states, note that a state is an accept state iff for each sub-formula of ϕ of the form $\mu \,\mathsf{U}\, \nu$, either `old` contains ν or it does not contain $\mu \,\mathsf{U}\, \nu$ and its strong bit is set to false.

As an important deviation from conventional FLTL, in our work, we would like to reason about the properties of a trajectory of a dynamical system, i.e., $w(x)$. Thus, we do not use formulas which involve the X operator and define $\text{FLTL}_{-\mathsf{X}}$ to be FLTL without the X operator.

2.2.2 *Model Checking LTL and its Variants*

To motivate automata based model checking for LTL, let us first consider verification of safety properties which are regular languages. Let us consider the complement of this set, i.e., all the bad finite strings. It is easy to see that we need only work with the "minimal" such set, i.e., the smallest set of bad prefixes. Since this by assumption is a regular language, we can find an automaton $\overline{A}$ that accepts all minimal bad prefixes. Given a Kripke structure K, the problem is then to find whether there exists a trace of K that is accepted by $\overline{A}$. This can be posed as simple reachability problem on the "product automaton" defined as follows:

Definition 2.6 (Product automaton). Given a Kripke structure $K = (S, s_{init}, R, \Pi, L, \Delta)$ and a non-blocking finite automaton A, the product automaton is a tuple $P = K \otimes A = (Q_P, q_{init,P}, 2^{\Pi}, \delta_P, F_P)$ where

- $Q_P = S \times Q$,
- $q_{init,P} = (s_{init}, q_{init})$ is the initial state,
- $\delta_P \subseteq Q_P \times Q_P$ is non-deterministic transition relation such that $(\langle s, q\rangle, \langle s', q'\rangle) \in \delta_P$ iff $(s, s') \in R$ and $(q, L(s'), q') \in \delta$,
- $F_p = S \times F$ is the set of accepting states.

Checking if K is safe with respect to $\overline{A}$ is then simply an analysis of where an accept state of P is reachable from its initial state, i.e., if the language of P is non-empty, there exists a trace of Kripke structure which satisfies the property encoded by $\overline{A}$ and is thus not safe. Moreover, this can be done efficiently, e.g., using depth-first search, and also provides a counter-example, i.e., the trace of K that fails. Let us note that the complexity of this algorithm is $\mathcal{O}(nm)$ where n, m are the number of states in the Kripke structure and the automaton, respectively.

Model checking ω-regular properties works in a very similar manner. We first construct the product automaton using K and say, an NBA $\overline{A}$ and run a depth-first search. Since NBAs are finite automata that can accept (or reject) infinite words, we however have to detect cycles while running the search and this gives rise to "nested depth-first search" [BK+08].

Let us describe the algorithm for formulas of the form $\mathsf{G}\,\mathsf{F}\,\phi$. Conceptually, if there exists a reachable cycle in the Kripke structure containing $\neg\phi$, this formula is not true. One way to check for this is to compute the strongly connected components of K and check for cycles with $\neg\phi$. We can however do this differently. First find all states that satisfy $\neg\phi$ (since they are possible candidates of cycles), then spawn a depth-first search from each of them to check for cycles. In fact, we can also interleave the two steps by using DFS for both of them, i.e., an *outer* DFS identifies all $\neg\phi$ states and an *inner* nested DFS tries to find a cycle that starts and ends at that state such that it does not contain any state of the outer DFS. The computational complexity of this algorithm is $\mathcal{O}(nm + n\,|\phi|)$ where n, m are the number of states and transitions in K respectively, and $|\phi|$ is the size of formula ϕ. Moreover, it can be shown that checking any ω-regular property on a

Kripke structure can be reduced to checking if some persistence property holds on the product automaton.

2.3 Process Algebras

Process Algebras (PA) are a mathematical framework that was developed to formalize and understand the behavior of concurrent systems, e.g., design requirements for software/ hardware. PA are formalized as *behaviors*, which represent the output of the system; these are the results of *actions* that happen at specific time instants when their preconditions are true. In the context of model checking, process algebra can be used to describe a set of equivalence relations enable to formally equate a given implementation of the system to its design specification or establish non-trivial properties of systems in an elegant fashion. A detailed description of the language and such applications can be found in [Fok00].

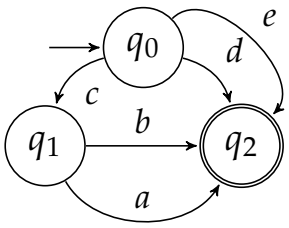


Figure 2.2: An example process graph generated for the expression $c \cdot (a + b) + (d + e)$. The different traces are, $c, c.a, c.b, d, e$. The last four out of these are accepting traces.

Definition 2.7 (Basic Process Algebra (BPA)). Let A be the finite set of atomic actions. The set $\mathbb{T}$ of BPA terms over A is defined inductively as:

- for all actions $p \in A$, $p \in \mathbb{T}$;
- for all $p, p' \in \mathbb{T}$, $p + p'$, $p \cdot p'$ and $p \parallel p'$ belong to $\mathbb{T}$.

The compositions $p + p', p \cdot p', p \parallel p'$ are called alternative, sequential and parallel compositions respectively. The semantics of this definition are as follows: if $p \in A$, the process $p \in \mathbb{T}$ executes p and terminates. The process $p + p'$ executes the behavior of either p or p'. $p \cdot p'$ first executes p and then executes the behavior of p' after p terminates. The parallel composition is used for concurrent processes and executes the behavior of p, p' simultaneously and terminates after both terminate. As a convention, the sequential operator has priority over the alternative operator.

Algorithm 2.1: `parse`$(p, q_0, q_a, (Q, \delta))$

```
1  *, p1, p2 ← split(p);
2  switch * do
3      case '+'
4          parse(p1, q0, qa, (Q, δ)) ;
5          parse(p2, q0, qa, (Q, δ)) ;
6      case '·'
7          qnew ← ∅;
8          Q ← Q ∪ qnew;
9          parse(p1, q0, qnew, (Q, δ));
10         parse(p2, qnew, qa, (Q, δ));
11     otherwise δ ← δ ∪ (q0, p, qa) ;
```

A sequence of actions, also known as a *trace* is denoted by ρ. Given a process $p \in \mathbb{T}$, we can construct a *process graph* that maps the process to a set of traces that describe its various possible behaviors.

Definition 2.8 (Process graph). A process graph for $p \in \mathbb{T}$ is a labeled transition system $G(p) = (Q, q_{init}, A, \pi, \delta, F)$ where Q is the set of states, q_{init} is the initial state, A is the set of atomic actions, $\pi : Q \to \mathbb{T}$ associates each state with a term $t \in \mathbb{T}$, $\delta \subseteq Q \times \Pi \times Q$ is a transition relation and $F \subset Q$ is the set of accepting states.

A process p is said to evolve to a process p' after actions $a_1, a_2, \ldots, a_n$ if there is a sequence of states $q_1, q_2, \ldots q_{n+1}$ in $G(p)$ such that $\pi(q_1) = p$ and $(q_i, a_i, q_{i+1}) \in \delta$ for all $1 \leq i \leq n$ with $\pi(q_{n+1}) = p'$. We can now define the trace of a process p as a sequence of atomic actions $\rho = (a_1, a_2, \ldots, a_n)$ for which the corresponding sequence of states $q_1, q_2, \ldots, q_{n+1}$ exists such that $(q_i, a_i, q_{i+1}) \in \delta$ for all $1 \leq i \leq n$. The set of all traces of a term p is denoted by $\Gamma_{G(p)}$. Note that we have modified the conventional definition of the process graph to include the set of accepting states F, which contains nodes without any outgoing edges. The set of traces of p that end at a state inside F are called *accepting* traces. In this context, let us define a function $\texttt{cost}_G(q)$ to be the length of the shortest accepting trace of q. Note that $\texttt{cost}_G(q) = 0$ if $q \in F$. As an example, the process graph for $c \cdot (a + b) + (d + e)$ shown in Fig. 2.2 has three states with the different traces being $c, c.a, c.b, d, e$. All traces except the first one are accepting traces. Given a BPA expression, the product graph G can be constructed using a simple recursive parser as shown below.

Parse Trees for BPA

PA terms (see Def. 2.7) are composed of atomic actions and sub-terms. In Alg. 2.1 we describe a recursive-descent parser that directly builds the process graph (see Def. 2.8) from a given process algebra term$p \in \mathbb{T}$ by scanning the specification top-down and parsing the sub-terms encountered, while states and transitions of the output process graph are modified accordingly.

The $\texttt{split}(p)$ procedure returns an operator $* \in \{+, \cdot\}$ and terms $p_1, p_2 \in \mathbb{T}$ such that $p = p_1 * p_2$. A process graph for any $p \in \mathcal{T}$ can then be generated by executing $\texttt{parse}(p, q_{init}, q_a, (Q, \delta))$ where q_0, q_a are the initial and final states respectively, and $Q = \{q_{init}, q_a\}$ and $\delta = \varnothing$.

2.4 Differential Games

We now introduce some background in game theory for multi-agent systems that are considered in Chap. 5. Let us first formalize a differential game between two players along with the notion of a Stackelberg equilibrium and then discuss the general n-player Nash equilibrium [BO95].

In this thesis, we consider games between the robot (**R**) and multiple-external agents, the dynamics of all the external agents is clubbed together into a dynamical system which we the "environment" (**E**). We can thus pose this as a two-player differential game. If $x = (x_r, x_e)^T$ be the combined state of the game, the dynamics can be written as

$$\frac{dx}{dt} = f(x(t), u_r(t), u_e(t)) = \begin{bmatrix} f_r(x_r(t), u_r(t)) \\ f_e(x_e(t), u_e(t)) \end{bmatrix}, \tag{2.2}$$

for all $t \in \mathbb{R}_{\geq 0}$ where $x_r \in \mathcal{X}_r \subset \mathbb{R}^{d_r}$, $u_r \in \mathcal{U}_r \subset \mathbb{R}^{m_r}$ and $x_e \in \mathcal{X}_e \subset \mathbb{R}^{d_e}$, $u_e \in \mathcal{U}_e \subset \mathbb{R}^{m_e}$. The functions $f_r : \mathcal{X}_r \times \mathcal{U}_r \to \mathcal{X}_r$ and $f_e : \mathcal{X}_e \times \mathcal{U}_e \to \mathcal{X}_e$ are assumed to be differentiable, measurable and Lipschitz continuous in both their arguments. Given a trajectory of the game $x : [0, T] \to \mathcal{X}$, let the corresponding trajectories of **R** and **E** be x_r and x_e, respectively.

Stackelberg Equilibria

Consider a game where **R** and **E** optimize the cost functions $J_e(x_0, u_r, u_e)$, $J_r(x_0, u_r, u_e)$, respectively. Let the information structure be such that **R** knows the cost function of **E**, but **E** does not know the cost function of **R**, i.e., it has no information of the intention of **R**. **E** however knows the control law

of player **R** and can take it into account while devising its strategy. Define BR to be the mapping from a trajectory $u_r : [0, T] \to \mathcal{U}_r$ to $u_e : [0, T] \to \mathcal{U}_e$ such that

$$\text{BR}(u_r) = \arg\min_{u_e} J_e(x_0, u_r, u_e).$$

$\text{BR}(u_r)$ is thus the best response that **E** can take given a control trajectory of **R**. The player **R** then picks its best strategy u_r^* such that

$$J_r(x_0, u_r^*, u_e^*) \leq J_r(x_0, u_r', \text{BR}(u_r')),$$

where $u_e^* = \text{BR}(u_r^*)$ for any u_r'. Such a strategy, i.e., (u_r^*, u_e^*), is called an open-loop Stackelberg (OLS) equilibrium for this differential game. Necessary conditions for the existence of open-loop Stackelberg equilibria can be characterized as shown in [CCJ72].

Nash Equilibria

Now consider a team of robots, say with indices $\{1, \ldots, N\}$. Each robot is modeled as a dynamical system governed by

$$\dot{x}_i = f_i(x_i(t), u_i(t)), \quad x_i(0) = x_{i,init}, \tag{2.3}$$

where $x_i(t) \in \mathcal{X}_i \subseteq \mathbb{R}^{d_i}$ and $u_i(t) \in \mathcal{U}_i \subset \mathbb{R}^{m_i}$ for all $t \in \mathbb{R}_{\geq 0}$. Given trajectories $\{x_1, x_2, \ldots, x_n\}$ of all agents, where each $x_k : [0, T_k] \to \mathcal{X}_k$, we require that x_ks belong to some computable set, say $\texttt{feasible}_k$. For example, $\texttt{feasible}_k$ consists of all trajectories of agent k that do not collide with other agents and reach some goal region $\mathcal{X}_{goal,k}$. Let $J_k(x_k, u_k)$ be the cost function that agent k would like to minimize; here $u_k : [0, T_k] \to \mathcal{U}_k$ is the control trajectory corresponding to x_k. We can then define the celebrated Nash equilibrium for this problem as follows.

Definition 2.9 (Nash equilibrium). A set of trajectories $\overline{x} = \{x_1, x_2, \ldots, x_n\}$ is a Nash equilibrium if for any $1 \leq k \leq n$, it holds that x_k is feasible and there is no feasible x_k' such that $J_k(x_k', u_k') < J_k(x_k, u_k)$.

The Nash equilibrium is thus a set of strategies such that no agent can improve his cost by unilaterally switching to some other strategy. It can be shown that such an equilibrium is not optimal in the "social" sense. The Pareto optimal equilibrium, on the other hand, is a notion that characterizes the socially optimal behavior.

Definition 2.10 (Social (Pareto) optimum). A set of trajectories $\overline{x} = \{x_1, x_2, \ldots, x_n\}$ is Pareto optimum if all x_ks are feasible and if there does not exist a set $\overline{x'} = \{x_1', x_2', \ldots, x_n'\}$ such that all x_k's are feasible and $J_k(x_k', u_k') < J_k(x_k, u_k)$ for all $1 \leq k \leq n$.

In other words, Pareto optimum is such that no agent can improve its cost without increasing the cost of some other agent, in other words, it is the social optimum if all players cooperate with each other. Necessary conditions for existence of open-loop non-cooperative equilibria, for special cases, e.g., LQR games, can be found in [BO95].

CHAPTER 3

Minimum-violation Planning

This chapter discusses the problem of control synthesis for dynamical systems to fulfill a given reachability goal while satisfying a set of temporal specifications. We focus on goals that become feasible only when a subset of the specifications are violated and motivate a metric known as the level of unsafety which quantifies this violation, for a trajectory of the continuous dynamical system. We will use ideas from sampling-based motion planning algorithms to incrementally construct Kripke structures. A product automaton that captures the specifications is then used in conjunction with the Kripke structure to compute the optimal trajectory that minimizes the level of unsafety.

3.1 Level of Unsafety

Let A be an NFA with some associated natural number, which we call its priority $\varpi(A)$. We assume here that the empty trace by convention is always accepted by A. Define the level of unsafety as follows.

Definition 3.1 (Level of unsafety). Given a word over 2^{Π}, $w = \ell_0, \ell_1, \ldots, \ell_n$ for any index set $I = \{i_1, i_2, \ldots, i_k\} \subset \{0, 1, \ldots, n\}$, define a sub-sequence

$$w' = \ell_0, \ell_{i_j - 1}, \ell_{i_j + 1}, \ldots, \ell_n,$$

where $1 \leq j \leq k$, i.e., w' is obtained by erasing states from I. The level of unsafety $\lambda(w, A)$ is then defined to be

$$\lambda(w, A) = \min_{w' \in L(A)} \langle w \rangle - \langle w' \rangle$$

where $\langle w \rangle$ denotes the length of w and $L(A)$ is the language of automaton A. Similarly, define the level of unsafety of a timed word $w_t(x) = (\ell_0, d_0), (\ell_1, d_1), \ldots, (\ell_n, d_n)$ produced by a trajectory of a dynamical system to be

$$\lambda(x, A) = \min_{w' \in L(A)} \sum_{i \in I} d_i \, \varpi(A).$$

This definition, with a slight abuse of notation thus measures the amount of time $w_t(x)$ violates the specification A. For a trace $\rho = s_0, \ldots, s_{n+1}$ of a durational Kripke structure, it thus becomes

$$\lambda(\rho, A) = \min_{w' \in L(A)} \sum_{i \in I} \Delta(s_i, s_{i+1}) \, \varpi(A).$$

Now consider a non-empty set of rules $\Psi = (\Psi_1, \ldots, \Psi_m)$ with each $\psi_j \in \Psi_i$ given in the form

of some NFA A_{ij} with priority $\varpi(A_{ij})$. The ordered set Ψ and ϖ is then formally defined to be the set of specifications with priorities. We can now easily extend the definition of level of unsafety from that of a single automaton to the set of safety rules as shown below.

Definition 3.2 (Level of unsafety for a set of rules). Level of unsafety with respect to a set of specifications Ψ and priorities ϖ is defined to be

$$\lambda(w, \Psi_i) = \sum_{A_{ij} \in \Psi_i} \lambda(w, A_{ij})$$
$$\lambda(w, \Psi) = [\lambda(w, \Psi_1), \ldots, \lambda(w, \Psi_m)].$$

$\lambda(x, \Psi)$ and $\lambda(\rho, \Psi)$ are defined similarly. $\lambda(w, \Psi)$ as defined above is a m-tuple and we use the lexicographic ordering to compare the level of unsafety of two words.

Thus Ψ_i for $1 \leq i \leq m$ are m priority classes, each with some number of specifications given using finite automata. Def. 3.2 uses the lexicographic ordering, however as we demonstrate in Chap. 5, we can appropriately normalize $\lambda(w, A)$ and construct a scalar version of $\lambda(w, \Psi)$.

3.2 Problem Formulation

We are now ready to formulate the problem of minimum-violation motion planning. Consider a compact set $\mathcal{X} \in \mathbb{R}^d$ and let $x_{init} \in \mathcal{X}$ be the initial state of the system, similarly let $\mathcal{X}_{goal} \subset \mathcal{X}$ be some compact set called as the goal region. Given the dynamical system in Eqn. (2.1), let us define a task specification to be "traveling from x_{init} to $\mathcal{X}_{goal}$". The finite word produced by a trajectory $x : [0, T] \to \mathcal{X}$, $w(x) = \ell_0, \ell_1, \ldots, \ell_n$ is said to satisfy the task Φ if $\ell_0 = \mathcal{L}_c(x_{init})$ and $\ell_n \in \mathcal{L}_c(\mathcal{X}_{goal})$. Similarly, a trace of the Kripke structure K satisfies the task if the labels of the first and final states of the trace are $\mathcal{L}_c(x_{init})$ and $\mathcal{L}_c(\mathcal{X}_{goal})$ respectively. We will assume here that this task is always feasible without considering any specifications. The problem that we tackle in this chapter can then be formally described as follows.

Problem 3.3 (Dynamical system form). *Given a dynamical system as shown in Eqn. (2.1), a task specification Φ, a set of safety rules with priorities (Ψ, ϖ) and some continuous, bounded function $c(x)$ which assigns a non-negative real cost to any trajectory, find a trajectory $x^* : [0, T] \to \mathcal{X}$ such that*

1. *$w(x^*)$ satisfies the task specification Φ;*
2. *$\lambda(x^*, \Psi)$ is the minimum among all trajectories which satisfy the task;*
3. *$c(x^*)$ is minimized among all trajectories that satisfy 1 and 2 above.*

The solution of this problem as defined above exists if the task Φ is feasible. In this work, we restrict ourselves to cost functions $c(x) = \int_0^T 1 \, dt$, i.e., minimum-time cost functions. The algorithm in the sequel can be easily modified for other types of cost, e.g., control and state based cost by appropriate modification of Defs. 2.1 and 3.1. In particular, instead of a "durational" Kripke structure, we would consider a weighted Kripke structure that also stores the cost of optimal trajectory between the two states.

In order to develop an algorithm approach to Prob. 3.3, we convert it into the following problem defined on trace-inclusive durational Kripke structures. We show in Sec. 3.3.3 using Thm. 3.11 that the solutions of these two problems are the same.

Problem 3.4 (Kripke structure form). *Given a durational Kripke structure* $K = (S, s_{init}, R, \Pi, L, \Delta)$ *that is trace-inclusive with respect to the dynamical system in Eqn.*(2.1)*, a task specification* Φ*, a set of safety rules* (Ψ, ϖ)*, find a finite trace* $\rho^* = s_0, s_1, \ldots, s_n$ *such that*

1. ρ^* *satisfies* Φ*,*
2. ρ^* *minimizes* $\lambda(\rho', \Psi)$ *among all other traces* ρ' *of* K *that satisfy the task,*
3. ρ^* *minimizes* $\langle \rho'' \rangle$ *among all traces* ρ'' *that satisfy 1 and 2 above.*

3.3 Algorithm and Analysis

This section describes an algorithm for finding minimum-constraint violation trajectories for a dynamical system. We first construct a weighted automaton whose weights are chosen such that the weight of an accepting run equals the level of unsafety of the input word. We then propose an algorithm, based on RRT*, to incrementally construct a product of the Kripke structure and automata representing safety rules. Roughly, the shortest path in the product uniquely maps to a trace of the Kripke structure that minimizes the level of unsafety. Let us note that the algorithm returns a trajectory that satisfies all rules and minimizes the cost function if it is possible to do so.

3.3.1 *Weighted Product Automaton*

Given an automaton $A_{ij} \in \Psi$, we first augment it with transitions and weights such that the resulting "weighted automaton", $\overline{A}_{ij}$, also accepts all words w that do not satisfy A_{ij}, but it gives them a weight. This weight is picked in such a way that the weight of a non-accepting run is exactly equal to the level of unsafety (cf. Def. 3.1). The objective is to combine all $\overline{A}_{ij}$ into a single weighted automaton $\overline{A}_\Psi$ that weights its input words according to safety rules Ψ with priorities ϖ. In line with the usual model checking procedure, we then construct the product of the Kripke structure K with $\overline{A}_\Psi$. The crucial aspect of this work is that in addition to generating K incrementally using sampling-based methods, we can also construct the weighted product P incrementally.

We now proceed to describe each of these steps in detail and summarize the purpose of each construction in a lemma.

Definition 3.5 (Weighted automaton). For a non-blocking NFA $A = (Q, q_{init}, \Sigma, \delta, F)$, the weighted NFA is defined as $\overline{A} = (Q, q_{init}, \Sigma, \overline{\delta}, F, \overline{W})$ where $\overline{\delta} = \delta \cup \{(q, \sigma, q') \mid q, q' \in Q,\ \sigma \in \Sigma\}$ and

$$\overline{W}(\tau) = \begin{cases} 0 & \text{if } \tau \in \delta \\ \varpi(A) & \text{if } \tau \in \overline{\delta} \setminus \delta. \end{cases}$$

Lemma 3.6. *Language of* $\overline{A}$ *is* Σ^* *and weight of its shortest accepting run* w *is equal to* $\lambda(w, A)$*.*

Proof. The proof of this lemma follows from Lem. 1 in [THK+13]. ■

Definition 3.7 (Automaton $\overline{A}_\Psi$). The weighted product automaton $\overline{A}_\Psi = (Q_\Psi, q_{init,\Psi}, \Sigma, \delta_\Psi, F_\Psi, W_\Psi)$ of automata A_{ij} is defined as

- $Q_\Psi = Q_{1,1} \ldots \times \ldots Q_{1,m_1} \ldots \times Q_{n,1} \ldots \times \ldots Q_{n,m_n}$;
- $q_{init,\Psi} = (q_{init,1,1}, \ldots, q_{init,n,m_n})$;
- $\tau = (q, \sigma, q') \in \delta_\Psi$ if

- $q = (q_{1,1}, \ldots, q_{n,m_n})$ and $q' = (q'_{1,1}, \ldots, q'_{n,m_n})$ and
- $\tau_{ij} = (q_{ij}, \sigma, q'_{ij}) \in \overline{\delta}_{ij}$ for all $i \in \{1, \ldots n\}$ and $j \in \{1, \ldots, m_i\}$
- $W_\Psi(\tau) = (w_1, \ldots, w_n)$ where $w_i = \sum_{j=1}^{m_i} \overline{W}_{ij}(\tau_{ij})$.

- $F_\Psi = \{(q_{1,1}, \ldots, q_{n,m_n}) \mid q_{i,j} \in F_{i,j}, \text{ for all } i \in \{1, \ldots, n\}, j \in \{1, \ldots m_i\}\}$.

The following lemma is now immediate from the above definition in conjunction with Lem. 3.6.

Lemma 3.8. *Language of $\overline{A}_\Psi$ is also Σ^* and the weight of the shortest accepting run w is $\lambda(w, \Psi)$.*

Let us now defined the weighted product automaton that we will work with. It is just a NFA with weights, expect that in our case, the symbol of the automaton is a subset of $2^\Pi \times 2^\Pi$, i.e., the semantics of this automaton are defined over the 2-tuples of labels of consecutive states in the trace of the Kripke structure K; note that the labeling function L of the Kripke structure assigns an element of 2^Π to each state.

Definition 3.9 (Weighted product automaton). The weighted product automaton of $K = (S, s_{init}, R, \Pi, L, \Delta)$ and $\overline{A}_\Psi$ is given by $P = K \otimes \overline{A}_\Psi = (Q_P, q_{init,P}, \Sigma_P, \delta_P, F_P, W_P)$ where

- $Q_P = S \times Q_\Psi$;
- $q_{init,P} = (s_{init}, q_{init,\Psi})$;
- $\Sigma_P = 2^\Pi \times 2^\Pi$;
- $\delta_P \subseteq Q_P \times Q_P$ and $\tau = (\langle s, q \rangle, \langle s', q' \rangle) \in \delta_P$ if $(s, s') \in R$ and $(q, \sigma, q') \in \delta_\Psi$ where $\sigma = (\sigma_s, \sigma_{s'})$ such that $\sigma_s \subseteq \mathcal{L}_c(s)$ and $\sigma_{s'} \subseteq \mathcal{L}_c(s')$;
- $W_P(\tau) = (w_1 \Delta(s, s'), \ldots, w_n \Delta(s, s'))$ where $w_i = W_\Psi(q, \sigma, q')$;
- $F_P = (S \cap \mathcal{X}_{goal}) \times F_\Psi$ is the set of accepting states.

A run of the weighted product automaton is a sequence $\rho = q_{P,0}, q_{P,1}, \ldots, q_{P,n}$ such that $q_{P,n} = (s_n, q_n) \in F_P$, i.e., $s_n \in \mathcal{X}_{goal}$ and $q_n \in F_\Psi$. The weight of the run is the component wise sum of the weights along every transition $(q_{P,i}, q_{P,i+1})$. A useful property easily follows from the construction of this product. As the following lemma shows, the weight of the shortest accepting run of the weighted product automaton projects on the trace of K that minimizes the level of unsafety.

Lemma 3.10. *The shortest accepting run of P, say $q_{P,0}, q_{P,1}, \ldots, q_{P,n}$, projects onto a trace of K that minimizes the level of unsafety, in lexicographical ordering with respect to W_P.*

Proof. This proof follows from Lem. 3.8 and the following two observations:

1. Given a trace $\rho = \rho_0, \ldots, \rho_n$ of K, the word $w(\rho)$ is accepted by P. Moreover, the weight of the shortest accepting run of P over $w(\rho)$ is equal to $\lambda(\rho, \Psi)$.
2. Given an accepting run of P, say $q_{P,0}, q_{P,1}, \ldots, q_{P,n}$, with $q_{P,k} = (s_k, q_k)$, the weight of the run is $\lambda(w(\rho), \Psi)$ and therefore the trace that minimizes the level of unsafety is simply $s_0, s_1, \ldots, s_n$.

■

3.3.2 *Preliminary Procedures*

In this section, we incrementally construct the weighted product automaton (cf. Def. 3.9) and maintain the trace that minimizes the level of unsafety for a set of safety rules Ψ. A few preliminary procedures of the algorithm are as follows :

1. *Sampling*: The `sample` procedure samples an independent, identically distributed state s from a uniform distribution supported over the bounded set $\mathcal{X}$.
2. *Nearest neighbors*: The `Near` procedure returns the set,

$$S_{near}(s) = \{s' \mid \|s' - s\|_2 \leq \gamma \left(\frac{\log n}{n}\right)^{1/d};\ s' \in S\}$$

   where $n = |S|$ and γ is a constant given in Thm. 3.11.

   The above definition of the `near` procedure searches for all nodes within the L_2 ball around a state s. However, while implementing this algorithm on dynamical systems with non-holonomic constraints, most of the states returned by $\texttt{near}(s)$ are not small-time reachable. Computationally, it is then very useful to construct a data-structure that allows us to compute the $\mathcal{O}(\log n)$ states *within* the reachable sub-space that are closest to s according to the cost metric $c(x)$. It turns out that an approximation, i.e., upper and lower bounds of the reachable space can be computed efficiently using the Ball-Box Theorem (see [KF13] for more details). For the dynamics of a Dubins car considered in this paper, the size of the boxes is computed in [KF13].
3. *Steering*: Given two states s, s', the $\texttt{Steer}(s', s)$ procedure computes the pair (x, T) where $x : [0, T] \to X$ is a trajectory such that, (i) $x(0) = s'$, (ii) $x(T) = s$ and, (iii) x minimizes the cost function $\texttt{cost}(x) = T$. If a trajectory x is found return true else return false.
4. *Connecting*: For a state $s' \in S_{near}$, if $\texttt{steer}(s', s)$ returns true, for all nodes $z' = (s', q') \in Q_P$, for all $(z', (s, q)) \in \delta_P$, the procedure $\texttt{connect}(s', s)$ adds the state $z = (s, q)$ to the set Q_P, adds (z', z) to δ_P and calculates $W_P(z', z)$. If $s \in \mathcal{X}_{goal}$ and $q \in F_\Psi$, it adds (s, q) to F_P.
5. *Updating costs*: The procedure $\texttt{update\_cost}(s)$ updates the level of unsafety $J_a(z)$ and the cost $J_t(s)$ from the root for a node $z = (s, q)$ as shown in Alg. 3.2 using the sets,

$$S_{steer}(s) = \{s' \mid s' \in S_{near}(s);\ \texttt{steer}(s', s) \text{ returns true}\},$$

$$Z_{steer}(s) = \{(s', q') \mid s' \in S_{steer}(s);\ (s', q') \in Q_P\}.$$

6. *Rewiring*: In order to ensure asymptotic optimality, the `rewire` procedure recalculates the best parent, i.e., $\texttt{parent}(s')$ for all states $s' \in S_{near}(s)$ as shown in Alg. 3.3. The complexity of this procedure can be reduced by noting that s' only needs to check if the new sample can be its parent by comparing costs J_a, J_t, otherwise its parent remains the same.

Finally, Alg. 3.1 creates the weighted product automaton as defined in Def. 3.9 incrementally. It also maintains the best state $z^* = (s^*, q^*) \in F_P$. The trace $\rho^* = s_0, s_1, \ldots, s_n$ of the Kripke structure K that minimizes the level of unsafety and is a solution to Prob. 3.4 can then be obtained from z^* by following $\texttt{parent}(s^*)$. Since K is trace-inclusive, the continuous-time trajectory x^* can be obtained by concatenating smaller trajectories. Let (x_i, T_i) be the trajectory returned by $\texttt{steer}(s_i, s_{i+1})$ for all states $s_i \in r^*$. The concatenated trajectory $x^* : [0, T] \to \mathcal{X}$ is such that $T = \sum_{i=0}^{n-1} T_i$ and $x_n(t + \sum_{k=0}^{i-1} T_k) = x_i(t)$ for all $i < n$.

Algorithm 3.1: MVRRT*

Input : $x_{init}, \mathcal{X}, \Sigma_P, n, \overline{A}_\Psi$;
$P \leftarrow \emptyset$; $Q_P \leftarrow q_{P,init}$;
$J_a(x_{init}) \leftarrow 0$; $J_t(x_{init}) \leftarrow 0$;
$i \leftarrow 0$;
for $i \leq n$ **do**
  // Sample new state
  $s \leftarrow$ sample;
  // Connect neighbors
  for $s' \in$ near(s) **do**
    if steer(s', s) **then**
      connect(s', s);
  // Find best parent
  parent, $J_a, J_t \leftarrow$ update_cost(s);
  // Rewire neighbors
  $P, J_a, J_t \leftarrow$ rewire(s);
$P_n \leftarrow (Q_P, q_{P,init}, \Sigma_P, \delta_P, F_P, W_P)$;
return P_n

Algorithm 3.2: update_cost(s, P)

for $z = (s, q) \in Q_P$ **do**
  $J_a(z) \leftarrow \min_{z' \in Z_{steer}} W_P(z', z) + J_a(z')$;
  $Z^* \leftarrow \arg\min_{z' \in Z_{steer}} W_P(z', z) + J_a(z')$;
  $J_t(s) \leftarrow \min_{z' \in Z^*}$ cost$(s', s) + J_t(s')$;
  parent$(z) \leftarrow \arg\min_{z' \in Z^*}$ cost$(s', s) + J_t(s')$;
return parent, J_a, J_t

Algorithm 3.3: rewire(s, P)

for $s' \in S_{steer}(s)$ **do**
  if steer(s, s') **then**
    connect(s, s');
  $J_a, J_t \leftarrow$ update_cost(s');
return P

Figure 3.1: Incremental algorithm for construction of the product automaton

3.3.3 *Analysis*

In this section, we analyze the convergence properties of Alg. 3.1. In particular, we prove that the continuous-time trajectory x_n given by the algorithm after n iterations converges to the solution of Prob. 3.3 as the number of states in the durational Kripke structure K_n goes to infinity, with probability one. A brief analysis of the computational complexity of the algorithm is also carried out here. Due to lack of space, we only sketch the proofs.

Theorem 3.11. *The probability that Alg. 3.1 returns a durational Kripke structure K_n and a trajectory of the dynamical system x_n, that converges to the solution of Prob. 3.3 in the bounded variation norm sense, approaches one as the number of states in K_n tends to infinity, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} \|x_n - x^*\|_{BV} = 0\}\right) = 1$$

Proof. (Sketch) The proof primarily follows from the asymptotic optimality of the RRT* algorithm (see Theorem 34 in [KF11b]). Let $x^* : [0, T] \to \mathcal{X}$ be the solution of Prob. 3.3 that satisfies the task Φ and minimizes the level of unsafety. For a large enough n, define a finite sequence of overlapping balls $B_n = \{B_{n,1}, \ldots, B_{n,m}\}$ around the optimal trajectory x^*. The radius of these balls is set to be some fraction of $\gamma(\log n/n)^{1/d}$ such that any point in $s \in B_{n,m}$ can connect to any other point $s' \in B_{n,m+1}$ using the steer(s, s') function. It can then be shown that each ball in B_n contains at least one state of K_n with probability one.

In such a case, there also exists a trace $\rho_n = s_0, s_1, \ldots, s_n$ of K_n such that every state s_i lies in some ball $B_{n,m}$. Also, for a large enough n, the level of unsafety of ρ_n, $\lambda(\rho_n, \Psi)$ is equal to the level of unsafety of the word generated by the trajectory x^*, $\lambda(x^*, \Psi)$, i.e., the algorithm returns the trace with the minimum level of unsafety among all traces of the Kripke structure K satisfying the task Φ. Finally, it can be shown that the trajectory x_n constructing by concatenating smaller

trajectories joining consecutive states of ρ, i.e., $s_0, s_1, \ldots$ converges to x^* almost surely as $n \to \infty$.

In this proof, $\gamma > 2\,(2+1/d)^{1/d}\,(\mu(\mathcal{X})/\zeta_d)^{1/d}$, where $\mu(\mathcal{X})$ is the Lebesgue measure of the set $\mathcal{X}$ and ζ_d is the volume of the unit ball of dimensionality d. ■

The following lemma is an immediate consequence of Thm. 3.11 and the continuity of the cost function $c(x)$.

Lemma 3.12. *The cost of the solution converges to the optimal cost, $c^* = c(x^*)$, as the number of samples approaches infinity, almost surely, i.e,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} c(x_n) = c^*\}\right) = 1.$$

Let us now comment on the computational complexity of MVRRT*.

Lemma 3.13. *The expected computational complexity of Alg. 3.1 is $\mathcal{O}(m^2 \log n)$ per iteration where n is the number of states in the durational Kripke structure K and m is the number of states of the automaton $\overline{A}_\Psi$.*

Proof. We do a rough analysis of Alg. 3.1. Note that there are an expected $\mathcal{O}(\log n)$ samples in a ball of radius $\gamma(\log n/n)^{1/d}$. The procedure `steer` is called on an expected $\mathcal{O}(\log n)$ samples while because the automaton $\overline{A}_\Psi$ is non-deterministic, the procedure `connect` adds at most m^2 new states in the product automaton per sample. The procedure `update_cost` requires at most $\mathcal{O}(m^2 \log n)$ time call. The `rewire` procedure simply updates the parents of the $\mathcal{O}(\log n)$ neighboring samples which take $\mathcal{O}(m^2 \log n)$ time. In total, the computational complexity of the algorithm is $\mathcal{O}(m^2 \log n)$ per iteration. ■

3.4 Experiments

In this section, we consider an autonomous vehicle modeled as a Dubins car in an urban environment with road-safety rules and evaluate the performance of MVRRT* in a number of different situations.

3.4.1 *Setup*

Consider a Dubins car, i.e., a curvature-constrained vehicle with dynamics

$$\begin{aligned} \dot{x}(t) &= v\cos\theta(t), \\ \dot{y}(t) &= v\sin\theta(t), \\ \dot{\theta}(t) &= u(t). \end{aligned} \tag{3.1}$$

The state of the system is the vector $[x, y, \theta]^T$, and the input is $u(t)$, where $|u(t)| \leq 1$ for all $t \geq 0$. The vehicle is assumed to travel at a constant speed v. As shown in [Dub57], time-optimal trajectories for this system in an obstacle-free environment can be easily calculated.

We partition the working domain $\mathcal{X}$ into compact non-empty subsets $\mathcal{X}_{obs}$ which is the union of obstacle regions, $\mathcal{X}_{sw}$ which represents the sidewalk and $\mathcal{X}_{rl}$, $\mathcal{X}_{ll}$ which are the right and left lanes, respectively, as illustrated in Fig. 3.2. $\mathcal{X}_{obs}$ is empty if there are no obstacles.

Based on this partitioning, we define the set of atomic propositions as

$$\Pi = \{sw, rl, ll, dir, dotted, solid\}.$$

A proposition $p \in \{sw, rl, ll\}$ is true at a state $s \in \mathcal{X}$, if $s \in \mathcal{X}_p$ with rl, ll being mutually exclusive. *dir* is true iff the heading of the car is in the correct direction, i.e., if s is such that the car heading forwards and rl is true. Atomic propositions, *dotted* and *solid*, depict the nature of lane markers. Note that obstacles are not considered while constructing Π since we do not desire a trajectory that goes over an obstacle. The `steer` procedure in Sec. 3.3.2, instead, returns false if any state along the trajectory lies in $\mathcal{X}_{obs}$. This change does not affect the correctness and the overall complexity of MVRRT*.

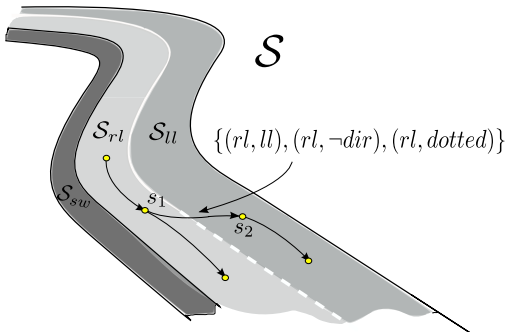


Figure 3.2: Partitions of the working domain $\mathcal{X}$. The transition from s_1 to s_2 is labeled with, for example, $\{(rl, ll), (rl, \neg dir), (rl, dotted)\}$.

3.4.2 *Safety Rules*

Given a task Φ such as finding a trajectory from x_{init} to the goal region $\mathcal{X}_{goal}$, we require the vehicle to follow the following rules: 1. do not travel on sidewalks (*sidewalk rule*), 2. do not cross solid center lines (*hard lane changing*), 3.a always travel in the correct direction (*direction rule*), 3.b do not cross dotted center lines (*soft lane changing*).

We describe the rules with the following FLTL$_{-X}$ formulas and corresponding finite automata in Fig. 3.3. Note that we use 2-tuples of atomic propositions from Π as the alphabet for both formulas and the automata, to specify not only properties of individual states, but also of transitions. The two components capture the atomic propositions of the starting and the ending state respectively.

1. *Sidewalk*: Do not take a transition that ends in $\mathcal{X}_{sw}$

$$\psi_{1,1} = \mathsf{G} \bigwedge_{*\in 2^{\Pi}} \neg(*, sw)$$

2. *Hard lane change* Do not cross a solid center line

$$\psi_{2,1} = \mathsf{G}\Big(\neg\big((rl, solid) \wedge (rl, ll)\big) \vee \big((ll, solid) \wedge (ll, rl)\big)\Big)$$

3. a. *Direction* Do not travel in the wrong direction

$$\psi_{3,1} = \mathsf{G} \bigvee_{*\in 2^{\Pi}} (*,\ dir)$$

b. *Soft lane change* Do not cross a dotted center line

$$\psi_{3,2} = \mathsf{G}\Big(\neg\big((rl, dotted) \wedge (rl, ll)\big) \vee \big((ll, dotted) \wedge (ll, rl)\big)\Big)$$

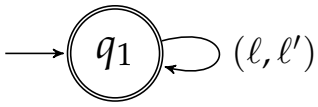


Figure 3.3: Rule 3.b : For the sake of brevity, the transition above represents all transitions, where *(i)* $\ell, \ell' \subseteq 2^{\Pi}$, such that $rl \in \ell$ and $dotted, ll \in \ell'$, or $ll \in \ell$ and $dotted, rl \in \ell$, and *(ii)* $\ell, \ell' \subseteq 2^{\Pi}$, such that $rl \in \ell$ and $solid, ll \in \ell'$, or $ll \in \ell$ and $solid, rl \in \ell$.

The finite automata for rules 1-3 are all of the same form (see Fig. 3.3).

While it is quite natural to disobey the direction and the soft lane change rules, a solid line should not be crossed. This gives three different priority classes

$$(\Psi_1, \Psi_2, \Psi_3), \varpi) = ((\{\psi_{1,1}\}, \{\psi_{2,1}\}, \{\psi_{3,1}, \psi_{3,2}\}), \varpi),$$

where $\varpi(\psi_{1,1}) = \varpi(\psi_{2,1}) = \varpi(\psi_{3,1}) = 1$ and $\varpi(\psi_{3,2}) = 10$. Note that costs for $\psi_{2,1}$ and $\psi_{3,2}$ are incurred only once per crossing and do not depend upon the duration of the transition. Within the third class, we put higher priority on the soft lane change rule to avoid frequent lane switching, for instance in case two obstacles are very close to each other and it is not advantageous to come back to the right lane for a short period of time, e.g., see Fig. 3.5.

3.4.3 *Simulation Experiments*

MVRRT* was implemented in C++ on a 2.2GHz processor with 4GB of RAM for the experiments in this section. We present a number of different scenarios in the same environment to be able to quantitatively compare the performance. In Fig. 3.5, the Dubins car starts from the lower right hand corner while the goal region marked in green is located in the lower left hand corner. Light gray denotes the right and left lanes, $\mathcal{X}_{rl}$ and $\mathcal{X}_{ll}$. A sidewalk $\mathcal{X}_{sw}$ is depicted in dark gray. The dotted center line is denoted as a thin yellow line while solid center lines are marked using double lines. Stationary obstacles in this environment are shown in red.

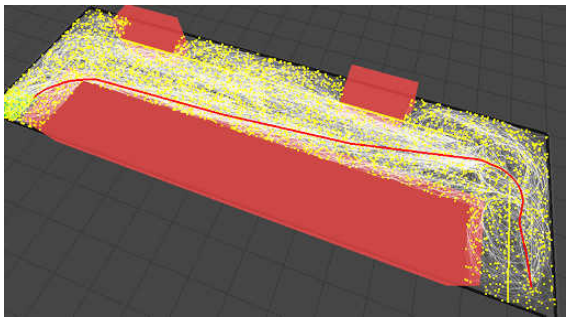
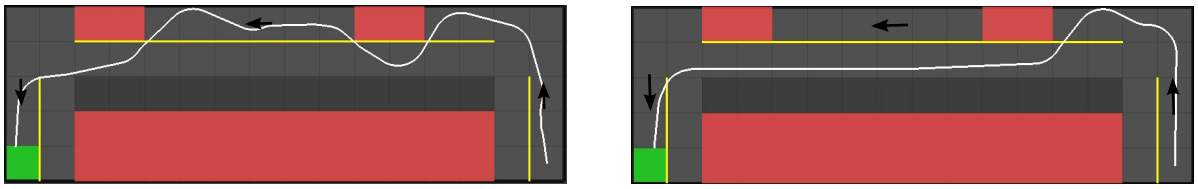


Figure 3.4: MVRRT* tree after 40 sec. on an example without any safety rules. States of the Kripke structure are shown in yellow while edges are shown in white. The shortest trajectory shown in red to the goal region avoids obstacles but uses the sidewalk.

***Case 1*:** First, we consider a scenario without any safety or road rules. The MVRRT* algorithm then simply aims to find the shortest obstacle-free trajectory from the initial state to the goal region. Note that in this case, MVRRT* performs the same steps as the RRT* algorithm. The solution computed after 40 seconds has a cost of 88.3 and is illustrated in Fig. 3.4 together with the adjoining tree.

***Case 2*:** Next, we introduce the sidewalk rule $\psi_{1,1}$ and the direction rule $\psi_{3,1}$. Without any penalty on frequent lane changing, the car goes back into the right lane after passing the first obstacle. It has to cross the center line again in order to pass the second obstacle and reach the goal region. Fig 3.5a depicts the solution that has a cost of 122.3 along with a level of unsafety of 46.4 for breaking $\psi_{3,1}$.



(a) (b)

Figure 3.5: Fig. 3.5a shows the solution after 60 secs. for sidewalk and direction rules. Upon introducing the soft lane changing rule in Fig. 3.5b, the vehicle does not return to the right lane after passing the first obstacle.

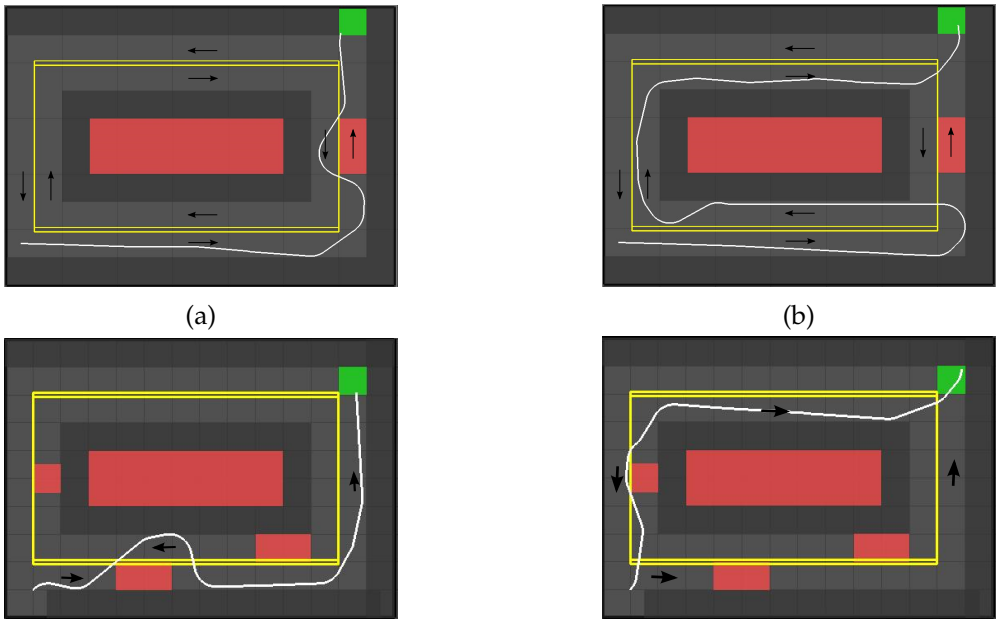
Upon introducing the rule $\psi_{3,2}$, the vehicle does not go back into the right lane after passing the first obstacle. Figure 3.5b shows this solution with a level of unsafety of 84.1 for breaking both $\psi_{3,1}$ and $\psi_{3,2}$ whereas the level of unsafety in this case for the trajectory in Fig. 3.5a is 87.4.

***Case 3*:** Fig 3.6a shows a run for the sidewalk, direction and soft lane changing rules after 60 secs. of computation time with a level of unsafety of $(0, 0, 28.3)$. In Fig. 3.6b, with 120 secs. of computation, the solution has a much higher cost (215.8) but a significantly lower level of unsafety $(0, 0, 1.6)$ because it only breaks the direction rule slightly when it turns into the lane. This thus demonstrates the incremental and anytime nature of the algorithm. *Case 4*: In our last example, we introduce hard and soft lane changing rules along with sidewalk and direction rules. After 15 secs., MVRRT* returns the solution shown in Fig. 3.6c, which breaks the hard lane changing rule twice, thereby incurring a level of unsafety of $(0, 2, 48.1)$ for the three rules. On the other hand, after about 300 secs., the solution converges to the trajectory shown in Fig. 3.6d which breaks the hard lane changing rule only once, this has a level of unsafety of $(0, 1, 25.17)$.

3.4.4 *Real Experiments*

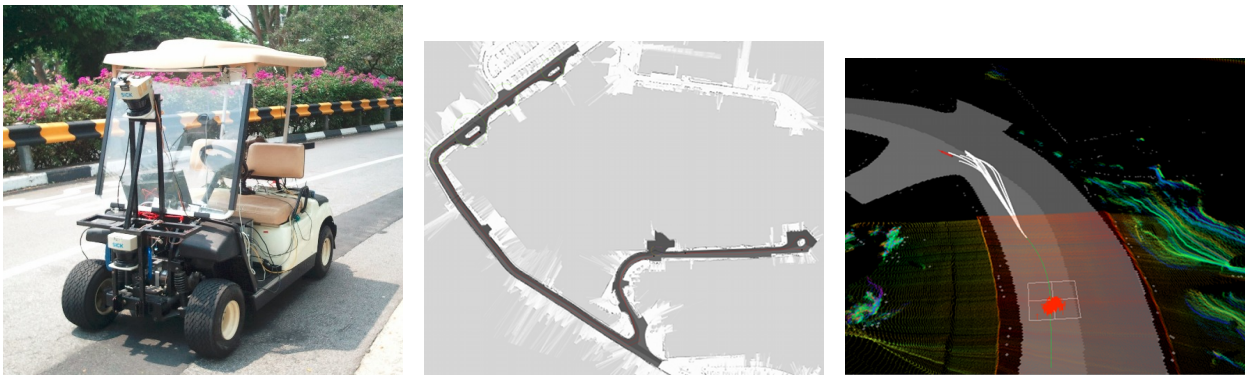
In this section, we present results of our implementation of MVRRT* on an autonomous golf-cart shown in Fig. 3.7 as a part of an urban mobility-on-demand system in the National University of Singapore's campus. The golf-cart was instrumented with two SICK LMS200 laser range finders and has drive-by-wire capability. The algorithm was implemented inside the Robot Operating System (ROS) [QCG+09] framework.

***Setup and implementation details*:** Traffic lanes and sidewalk regions are detected using pre-generated lane-maps of the campus roads, while obstacles are detected using data from laser range-finders. We use the sidewalk, direction and soft-lane changing rules for the experiments here. For an on line implementation of MVRRT*, we incrementally prune parts of Kripke structure that are unreachable from the current state of the golf-cart. The algorithm adds new states in every iteration (Lines 5-10 in Alg. 3.1) until the change in the level of unsafety of the best trajectory is within acceptable bounds between successive iterations. This trajectory is then passed to the controller that can track Dubins curves. We use techniques such as branch-and-bound and biased sampling to enable a fast real-time implementation and the golf-cart can travel at a speed of approximately 10 kmph while executing the algorithm. Fig. 3.7 shows a snapshot of the experimental setup while Fig. 3.8 shows an instance of the golf-cart going into the incoming lane in order to overtake a stalled car in its lane. Note that traffic in Singapore drives on the left hand side.



(a) (b) (c) (d)

Figure 3.6: Fig. 3.6a and 3.6b show the solution of MVRRT* after 60 and 120 secs. respectively, with the sidewalk, direction and soft lane changing rules. Note that the algorithm converges to a long trajectory which does not break any rules. Fig. 3.6c shows a solution after 20 secs. which breaks the hard lane changing rule twice. After 120 secs., the algorithm converges to the solution shown in Fig. 3.6d, which features only one hard lane change and three soft lane changes.



(a) (b) (c)

Figure 3.7: Fig. 3.7a shows the Yamaha golf-cart instrumented with laser range-finders and cameras. Fig. 3.7c shows the on line implementation of MVRRT* in ROS. Red particles depict the estimate of the current position of the golf-cart using laser data (shown using colored points) and adaptive Monte-Carlo localization on a map of a part of the NUS campus shown in Fig. 3.7b. Trajectories of the dynamical system, that are a part of the Kripke structure are shown in white while the trajectory currently being tracked is shown in green.

(a) (b)

(c) (d)

Figure 3.8: The autonomous golf-cart comes back into the correct lane after overtaking a stalled vehicle in spite of the road curving to the right. Note that the optimal trajectory without road-safety rules would cut through the incoming lane to reach the goal region.

CHAPTER 4

Process Algebra Specifications

This chapter investigates specifications using process algebras for complex task specifications for autonomous vehicles. It constructs incremental Kripke structures using ideas from the previous chapter and uses model checking methods based on the process graph of the task specifications to construct trajectories that satisfy the specifications. Throughout, we motivate the approach using tasks such as charging of an electric car at a busy charging station and scheduling pick-ups and drop-offs for capacity constrained mobility-on-demand.

Languages such as Linear Temporal Logic (LTL), Computational Tree Logic (CTL) and deterministic μ-calculus are powerful ways to specify task specifications. With greater expressiveness however comes greater complexity, e.g., checking LTL formulas is PSPACE-complete [Sch02] while model checking deterministic μ-calculus belongs to NP $\cap$ co-NP [Eme97]. In this chapter, we focus on languages which forgo expressive power but in turn allow for efficient model checking algorithms. As an example, note [KGFP07], which restricts specifications to reactive temporal logic to obtain polynomial complexity of translation into automata as opposed to doubly exponential complexity of the full LTL. We use process algebra specifications [Bae05, Fok00] which, although not as expressive as LTL or μ-calculus, are still enough to specify a number of tasks of practical interest. In fact, expressing general specifications in these languages requires that the underlying Kripke structure is a graph, whereas in our work we can converge to optimality even with a tree structure.

Our work is similar to problems considered in [KRKF09, DKCB11]; we however focus on continuous dynamical systems with differential constraints and show that our algorithm converges to the optimal solution in the limit. Another line of literature poses these problems using mixed-integer linear programs (e.g., [ED05, KF08]). These methods are however restricted to linearizable dynamics and discrete time; also the number of integer constraints grows quickly with the number of obstacles. Sampling-based approaches to abstract the continuous dynamics help us alleviate these issues while still maintaining computational efficiency.

The rest of the chapter is structured as follows: first, we formulate the problem using specifications from process algebras (note the similarities here with Sec. 3.2 in Chap. 3). The algorithm then follows the usual model checking program, i.e., it constructs the product automaton incrementally and computes the trace of the product that uniquely maps onto a trajectory of the dynamical system. Next, we provide results of simulation experiments on examples motivated by autonomous vehicles operating in an urban mobility-on-demand scenario viz., charging at a busy charging station and picking up and dropping off passengers at various locations.

4.1 Problem Formulation

This section describes our approach and motivates the algorithm in Sec. 4.2. Roughly, to compute a trajectory that satisfies a specification $p \in \mathbb{T}$, we construct the product of a trace-inclusive Kripke structure K and the process graph $G(p)$. This product is constructed in such a way that accepting traces of the product graph can be mapped to accepting traces of the process graph. A lexicographical cost function shown in Sec. 4.2 then enables us to identify a single trace that minimizes the cost and satisfies the process algebra specification at the same time. We first define the problem using the continuous-time dynamical system as shown below.

Problem 4.1. *Given a dynamical system modeled by Eqn. 2.1 and a process algebra term p, find a trajectory $x^* : [0, T] \to \mathcal{X}$ such that, (i) x^* starts at the initial state, i.e., (i) $x^*(0) = x_0$, (ii) x^* satisfies the task specification p, i.e., destutter$(\rho(x^*))$ is an accepting trace of $G(p)$, and (iii) x^* minimizes the cost function $c(\cdot)$ among all trajectories that satisfy (i) and (ii).*

We again consider cost functions $c(x) = \int_0^T 1\, dt$, i.e., time optimal cost functions. However, the problem formulation is general and can incorporate other kinds of costs such as minimum effort by changing the following definition slightly.

Definition 4.2 (Weighted product graph). Given a Kripke structure $K = (S, s_{init}, R, \Pi, L, \Delta)$ and a process graph $G(p) = (Q, q_0, A, \pi, \delta, F)$ for a process p, the product graph is a labeled transition system defined as the tuple $P = (Q_P, q_{init,P}, A_P, \pi_P, \delta_P, F_P, W_P)$ where:

- $Q_P = S \times Q$ is the set of states;
- $q_{init,P} = (s_{init}, q_{init})$ is the initial state;
- $A_P = S \times S \times A$ is the set of atomic actions;
- $\pi_P((s, q)) = \pi(q)$ for all $(s, q) \in Q_P$;
- $(q_1', a', q_2') \in \delta_P$ iff $(s_1, s_2) \in R$, and either $q_1 = q_2$ or $(q_1, a, q_2) \in \delta$ with $a \in L(s_2)$;
- $F_P = S \times F$ is the set of accepting states;
- $W_P(q_1', q_2') = \Delta(s_1, s_2)$ is a weighing function.

where $q_i' = (s_i, q_i)$ and $a' = (s_1, s_2, a)$.

We use $P = K \oplus G(p)$ to denote the product of a Kripke structure K and a process graph $G(p)$. Let us note how traces of the product graph are related to traces of the process graph. Given a $\rho' = \{a_1', a_2', \ldots, a_n'\} \in \Gamma_P$, where $a_i' = (s_{i,1}, s_{i,2}, a_i)$, if $z_1, z_2, \ldots, z_{n+1}$ are the corresponding states in P, where $z_i = (s_i, q_i)$, the projection on the Kripke structure is $\alpha_K(\rho') = s_{1,1}, s_{2,1}, \ldots, s_{n+1,1}$ and the projection on the process graph is $\alpha_G(\rho') = q_1, q_2, \ldots, q_{n+1}$. The following lemma then follows easily.

Lemma 4.3. *An accepting trace of the product uniquely maps on to an accepting trace of $G(p)$.*

Proof. Every accepting trace ρ' of P is such that the corresponding sequence of states, $q_1', q_2', \ldots, q_{n+1}'$ ends with $q_{n+1}' \in F_P$. But since $F_P = S \times F$, $q_{n+1}' = (s_{n+1}, q_{n+1})$ is such that $q_{n+1} \in F$ as well. Also, since a transition in the product graph $(q_1', a', q_2') \in \delta_P$ if and only if $(s_1, s_2) \in R$ and either $a \in \mathcal{L}_c(s_2)$ with $(q_1, a, q_2) \in \delta$ or $q_1 = q_2$, we have that its projection on the process graph, i.e., $\alpha_G(\rho') = q_1, \ldots, q_{n+1}$ is also unique. ∎

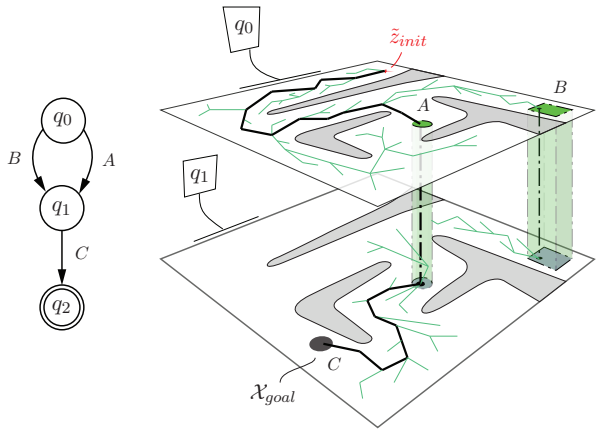


Figure 4.1: Conceptually, the product graph is the Cartesian product of the process graph with the Kripke structure and can be visualized using layers. Given a term, e.g., $(A + B) \cdot C$, and the Kripke structure shown using green lines, the optimal trace in the product graph goes to either region A or region B on the same layer and then travels downwards to reach region C, whichever minimizes the cost.

Problem 4.4. *Given a trace-inclusive Kripke structure $K = (S, s_{init}, R, \Pi, L, \Delta)$ and a process p, find a trace ρ^* of the product graph $P = K \oplus G(p)$ such that,*

1. *$\alpha_K(\rho^*) = s_1 s_2 \dots s_n$ is such that $s_1 = s_{init}$;*
2. *$destutter(\alpha_G(\rho^*))$ is an accepting trace of $G(p)$;*
3. *ρ^* minimizes $\texttt{cost}(\rho')$ among all traces satisfying 1, 2.*

Algorithm 4.1: PARRT*

```
Input : n, 𝒳, x_init, G(p);
z_0 ← (s_init, q_init), Q_P ← {z_0};
cost(z_0) ← (cost_G(q_0), 0);
i ← 0;
for i ≤ n do
    s ← sample;
    for z' ∈ near(s) do
        for q ∈ Q do
            z ← (s, q), (s', q') ← z';
            x, u, T ← steer(z', z);
            Q_P ← Q_P ∪ {z}, δ_P ← δ_P ∪ {(z', z)};
            W_P(z', z) ← T;
            if q ∈ F then
                F_P ← F_P ∪ {z};
            c ← calculate_cost(z', z);
            if c < cost(z) then
                parent(z) ← z', cost(z) ← c;
    rewire(s);
P_n ← (Q_P, q_{P,init}, δ_P, F_P, W_P);
return P_n
```

4.2 Algorithm

In this section, we describe the algorithm used to construct the product graph in Def. 4.2. This algorithm is very similar to Alg. 3.1 in Chap. 3 with superficial changes. Instead of using a weighted product automaton, we use an ordered tuple of the length of the shortest accepting trace from a node and the weighing function of the product graph as costs in this algorithm. Please refer to Sec. 3.3.2 for the procedures `sample`, `steer`, `near` and `rewire`. We describe the procedure to compute the cost here which differs from the `update_cost` procedure in Alg. 3.1.

Calculate Cost: Let the nodes of the process graph be denoted by $z = (s, q)$ and $z' = (s', q')$. Each node maintains a 2-tuple $\texttt{cost}(z) = (\texttt{cost}_G(q), \texttt{cost}_K(s))$ which is an ordered tuple of the minimum number of steps that q takes to reach an accepting state in G and the cost of reaching a state s in the Kripke structure. Given z', z, the `calculate_cost` procedure returns the cost, $\texttt{cost}(z)$, of reaching z through z'. $\texttt{cost}_K(z', z)$ is the cost returned by $\texttt{steer}(z', z)$, i.e., $W_P(z', z)$, and hence $\texttt{cost}_K(z) = \texttt{cost}_K(z') + \texttt{cost}_K(z', z)$. Note that $\texttt{cost}_G(z)$ can be calculated from the process graph. We use the lexicographical ordering to compare cost tuples.

Alg. 4.1 also maintains the state $z^* = (s^*, p^*)$ in the product with the least cost. Let the trace constructed by following $\texttt{parent}(z^*)$ backwards be $z_0, z_1, z_2, \ldots, z_m$ where $z_0 = (s_{init}, q_0)$ and $z_m = z^*$ with ρ^* being the corresponding trace in Γ_P. The projection $\alpha_K(\rho^*) = s_0, s_1, s_2, \ldots, s_m$ gives the trace on the Kripke structure. Since K is trace-inclusive, we can construct a continuous trajectory of the dynamical system after n iterations of Alg. 4.1, say x_n by concatenating the outputs of the `steer` procedure between s_i, s_{i+1}. The following analysis shows that this trajectory converges to the optimal solution of Prob. 4.1.

Theorem 4.5. *As the number of states in the Kripke structure tends to infinity, x_n converges to x^* in the bounded variation norm sense, almost surely, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} \|x_n - x^*\|_{BV} = 0\}\right) = 1.$$

The proof of the above theorem is very similar to that of Thm. 3.11 and is hence omitted. Roughly, it can be shown that for a large enough n, the Kripke structure is such that it contains states in the neighborhood of the optimal trajectory x^*. This neighborhood is a function of the $k \log n$ nearest neighbors of the set S_{near} considered in the `near` procedure. Instead of covering the optimal trajectory x^* with an overlapping sequence of balls of radius $\rho(\log n/n)^{1/d}$, we cover it with an overlapping sequence of scaled boxes from the Ball-Box Theorem which also have volume $\mathcal{O}(\log n/n)$. The result then follows by noticing that a transition in the Kripke structure exists for any two states lying in adjacent boxes, i.e., there exists a trace of the Kripke structure around the optimal trajectory x^*. It can be shown that the continuous trajectory constructed from this trace converges to the optimal trajectory almost surely [KF11b].

Computational Complexity

The computational complexity of Alg. 4.1 is similar to Alg. 3.1 in Chap. 3. For a process algebra expression p of length m, the size of process graph can be upper bounded by m. Therefore, at most $\mathcal{O}(m^2 \log n)$ states are added to the product graph in lines 11-14 and the total computational complexity in expectation can then be shown to be $\mathcal{O}(m^2 \log n)$ per iteration.

4.3 Experiments

4.3.1 *Local Planner*

Consider an autonomous electric vehicle that needs to charge at a charging station. In the event that the charging station is already occupied, the vehicle waits in an empty parking spot and proceeds to charge once the charging station becomes free. This behavior, which we call *local planner* is a direct application of Alg. 4.1 and is discussed in detail in this section. All examples in the following two sections were implemented in `C++` using the Robot Operating System (ROS) platform [QCG+09] on a computer with a 2.0 GHz processor and 4 GB of RAM.

Similar to Chap. 3, we model the autonomous car as a Dubins vehicle with the dynamics

$$\dot{x}(t) = v\cos\theta(t), \quad \dot{y}(t) = v\sin\theta(t), \quad \dot{\theta}(t) = u, \tag{4.1}$$

where the state of the vehicle is $[x, y, \theta]^T$ while the control input is the turning rate, u with $|u(t)| \leq 1$ for all times $t > 0$. The velocity v is taken to be a constant. Time optimal trajectories for this dynamics can be easily calculated using Dubins curves [Dub57]. In particular, a trajectory between two states that minimizes time is a concatenation of curves from the set $\{L, C, R\}$, where L, R denote turning left and right at maximum turning rate, respectively, while C denotes $u = 0$.

Fig. 4.3a shows the experimental scenario of a charging station inside a parking lot. The station is labeled s_1 while parking lots are labeled $w_1, w_2, \ldots, w_4$. The charging specification can then be written as

$$\Phi_c = s_1 + (w_1 + w_2 + w_3 + w_4) \cdot\ s_1, \tag{4.2}$$

which express the task "either directly go to charging station" or "first go to one of the parking spots and then go to the charging station".

The problem domain $\mathcal{X}$ is split into disjoint sets $\mathcal{X}_{obs}$, $\mathcal{X}_{free}$. In case of multiple obstacles, $\mathcal{X}_{obs}$ is the union of all states $x \in \mathcal{X}$ that lie inside obstacles while $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$. We define a region $\mathcal{X}_{a_i}$ for each atomic action a_i in the set $A = \{a_1, a_2 \ldots, a_n\}$. A state $x \in \mathcal{X}_{a_i} \subset \mathcal{X}$ if taking an action a_i from any other state x' results in x. Regions corresponding to atomic actions $\{w_1, w_2, w_3, w_4, s_1\}$ are shown in Fig. 4.3b. We modify the `steer` procedure to return false if the trajectory passes through $\mathcal{X}_{obs}$. Note that this does not affect the convergence properties of Alg. 4.1.

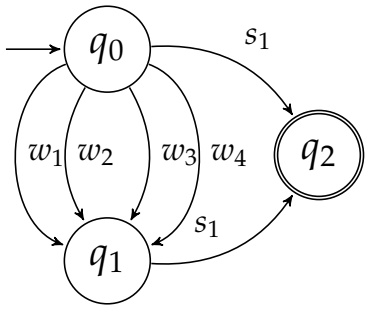
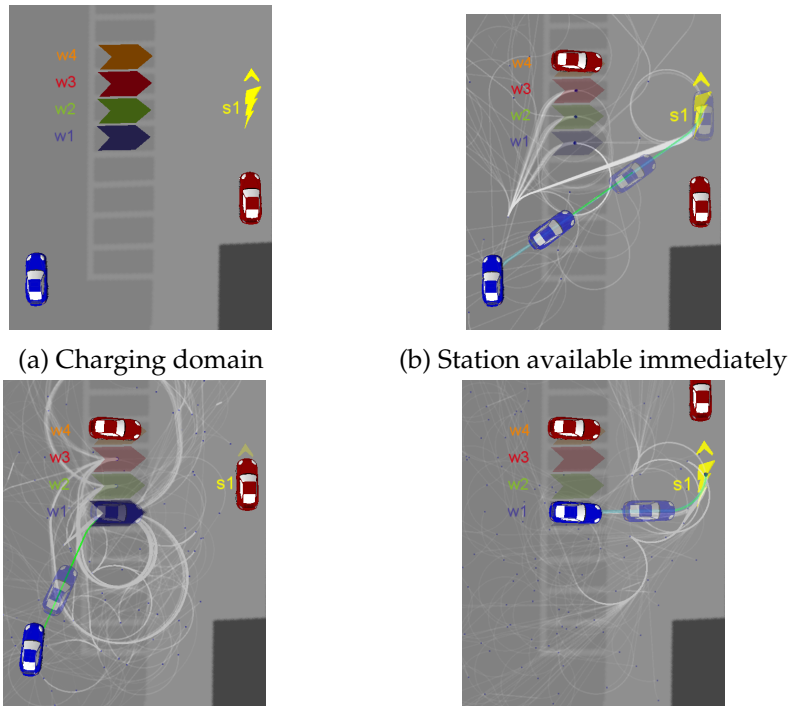


Figure 4.2: Process graph for $s_1 + (w_1 + w_2 + w_3 + w_4) \cdot\ s_1$.

Fig. 4.3b shows a situation when the charging station is immediately available; the electric car therefore directly heads towards s_1 using the optimal trajectory shown in blue. Note that branches in Kripke structure also lead to the empty parking lots, but they are not preferred. On the other hand, when the station is occupied as shown in Fig. 4.3c, the Kripke structure only contains feasible trajectories that take the car into the parking lots. The cost tuple of these trajectories that end in the parking lot is $(1, *)$ since they are all one step away from reaching the accepting state s_1 in the process graph (see Fig. 4.2). As soon as the charging station becomes available, the algorithm obtains a trajectory that reaches the accepting state as shown in Fig. 4.3d. This example was run



(a) Charging domain

(b) Station available immediately

(c) Station occupied

(d) Station freed

Figure 4.3: Charging specification

on line and a first solution was computed in 0.8 sec.

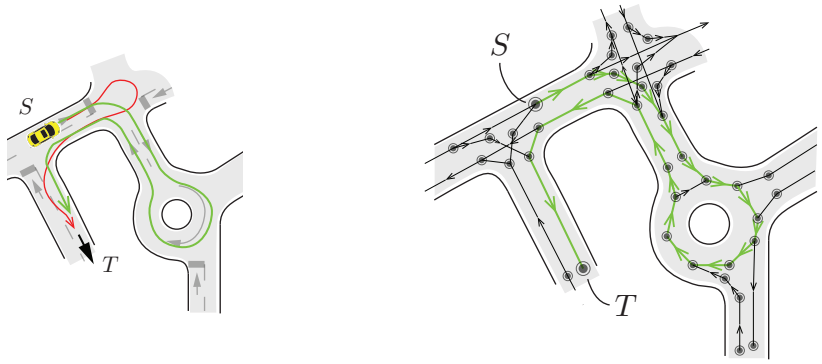
4.3.2 *Route Planner*

In this section, we provide another application, a *route planner*, which receives requests for pick-up and drop-off from a number of locations at the same time. The autonomous vehicle schedules these tasks using a process algebra specification which ensures that all requests are served optimally. We consider cases when these requests can be satisfied simultaneously, i.e., passengers share the autonomous car during transit. The autonomous vehicle can also include a charging maneuver in addition to these pick-ups and drop-offs. We assume that a passenger is not dropped off at a location that is not her destination.

Algorithm 4.2: $\texttt{permute}(s, k)$

```
1 if capacity(s) ≤ κ and is_ordered(s) then
2     Φ ← ∅;
3     for k ≤ i ≤ 2n do
4         swap(s_k, s_i);
5         Φ ← Φ + (s_k · (permute(s, k + 1)));
6         swap(s_k, s_i);
7     return Φ;
```

We first discuss an algorithm to automatically construct process algebra specifications for n pick-ups $\{p_1, p_2, \ldots, p_n\}$ and drop-offs $\{d_1, d_2, \ldots, d_n\}$. Given a capacity κ, we require that (i) for any subsequence, the difference in the number of pick-ups and number of drop-offs is never larger than κ and, (ii) the index of p_i, which we denote by p_i itself, is always smaller than the index of d_i, i.e., $p_i < d_i$ for all $1 \leq i \leq n$. Valid traces for the n request problem are simply all permutations of the set $\{p_1, \ldots, p_n, d_1, \ldots, d_n\}$ that satisfy these two conditions.



(a) (b)

Figure 4.4: Kripke structure for the route planner: Fig. 4.4a shows the optimal trajectory from S to T in red, while the green trajectory follows road rules. Fig. 4.4b shows how the generated Kripke structure incorporates these rules.

Listing all permutations with the alternative operator will however give a PA term with a large number of sub-terms. The process graph for such an expression is not computationally efficient for Alg. 4.1 since it has a large branching factor, roughly $\mathcal{O}(e^{\mathcal{O}(n \log n)})$, with all nodes at a depth of one. We instead use a recursive algorithm that finds all permutations which satisfy constraints (i) and (ii) and also automatically finds an efficient compression for the specification. For a sequence s, we denote the i^{th} element by s_i. Note that both (i) and (ii) can be efficiently checked by one traversal along the sequence.

The algorithm starts with $s = p_1.d_1.p_2.d_2 \ldots .p_n.d_n$. The procedure `capacity` calculates the capacity of the vehicle required for the current sequence s while the procedure `is_ordered` checks constraint (ii). Alg. 4.2 depicts a procedure to recursively compute a process algebra term for n pick-ups and drop-offs which can be obtained by executing `permute`$(s, 1)$.

Examples

The experiments in this section are performed on a pre-generated map of a part of the National University of Singapore's campus. We manually construct a Kripke structure on the map in such a way that road-lanes and directions of one-way roads are incorporated by including the appropriate directed edges in the Kripke structure. We have written a Flash based utility where the user can easily generate such a Kripke structure given a road map, as shown in Fig. 4.4. Note again that traffic in Singapore drives on left side of the road.

***Case 1*:** Fig. 4.5 shows the optimal plan with two pick-up and drop-off requests. The process algebra specification for this task with a capacity, $\kappa = 2$, is

$$\Phi_2 = p_1.(d_1.p_2.d_2 + p_2.(d_1.d_2 + d_2.d_1))p_2.(d_2.p_1.d_1 + p_1.(d_1.d_2 + d_2.d_1)). \tag{4.3}$$

Based on the locations of p_1, p_2, d_1 and d_2, the algorithm returns a trajectory that satisfies the task $p_1.p_2.d_1.d_2$. It therefore outputs a trajectory where p_2 shares the car with p_1 while the vehicle drops off p_1 at her destination d_1.

***Case 2*:** We now incorporate the charging specification for pick-ups and drop-offs. Consider m charging locations, $c_1, c_2, \ldots, c_m$, we can again find permutations of the extended set

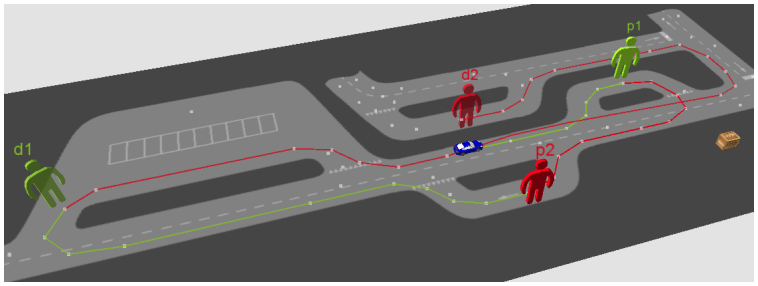
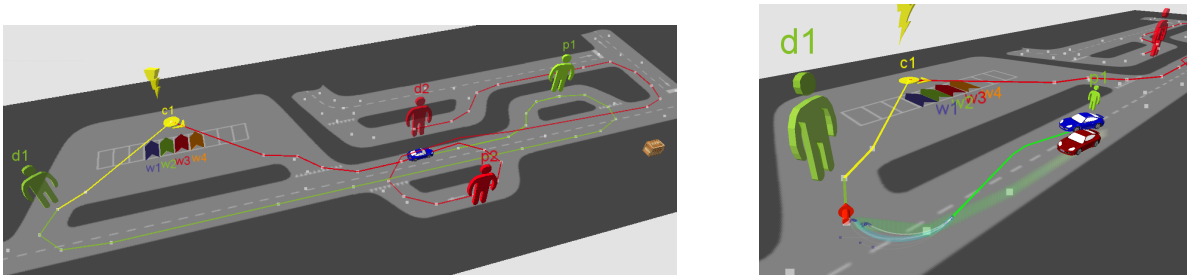


Figure 4.5: Two requests: The autonomous vehicle first picks up p_1, shown using the green trajectory. p_2 shares the car with p_1 while the vehicle goes to destination d_1. The vehicle then proceeds to d_2 to drop-off p_2. Note that all trajectories satisfy rules of the road.



(a) (b)

Figure 4.6: Two requests with charging : Fig. 4.6a shows the trajectory generated for $\Phi_{2,c}$. Fig. 4.6b shows randomly sampled states of the Kripke structure in order to avoid obstacles.

$\{p_1, \ldots, p_n, d_1, \ldots, d_n, c_1, \ldots, c_m\}$ such that, (i) and (ii) are true from Case 1, and, (iii) vehicle charges only when it is empty. The third condition can be checked by ensuring that before every c_i in the current string s, there are equal number of pick-ups and drop-offs, i.e., the vehicle is empty. This is checked by a procedure `charging_constraint` which is executed along with `is_ordered` on Line 1 of Alg. 4.2 and returns true if (iii) is satisfied. Each node in the product graph stores the battery level $b(z)$ and the cost as a 3-tuple, i.e., $\texttt{cost}(z) = (\texttt{cost}_B(z), \texttt{cost}_P(z), \texttt{cost}_K(z))$ where $\texttt{cost}_P$ and $\texttt{cost}_\mathcal{K}$ are the same as the `calculate_cost` procedure in Sec. 4.2. If $z = (s, q)$ is a charging station, $\texttt{cost}_B(z) = 0$ while it is $\max\{b_{min} - b(z), 0\}$ otherwise. b_{min} is some minimum battery threshold and $b(z)$ decreases linearly with the distance traveled since the last charging station.

Fig. 4.6 shows an example with two pick-up and drop-off requests and one charging station. The specification for this problem is

$$\Phi_{2,c} = p_1.(d_1.(\epsilon + \Phi_c).p_2.d_2 + p_2.(d_1.d_2 + d_2.d_1).(\epsilon + \Phi_c)) + \\ p_2.(d_2.(\epsilon + \Phi_c).p_1.d_1 + p_1.(d_1.d_2 + d_2.d_1).(\epsilon + \Phi_c)).$$

where we denote the empty atomic action by ϵ. Note that we have replaced c_1 by the charging specification Φ_c from Eqn. (4.2). This enables behaviors similar to those shown in Fig. 4.3 in case the charging station is occupied. Locations of p_1, p_2, d_1, d_2 are same as Case 1 along with a charging station. When the vehicle starts with a depleted battery, as opposed to Case 1, it picks up p_1 but directly drives to d_1 to drop her off. After swapping batteries at c_1, it picks up p_2 and goes to d_2.

CHAPTER 5

Multi-agent Systems

This chapter focuses on control synthesis for robots interacting with external agents and formulates game theoretic approaches for them. This problem is modeled as a non-zero sum differential game where agents fulfill their respective task specifications while satisfying a set of LTL safety rules. We also consider generic multi-robot motion planning problems and decoupled task specifications. The problems are formulated as an open-loop non-cooperative differential game. In both these approaches, we utilize ideas from sampling-based algorithms and model checking to converge to well-defined game theoretic notions of equilibria that enable various behaviors.

As autonomous agents transition from experimental projects into viable means of urban transportation, they increasingly have to share infrastructure with other drivers, both human and autonomous. It is then imperative to ensure that they interact with human-driven vehicles according to the rules of driving and safety on the road. Behaviors such as merging into busy lanes or handling stop-signs at cross-roads, which are typically enforced using right-of-way or even communication, are easy for human drivers but are arguably much harder for autonomous agents [BWF+13].

This chapter discusses a formulation to compute equilibria for two-player differential games where players try to accomplish a task specification while satisfying safety rules expressed using temporal logic. It builds upon the concepts develop in Chap. 3 and formulate the interaction between an autonomous agent and its environment as a non-zero sum differential game; both the robot and the environment minimize the *level of unsafety* of a trajectory with respect to safety rules expressed using LTL formulas. We abstract a continuous-time dynamical system with differential constraints into finite Kripke structures and employ model checking techniques to quantify the level of unsafety. We describe an algorithm to compute the open-loop Stackelberg equilibrium (OLS) of the differential game on these Kripke structures. It can be shown that the algorithm converges to the OLS as the number of samples in the Kripke structure goes to infinity.

In a related direction, we also consider the problem of multi-robot motion planning with decoupled task specifications and model it as a non-cooperative game. In these approaches, we look for the non-cooperative Nash equilibrium that characterizes the stable solutions among inherently self-interested players where none can benefit from unilateral deviations. Again, using ideas from sampling-based motion planning algorithms, we will be able to construct efficient data structures on which we can compute the Nash equilibrium iteratively. In order words, each robot sequentially updates its response based on the current solution in such a way that it fulfills the task specification. We will show the resulting "iterative better response" algorithm converges to the open-loop non-cooperative Nash equilibrium in the limit.

5.1 Problem Formulation

This section formalizes the problem considered in this paper and models it as a nonzero-sum differential game between two players, the robot (**R**) and the environment (**E**). Both **R** and **E** minimize their respective cost functions while satisfying their task specifications. In the sequel, for clarity of presentation, we assume that the dynamics of both players is the same. Also, atomic propositions, Π, and safety rules Ψ, are same for both players. The formulation is however general and also applies to cases where players with different dynamics minimize cost for different sets of safety rules.

We consider the task specification defined as "traveling from an initial state to a final goal set without colliding with any obstacles or other agent". In this context, define compact sets $\mathcal{X}_{r,obs}, \mathcal{X}_{r,G} \subset \mathcal{X}_r$ and $\mathcal{X}_{e,obs}, \mathcal{X}_{r,G} \subset \mathcal{X}_e$. A trajectory of the game in Eqn. (2.1) x, can be projected to obtain trajectories x_r, x_e of players **R**, **E** respectively. x_r is said to satisfy the task specification Φ_r if for some $T_r \in \mathbb{R}_{\geq 0}$; $x_r(0) = x_{r,0}$, $x_r(T_r) \in \mathcal{X}_{r,G}$, $x_r(t) \notin \mathcal{X}_{r,obs}$ and $\|x_r(t) - x_e(t)\|_2 > c$ for all $t \in [0, T_r]$ for a fixed constant c. Similarly, x_e is said to satisfy Φ_e if for some $T_e \in \mathbb{R}_{\geq 0}$; $x_e(0) = x_{e,0}$, $x_e(T_e) \in \mathcal{X}_{e,G}$, $x_e(t) \notin \mathcal{X}_{e,obs}$ and $\|x_r(t) - x_e(t)\|_2 > c$ for all $t \in [0, T_e]$.

5.1.1 *Normalized Level of Unsafety*

The level of unsafety for a set of safety rules in Def. 3.1 is a cost tuple and is compared using the lexicographical ordering. For convenience, we normalize it and convert it into a scalar cost function $\overline{\lambda}(w, \Psi)$ by setting $\varpi(A_{ij}) = 2^{-i}$. Specifically, define

$$\overline{\lambda}(w, A) = \begin{cases} 1 - \exp(-\lambda(w, A)) & \text{if } \lambda(w, A) < \infty \\ 1 & \text{otherwise;} \end{cases}$$

for any word w and $\overline{\lambda}(w, \Psi) = \sum_{i=0}^{n} \sum_j \overline{\lambda}(w, A_{ij})$ with $\varpi(A_{ij}) = 2^{-i}$. It is easy to see that given two words w_1, w_2 and a set of safety rules $\Psi = \{A_{ij}\}$ such that $\lambda(w_1, \Psi) \leq \lambda(w_2, \Psi)$, we also have $\overline{\lambda}(w_1, \Psi) \leq \overline{\lambda}(w_2, \Psi)$.

Let us now define a continuous version of the level of unsafety. For a continuous trajectory $x : [0, T] \to \mathcal{X}$, if $D(x) = \{t_1, t_2, \ldots, t_n\}$, then the minimizing index set in Def. 3.1 for $w(x)$, say $I^* = \{t_{i_1}, \ldots, t_{i_k}\}$, is a subset of $D(x)$. The cost of violating a rule A is

$$\lambda_c(x, A) = \sum_{j=1}^{k} \int_{t_{i_j}}^{t_{i_j+1}} \varpi(A)\, dt \triangleq \int_0^T g_A(x(t))dt, \tag{5.1}$$

where the cost function $g_A(x)$ is defined as,

$$g_A(x(t)) = \begin{cases} \varpi(A) & \text{if } t \in [t_{i_j}, t_{i_j+1}) \text{ for some } t_{i_j} \in I^* \\ 0 & \text{else.} \end{cases}$$

$g_A(x(\cdot))$ is differentiable everywhere except on I^*. We now construct the normalized continuous level of safety, call it $\overline{\lambda}_c(x, \Psi)$, in a similar fashion as $\overline{\lambda}(w, \Psi)$. Note that for a trace-inclusive Kripke structure, we have $\overline{\lambda}_c(x, \Psi) = \overline{\lambda}(\rho, \Psi)$ for any trace ρ and its corresponding trajectory x.

5.1.2 *Problem Statement*

Sec. 5.1.1 motivates the form of cost functions for the robot and the environment. For task specifications Φ_r, Φ_e, and a set of safety rules, $\Psi = \{\Psi_1, \ldots, \Psi_n\}$, define the cost function of $\mathbf{R}$ as

$$J_r(x_0, u_r, u_e) = \overline{\lambda}_c(x_r, \Psi) + 2^{-(n+1)}\ T_r, \tag{5.2}$$

where $T_r = \inf\{t : x_r(t) \in \mathcal{X}_{r,G}\}$ and $u_r : [0, T_r] \to \mathcal{U}_r$. Similarly, let $T_e = \inf\{t : x_e(t) \in \mathcal{X}_{e,G}\}$ and $u_e : [0, T_e] \to \mathcal{U}_e$. Define the cost function of $\mathbf{E}$ to be

$$J_e(x_0, u_r, u_e) = \overline{\lambda}_c(x_e, \Psi) + 2^{-(n+1)}\ T_e. \tag{5.3}$$

Note that J_r is really the normalized, scalar form of the lexicographic cost tuple

$$(\lambda_c(x_r, \Psi_1), \ldots, \lambda_c(x_r, \Psi_n), T_r),$$

i.e., players first minimize the level of unsafety of the trajectory with respect to safety rules and then prioritize reaching the goal set as quickly as possible.

Problem 5.1. *Given task specifications Φ_r, Φ_e, a set of prioritized safety rules Ψ, for dynamics described by Eqn. (2.1), find a control strategy, $u_r^* : [0, T_r] \to \mathcal{U}_r$ where $T_r = \inf\{t : x_r(t) \in \mathcal{X}_{r,G}\}$, such that:*

1. *Trajectories x_r, x_e satisfy tasks Φ_r, Φ_e respectively, and*
2. *Among all trajectories x_r, x_e respectively, that satisfy 1, (u_r^*, u_e^*), where $u_e^* = BR(u_r^*)$, is the open-loop Stackelberg equilibrium of the differential game with cost functions J_r, J_e, respectively.*

Note that the necessary conditions for the OLS equilibrium are obtained by minimization of the Hamiltonian if the dynamics and running cost are differentiable [CCJ72]. In our case, even though we are guaranteed that $f \in C^1$, the running cost, shown in Eqn. (5.1), is discontinuous. We can however still characterize the OLS equilibrium using viscosity solutions to optimal control problems with discontinuous running cost.

Theorem 5.2 (Thm. 4.1 in [Sor00]). *Consider a dynamical system given by $\dot{x}(t) = f(x(t), u(t))$ with $x(0) = x_0$ where $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ belongs to C^1 and $u(\cdot)$ belongs to the class of relaxed controls, i.e.,*

$$u \in \mathcal{U}_{relax} = \{u(\cdot) : u : \mathbb{R}_{\geq 0} \to P(\mathcal{U}) \text{ measurable}\}$$

where $P(\mathcal{U})$ is the set of Radon measures over the compact set $\mathcal{U} \subset \mathbb{R}^m$. For cost functionals of the form, $J(x, t) = \int_0^t g(x(s))\, ds$ where $g(x(s))$ is bounded but possibility discontinuous on a set of measure zero, there exists a unique viscosity solution if the dynamics is small time locally controllable.

Thm. 5.2 can be used to show that the solution of Prob. 5.1 is unique if the dynamical system in Eqn. (2.1) is small-time locally controllable and if the initial conditions x_0 and the goal regions $\mathcal{X}_{r,G}, \mathcal{X}_{e,G}$ are such that optimal trajectories spend only a measure zero time at the discontinuities of the labeling function $\mathcal{L}_c$. Let us also note that under the above assumptions, cost functions in Eqn. (5.2) and Eqn. (5.3) are continuous in a neighborhood of the optimal trajectory.

5.2 Stackelberg Games: Algorithm and Analysis

This section describes an algorithm to incrementally construct the product automaton (see Def. 3.9 in Sec. 3.3.1). Ideas from sampling-based motion-planning algorithms are used to construct Kripke structures for **R** and **E**, say K_r and K_e, resp. Safety rules with priorities, expressed as finite automata are used to first construct their weighted product, i.e, $\overline{A}_\Psi = (Q_\Psi, q_{0,\Psi}, \Sigma, \delta_\Psi, F_\Psi, W_\Psi)$ (see Sec. 3.5). We then construct the weighted product automata, call them P_r, P_e, for players **R**, **E**, respectively. For convenience, let $\alpha \in \{r, e\}$. Conceptually, these product automata are used to compute traces corresponding to the Stackelberg equilibrium incrementally. The algorithm however maintains two product automata $P_{\alpha,f}$ and $P_{\alpha,b}$ for each player α, i.e., $P_\alpha = P_{\alpha,f} \cup P_{\alpha,b}$. This enables some computational benefits and helps us quickly compute best responses and corresponding costs.

Let $Q_{P_{\alpha,f}}$ and $Q_{P_{\alpha,b}}$ be the set of states of $P_{\alpha,f}$ and $P_{\alpha,b}$, resp. $Q_{P_{\alpha,f}}$ is initialized with $z_{0,\alpha} = (s_{0,\alpha}, q_{0,\Psi})$ while $Q_{P_{\alpha,b}}$ is initialized with $z_{g,\alpha} = (s_{g,\alpha}, q_{g,\alpha})$ for some $s_{g,\alpha} \in X_{\alpha,G}$ and $q_{g,\alpha} \in F_\Psi$. Each sampled vertex z_α is added to both $Q_{P_{\alpha,f}}$ and $Q_{P_{\alpha,b}}$; however, transitions are made towards z_α, i.e., $(*, z_\alpha)$ in $\delta_{P_{\alpha,f}}$ while they are made away from z_α, i.e., $(z_\alpha, *)$ in $\delta_{P_{\alpha,b}}$. Each vertex maintains two costs, $J^d_{\alpha,f}$ and $J^d_{\alpha,b}$, where $J^d_{\alpha,f}$ is the least weight of a trace reaching z_α from $z_{0,\alpha}$ while $J^d_{\alpha,b}$ is the least weight of a trace from z_α to $z_{g,\alpha}$ in $Q_{P_{\alpha,b}}$. Let $J^d_\alpha(z_\alpha) = J^d_{\alpha,f}(z_\alpha) + J^d_{\alpha,b}(z_\alpha)$. The preliminary procedures below are written for $P_{\alpha,f}$, they are analogous for $P_{\alpha,b}$.

5.2.1 *Procedures*

1. *Sampling:* The sampling procedure `sample` : $\mathbb{N} \to \mathcal{X}_\alpha$ returns independent, identically distributed samples from a distribution supported over $\mathcal{X}_\alpha \setminus \mathcal{X}_{\alpha,obs}$.
2. *Steer:* Given two samples $z_1 = (s_1, q_1)$ and $z_2 = (s_2, q_2)$ in $P_{\alpha,f}$, the `steer` procedure returns the minimum cost of going from z_1 to z_2.

   It computes a time $T \in \mathbb{R}_{\geq 0}$ and trajectories $x_\alpha : [0, T] \to \mathcal{X}_\alpha$, $u_\alpha : [0, T] \to \mathcal{U}_\alpha$, such that, x_α, u_α satisfy the dynamics $\dot{x}_\alpha = f_\alpha(x_\alpha, u_\alpha)$ with $x(0) = s_1$, $x(T) = s_2$ and x_α is trace-inclusive and minimizes the cost function $J_\alpha(s_1, u_\alpha, \cdot)$. It returns $J^d_{\alpha,f}(z_1, z_2) = J_\alpha(s_1, u_\alpha, \cdot)$ and returns false if such a trajectory is infeasible or if $(q_1, q_2) \notin \delta_\Psi$. Examples of how the `steer` procedure can be constructed for certain dynamics can be found, for example, in [CL07, Dub57].
3. *Near vertices:* For a state $s \in \mathcal{X}_\alpha$, let $\mathcal{X}_{\alpha,near}(s) \subset \mathcal{X}_\alpha \cap K_\alpha$ consist of the closest $k \log n$ ($k > 2$) samples according to the cost metric J_α used in the `steer` procedure. It is thus a subset of the reachable space of the dynamical system and can be computed efficiently using the Ball-Box Theorem (see [KF13] for details). The `near` procedure returns:

$$\texttt{near}(s) = \left\{ (s', q) \ : \ s' \in \mathcal{X}_{\alpha,near}(s), (s', q) \in Q_{P_{\alpha,f}} \right\}.$$

4. *Connect:* Given a vertex $z = (s, q)$, the `connect` procedure computes a vertex z_p such that

$$z_p = \arg \min_{z' \in \texttt{near}(s)} J^d_{\alpha,f}(z') + \texttt{steer}(z', z).$$

   It updates the transition function as $\delta_{P_{\alpha,f}} = \delta_{P_{\alpha,f}} \setminus \{(*, z)\} \cup \{(z_p, z)\}$, i.e., it removes all transitions to z and adds the one that minimizes the cost of z. The weighing function is updated to be $W_{P_{\alpha,f}}(z_p, z) = \texttt{steer}(z_p, z)$.

Algorithm 5.1: $\texttt{extend}_\alpha$

$s \leftarrow \texttt{sample}$;
$Z, Z_{rewire} \leftarrow \emptyset$;
foreach $q \in Q_\Psi$ **do**
  $z \leftarrow (s, q)$;
  foreach $z' \in \texttt{near}(s)$ **do**
    if $\texttt{steer}(z', z)$ **then**
      $Z \leftarrow Z \cup z$;
foreach $z \in Z$ **do**
  $Q_{P_{\alpha,f}} \leftarrow Q_{P_{\alpha,f}} \cup \{z\}$;
  $Q_{P_{\alpha,b}} \leftarrow Q_{P_{\alpha,b}} \cup \{z\}$;
  $\texttt{connect}(z)$;
  if $s \in X_{\alpha,G}$ *and* $q \in F_\Psi$ **then**
    $F_{P_{\alpha,f}} \leftarrow F_{P_{\alpha,f}} \cup \{z\}$;
  $\texttt{update}_\alpha(z)$;
  $Z_{rewire} \leftarrow Z_{rewire} \cup \texttt{rewire}(z)$;
return Z, Z_{rewire}

Algorithm 5.2: `construct_product`

foreach $\alpha \in \{r, e\}$ **do**
  $Q_{P_{\alpha,f}} \leftarrow \{z_{0,\alpha}\}$;
  $Q_{P_{\alpha,b}} \leftarrow \{z_{g,\alpha}\}$;
  $\delta_{P_\alpha} \leftarrow \emptyset$;
for $i \leq n$ **do**
  $Z_r, Z_{r,rewire} \leftarrow \texttt{extend}_r$;
  $Z_e, Z_{e,rewire} \leftarrow \texttt{extend}_e$;
  $\texttt{update\_best\_response\_queue}(Z_{e,rewire})$;
  $z_r^* \leftarrow \texttt{update\_best\_vertex}$;
  $\texttt{delete\_collisions}(\texttt{trace}(z_r^*))$;
return z_r^*, P_r, P_e

Figure 5.1

5. *Update:* Given $z \in Q_{P_{\alpha,f}}$, the $\texttt{update}_\alpha$ procedure updates the cost as, $J^d_{\alpha,f}(z) = J^d_{\alpha,f}(z_p) + \texttt{steer}(z_p, z)$ for $(z_p, z) \in \delta_{P_{\alpha,f}}$.
6. *Descendants* $\texttt{descendents}_\alpha(z_\alpha)$ are all vertices in $Q_{P_{\alpha,f}}$ that are reachable from z_α. $\texttt{ancestors}_\alpha(z_\alpha)$ in $Q_{P_{\alpha,b}}$ are defined similarly.
7. *Rewiring* For a vertex $z = (s, q)$, if $J^d_{\alpha,f}(z') > J^d_{\alpha,f}(z) + \texttt{steer}(z, z')$ for some z' in the set $\texttt{near}(s)$, the `rewire` procedure executes the `connect` procedure on z'. For every such z', which requires rewiring, we call $\texttt{update}_\alpha$ for all $z'' \in \texttt{descendents}_\alpha(z')$. Let $Z_{\texttt{rewire}}$ be the set of all the vertices which are rewired.
8. *Compute trace:* For $z_\alpha \in Q_{P_{\alpha,f}}$, the procedure `trace` returns ρ_α which is the unique trace from $z_{0,\alpha}$ to z_α in $P_{\alpha,f}$ concatenated with the unique trace from z_α in $P_{\alpha,b}$ to $z_{g,\alpha}$. Since the Kripke structures are trace-inclusive, we can also construct the continuous trajectory $\texttt{traj}(\rho_\alpha)$ from ρ_α using the `steer` procedure.
9. *Calculate best response:* Given $z_r \in Q_{P_{r,f}}$ as input, the procedure `best_response` returns a vertex $\texttt{BR}(z_r) \in Q_{P_{e,f}}$ such that $\texttt{trace}(\texttt{BR}(z_r))$ is the best response of **E** if **R** executes $\texttt{trace}(z_r)$. We maintain a priority queue for the set $\{J^d_e(z_e) : z_e \in Q_{P_{e,f}}\}$, this enables us to search for the best response of any given $z_r \in Q_{P_{r,f}}$ efficiently.
10. *Update best response:* For every vertex z_e that is rewired, the cost, $J^d_{e,f}(z'_e)$ changes for all $z'_e \in \texttt{descendents}_e(z_e)$ (and $J^d_{e,b}$ changes for all $\texttt{ancestors}_e(z_e)$). The new costs, J^d_α are computed using the `update_best_response_queue` procedure.
11. *Update best vertex* The algorithm incrementally maintains the best vertex z_r^* in $Q_{P_{r,f}}$ that minimizes the cost of **R**. For a newly sampled vertex $z_r \in Q_{P_{r,f}}$, if **R** executes $\rho_r = \texttt{trace}(z_r)$ and **E** executes $\rho_e = \texttt{trace}(\texttt{BR}(z_r))$, this procedure evaluates the cost, i.e, $J_r(z_{0,r}, \rho_r, \rho_e)$ and updates the best vertex z_r^* if the trajectories do not collide.

12. *Delete collisions* Given a trace ρ_r, this procedure removes all vertices z_e from $Q_{P_{e,f}}$ and their descendants whose trajectories collide with $\texttt{traj}(\rho_r)$ and then calls the `update_best_response_queue` procedure.

The tuple $(\rho^*_{P_r}, \rho^*_{P_e})$, where $\rho^*_{P_r} = \texttt{trace}(z^*_r)$ and $\rho^*_{P_e} = \texttt{trace}(\texttt{BR}(z^*_r))$ is therefore the open-loop Stackelberg equilibrium for the game played on discrete product automata. The projections of these traces onto individual Kripke structure are ρ^*_r, ρ^*_e respectively. Trajectories $x^*_r = \texttt{traj}(\rho^*_r)$ and $x^*_e = \texttt{traj}(\rho^*_e)$ then are the continuous-time trajectories returned by the algorithm. Alg. 5.2 presents the complete procedure for computing $(\rho^*_{P_r}, \rho^*_{P_e})$.

5.2.2 *Analysis*

This section provides an analysis of the algorithm presented in Sec. 5.2. We first show how the two product automata differ. Roughly, the automaton $P_r = P_{r,f} \cup P_{r,b}$ of player **R** is such that $J^d_{\alpha,f}$ for every vertex is the best cost from $z_{0,r}$. On the other hand, the automaton $P_e = P_{e,f} \cup P_{e,b}$ of player **E** consists of best responses and hence the cost of a vertex $z_e \in P_e$ can be larger than its optimal cost, i.e., without considering the trajectory of **R**. We omit most proofs in the interest of space. Technical arguments presented in this section are similar to those in [KF11b, KF11a, CCT+13].

Asymptotic optimality and Probabilistic completeness

Theorem 5.3 (Asymptotic optimality of P_r). *For any vertex $z_r = (s_r, q_r)$ of Q_{P_r}, let $x_r : [0, T] \to X_r$ be the optimal trajectory that minimizes J_r such that $x_r(0) = x_{0,r}$ and $x_r(T) = s_r$. Then the cost $J^d_{r,f}(z_r)$ in Alg. 5.2 converges to $J_r(x_r)$ in the limit almost surely, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} J^d_{r,f}(z_r) = J_r(x_r)\}\right) = 1.$$

The above theorem is an application of asymptotic optimality of the RRT* algorithm [KF11b]. As a particular instance of the above theorem, the best vertex z^*_r returned by Alg. 5.2 also has the optimal cost. On the other hand, the continuity of the cost function J_r translates to **E**'s Kripke structure as follows.

Lemma 5.4. *For $z_e = (s_e, q_e) \in Q_{P_e}$ if $x_e : [0, T] \to \mathcal{X}_e$ is the optimal trajectory that minimizes J_e such that $x_e(0) = x_{0,e}$ and $x_e(T) = s_e$, then the cost $J^d_{e,f}(z_e)$ is at least as much as $J_e(x_e)$ almost surely, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} J^d_{e,f}(z_e) \geq J_e(x_e)\}\right) = 1.$$

The proof of this is an immediate consequence of the fact that the `delete_collisions` procedure removes transitions from P_e that collide with the current best trajectory of **R**. It thus contains fewer transitions than the optimal RRT* tree.

Alg. 5.2 inherits probabilistic completeness from the RRT* algorithm (see Thm. 23 in [KF11a]), i.e., it returns trajectories $x^*_{r,n}$ and $x^*_{e,n}$ such that they converge to the open-loop Stackelberg equilibrium of Prob. 5.1, with probability one, as the number of samples approaches infinity. In addition to this, we can also show the nature of convergence as follows.

Theorem 5.5. *The trajectories returned by Alg. 5.2 after n iterations, $x^*_{r,n}$ and $x^*_{e,n}$, converge to the solution of Prob. 5.1 in the bounded variation norm sense, $\mathbb{P}\left(\{\lim_{n\to\infty} \|x^*_{r,n} - x^*_r\|_{BV} = 0\}\right) = 1$ and similarly for $x^*_{e,n}$.*

Proof. The proof of this theorem is very similar to that of Thm. 16 in [CCT+13] and is hence omitted. Roughly, since the Stackelberg equilibrium exists and is unique from Thm. 5.2, we can show that there exists traces ρ_r of K_r and ρ_e of K_e, that lie in the neighborhood of x_r^* and x_e^*. The corresponding continuous trajectory then satisfies the claim. ■

Now, it easily follows that the cost of the trajectories returned by the algorithm converges to the optimal cost.

Corollary 5.6 (Asymptotic optimality). *The costs, $J_r(x_{r,n}^*)$ and $J_e(x_{r,e}^*)$ converge to the optimal costs in the limit with probability one, i.e., $\mathbb{P}\left(\{\lim_{n\to\infty} J_r(x_{r,n}^*) = J_r(x_r^*)\}\right) = 1$, and similarly for $x_{e,n}^*$.*

Let us briefly discuss the computational complexity of Alg. 5.2. If m is the number of states in $\overline{A}_\Psi$, `near`(s) returns $\mathcal{O}(m \log n)$ vertices in expectation. The complexity of running `connect` these vertices is $\mathcal{O}(m^2 \log^2 n)$. The `rewire` procedure considers an $\mathcal{O}(m \log n)$ neighbors and updates the cost of their descendants. It can be shown that the complexity of such an update is $\mathcal{O}(m^2 \log^2 n)$ in expectation. Similarly, we see that the `update_best_response_queue` procedure also updates $\mathcal{O}(m^2 \log^2 n)$ vertices in $Q_{P_{e,f}}$. The complexity of `update_best_vertex` is $\mathcal{O}(m \log n)$ if we maintain a priority queue of vertices in $Q_{P_{r,f}}$ with their best responses.

It can be shown that the `delete_collisions` procedure deletes descendants of nodes in an area that scales as $\mathcal{O}(\log n / n)$. Also, the height of a random node in the random Kripke structure concentrates around its expectation which is $\mathcal{O}((n^2/\log n)^{1/d})$ [FSS11]. Thus, the `delete_collisions` procedure removes only a small number of descendants beyond this height; the number of nodes deleted per iteration in fact goes to zero in the limit as $n \to \infty$.

The total expected amortized complexity of Alg. 5.2 is therefore $\mathcal{O}(m^2 \log^2 n)$ per iteration.

5.3 Non-cooperative Games

Let us now consider general n-player non-cooperative Nash equilibria. In this section, we will only consider generic motion-planning tasks without any temporal specifications. We construct algorithms that converge to the n-player non-cooperative Nash equilibrium and the Pareto optimum using ideas from sampling-based motion planning algorithms. Note that these algorithms can be easily modified to incorporate temporal specifications such as safety and eventually reaching a goal region, as shown in [ZOCF14a, ZOCF14b].

5.3.1 *Problem Setup*

As considered in Sec. 2.4, we consider a team of robots, say $[N] = \{1, \ldots, N\}$, with individual dynamics governed by

$$\dot{x}_i = f_i(x_i(t), u_i(t)), \quad x_i(0) = x_{i,0},$$

where $x_i(t) \in \mathcal{X}_i \subseteq \mathbb{R}^{d_i}$ and $u_i(t) \in \mathcal{U}_i \subset \mathbb{R}^{m_i}$ for all $t \in \mathbb{R}_{\geq 0}$. Given trajectories $\{x_1, x_2, \ldots, x_n\}$ of all agents where each x_k belongs to some computable set, say $\texttt{feasible}_k$, we compute the Nash equilibrium of the game when each player minimizes a cost function $J_k(x_k, u_k)$. For example, $\texttt{feasible}_k$ consists of all trajectories of agent k that do not collide with other agents and reach some goal region $\mathcal{X}_{goal,k}$.

5.3.2 *Algorithm Overview*

Let us define some preliminary procedures that will be used to construct the algorithm. Most procedures are borrowed from Rapidly-exploring Random Graphs [KF11b] and are similar to the ones in Sec. 3.3.2 and Sec. 4.2. Let G_k^i be the Kripke structure maintained by an agent k at iteration i; note that we can think of the Kripke structure as just a finite transition system if we do not have temporal specifications, as this exposition assumes.

The `sample`, `steer` and `near` procedures are the same as previous chapters. Let us elaborate on the new ones below.

1. The `extend_basic` procedure calls the `steer` procedure on all vertices returned by the `near` procedure, i.e., all nodes within a distance $\mathcal{O}((\log n/n)^{1/d})$ try to steer towards the new sample $z_{k,new}$. An edge is added to G_k^i if it returns true.
2. The `better_response` procedure works as follows. Let $\overline{\rho} = \{\rho_1^i, \ldots, \rho_N^i\}$ be the trajectories of all the agents according to their respective Kripke structures G_k^i. We understand that if $V_k^i \cap \mathcal{X}_{k,goal} = \emptyset$ for some agent k, ρ_k is the trivial trajectory that ends as $x_{k,init}$ and has zero duration T_r.

   The `better_response` procedure on an agent k then considers all the trajectories $R = \{\rho_1^i, \ldots, \rho_N^i\} \setminus \{\rho_k^i\}$ and finds a trajectory that belongs to the set of feasible trajectories for agent k, e.g., does not collide against any trajectory from R and reaches the goal region $\mathcal{X}_{k,goal}$. Note that `better_response` for agent k at iteration i considers the updated trajectories of agents $[N]$ at iteration i as well. By convention, if `better_response` is yet to be called on an agent $k' > k$ for iteration i, we set $\rho_{k'}^i = \rho_{k'}^{i-1}$.

   As shown in Alg. 5.4, this procedure can be implemented naively, i.e., by constructing the set of all feasible trajectories on G_k^i and picking the one that minimizes the cost function $J_k(\cdot,\cdot)$ from among them. Let us note that this can be improved significantly. Specifically, each agent k maintains the set R constructed in Alg. 5.4 and uses this set in the `extend_basic` procedure to find the feasible paths directly. The `better_response` procedure then becomes similar to the `update_cost` procedure in Alg. 3.2.

5.3.3 *Analysis*

Roughly speaking, Alg. 5.3 consists of multiple "rounds". In each round i, agents $[N]$ take turns to update their Kripke structures G_k^i and construct a trajectory ρ_k^i that does not conflict with the best trajectory of other agents. In this section, we analyze the convergence properties of Alg. 5.3.

Remark 5.7. Note that instead of constructing a Kripke structure that is a tree as done in all previous chapters, we construct a directed graph here. This is due to the fact that the tree constructed in Alg. 3.1 for example, has a "rewiring" update which can lead to G_k^{i-1} not being a sub-graph of G_k^i. This property is crucial for convergence to the Nash equilibrium as we show in the analysis below.

Also, the Kripke structure G_k^i has no notion of "rewiring", i.e., we only draw transitions from vertices in V_k^{i-1} to the new sample. This ensures that G_k^i does not include any directed cycles. It is similar to the auxiliary graph constructed in Thm. 38 of [KF11b].

Algorithm 5.3: iNash

$V_k \leftarrow x_{k,init}$;
$E_k \leftarrow \emptyset$;
$G_k^0 \leftarrow (V_k, E_k)$;

$i \leftarrow 0$;
while $i < n$ **do**
  for $k \leq N$ **do**
    $z_{k,new} \leftarrow$ `sample`;
    $G_k^i \leftarrow$ `extend_basic`$(G_k^{i-1}, z_{k,new})$
  for $k \leq N$ **do**
    $\rho_k^i \leftarrow$ `better_response`(G_k^i);
  $i \leftarrow i + 1$;

Algorithm 5.4: `better_response`

$R \leftarrow \left\{\rho_1^i, \ldots, \rho_{k-1}^i, \rho_{k+1}^i, \ldots, \rho_N^i\right\}$;
$R_{feasible} \leftarrow \emptyset$;
for $\rho_k \in$ `all_simple_paths`(G_k^i) **do**
  if `is_collision_free`(ρ_k, R) *and* $\rho_k \in$ `feasible`$_k$ **then**
    $R_{feasible} \leftarrow \{\rho_k\}$
$\rho_k^* \leftarrow \arg\min \left\{\texttt{cost}(\rho_k) \mid \rho_k \in R_{feasible}\right\}$;
return ρ_k^*

Figure 5.2: iNash algorithm

Remark 5.8 (Some notation). With some abuse of notation, we define the Nash equilibrium and Pareto optimum for traces of Kripke structures G_k^i for $k \leq N$ using the same notation as Def. 2.9 and 2.10. Let R_{NE}, R_{SO} be the set of Nash equilibria and Pareto optima, respectively. We also assume in the following analysis that $R_{SO} \neq \emptyset$; in other words, the problem where the combined state is $[x_1(t), \ldots, x_N(t)]$ and the goal region is $\mathcal{X}_{goal} = \bigoplus \mathcal{X}_{k,goal}$ is feasible. Denote by $\overline{R_{NE}}$, the complement of R_{NE}, i.e., all traces or trajectories that do not belong to R_{NE}.

Note that for a trace ρ, we will refer to x, u as the state and control trajectory generated by appending the `steer` function outputs. In the following proofs, if ρ is used with a sub-script and super-script, the corresponding state and control trajectories are denoted by the same sub-script and super-script.

Also, with some more abuse of notation, we denote the cost of the game by $J(\overline{\rho})$, i.e., $J(\overline{\rho}) = \bigoplus_k J_k(x_k, u_k)$ where $\overline{\rho} = \{\rho_1, \ldots, \rho_N\}$. For convenience, we let $R_k = \texttt{feasible}_k$ and $R_{-k} = \bigoplus_{k' \neq k} \texttt{feasible}_{k'}$. Similarly, let $\overline{\rho}^i = \left\{\rho_1^i, \ldots, \rho_N^i\right\}$, i.e., the set of traces for every agent at iteration i. $\overline{\rho}_{-k}^i = \overline{\rho}^i \setminus \rho_k^i$.

Lemma 5.9 (Nash equilibrium exists). *It holds that $R_{SO} \subseteq R_{NE}$ and R_{NE} is non-empty.*

Proof. Assume $R = R_{SO} \cap \overline{R_{NE}} \neq \emptyset$. Pick some $\overline{\rho} = \{\rho_1, \ldots, \rho_N\} \in R$. Since $\overline{\rho} \notin R_{NE}$, there exists some agent k and some state and control trajectory x_k', u_k' such that $J_k(x_k', u_k') < J_k(x_k, u_k)$.

But note that a new solution $\overline{\rho'} = \left\{\rho_1, \ldots, \rho_k', \ldots, \rho_N\right\}$ is such that all the agents have feasible trajectories and the total cost, i.e., $J(\overline{\rho'}) < J(\overline{\rho})$, which contradicts the assumption that $\overline{\rho} \in R_{SO}$. Therefore $R_{SO} \subseteq R_{NE}$. Since R_{SO} is non-empty by assumption, R_{NE} is non-empty as well. ■

Before we go further, let us introduce the concepts of weak feasibility and strong feasibility.

Definition 5.10 (Weak feasibility). Given R_k, a set of traces $\overline{\rho}_{-k}$, the set of weakly feasible traces, denoted as $\texttt{weak\_feasible}_k(\overline{\rho}_{-k}) \subset R_k$, consists of all traces ρ_k s.t. there exists a sequence $(\rho_k^i)_{i \geq 0}$ with each $\rho_k^i \in R_k$ and a sequence $\delta^i \to 0$ such that

1. $\rho_k^i \to \rho_k$;

2. $B(x_k^i, \delta^i) \in \mathcal{X}_{free}$;
3. $\|x_k(t) - x_{k'}(t)\| \geq C + \delta^i$ for all $k' \neq k$ for all t.

Here C is a fixed given constant (akin to the size of the agents), x_k^i is the trajectory obtained by appending the states on the trace ρ_k^i using the `steer` function and $B(x, \delta)$ is a ball of size δ around the trajectory x.

Informally, weak infeasibility ensures that we can converge to two trajectories for agents k, k' such that the limiting trajectories never collide.

Definition 5.11 (Strong feasibility). The set of strongly feasible traces, which we denote as $\texttt{strong\_feasible}_k(\overline{\rho}_{-k}) \subset R_k$ differs from weak feasibility in only the third condition. We assume that there exists some $\delta > 0$ such that $\|x_k(t) - x_{k'}(t)\| \geq C + \delta$, for all $k' \neq k$ for all t. It follows easily that

$$\texttt{strong\_feasible}_k(\overline{\rho}_{-k}) \subset \texttt{weak\_feasible}_k(\overline{\rho}_{-k}) \subset \texttt{feasible}_k.$$

Lemma 5.12. *For any agent k, $\rho_k \in \texttt{weak\_feasible}_k(R_{-k})$ and $\epsilon > 0$, there exists an iteration $i(k)$ such that for all $i > i(k)$, there exists a trace $\hat{\rho}_k \in R_k$ such that $\|J_k(\hat{x}_k, \hat{u}_k) - J_k(x_k, u_k)\| \leq \epsilon$.*

Proof. Recall that G_k^i constructed in Alg. 5.3 is the auxiliary graph G_n constructed in the proof of Thm. 38 in [KF11b]. The proof of this lemma follows along the same lines; we also use the fact that $J_k(\cdot, \cdot)$ is continuous. ∎

Lemma 5.13. *Assume that the sequence (ρ_k^i) for $i \geq 0$ converges to some ρ_k for all $k \leq N$. Recall that $\overline{\rho} = (\rho_1, \ldots, \rho_N)$. Then there exists a time n such that for each $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho}_{-k})$, we have $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho}^i_{-k})$ for all $i > n$, i.e., once a trace becomes strongly feasible, it remains so.*

Proof. Since (ρ_k^i) converges to ρ_k, for all k, there is an $i(k)$ s.t., $\|x_k^i(t) - x_k(t)\| \leq \delta/2$ for all $i > i(k)$ and t. Note that since $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho}_{-k})$, by definition, there is a some converging sequence $(\rho_k^l) \to \rho_k$ that satisfies Def. 5.11. Thus,

$$\|x_k^l(t) - x_{k'}(t)\| > C + \delta \;\Rightarrow\; \|x_k^i(t) - x_{k'}(t)\| > C + \delta/2 \quad \forall\, k' \neq k$$

by the triangle inequality. Thus we conclude that $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho}^i_{-k})$ for all $i > n$ where $n = \max i(k)$. ∎

Let us now prove asymptotic optimality of Alg. 5.3. Let $\hat{R}$ be the set of limit points of sequences of the form $(\rho_1^i, \ldots, \rho_N^i)$ for $i \geq 0$.

Theorem 5.14. *Any limit point in $\hat{R}$ is a Nash equilibrium.*

Proof. For a limit point of $\overline{\rho}^i = (\rho_1^i, \ldots, \rho_N^i)$ there exists a sub-sequence $\kappa_k = \kappa_k^1, \ldots, \kappa_k^n \ldots$ such that $\rho_k^i \to \rho_k^*$ for $i \in \kappa_k$. Let the limit point be denoted as $\overline{\rho^*}$ and let $\kappa = \cap\, \kappa_k$. Without loss of generality, "rename" κ to $\mathbb{Z}_{\geq 0}$ and consider the resulting sub-sequence below. Pick any $\epsilon > 0$ and using Lem. 5.12, beyond some $i > n$, for any $\hat{\rho_k}^i \in \texttt{weak\_feasible}_k(\overline{\rho^*}_{-k})$ there is a $\tilde{\rho_k^i} \in \texttt{feasible}_k(\overline{\rho^*}_{-k})$ such that

$$J_k(\tilde{x_k^i}, \tilde{u_k^i}) \leq J_k(\hat{x_k^i}, \hat{u_k^i}) + \epsilon.$$

But since ρ_k^i is the best response by agent k at iteration i, there cannot not exist any $\tilde{\rho_k^i} \in \texttt{feasible}(\overline{\rho^*}_{-k})$ such that

$$J_k(x_k^*, u_k^*) \geq J_k(\tilde{x_k^*}, \tilde{u_k^*}).$$

Together, this gives that there does not exist a $\hat{\rho_k}^i \in \texttt{weak\_feasible}_k(\overline{\rho^*}_{-k})$ for $i > n$ with

$$J_k(x_k^*, u_k^*) \geq J_k(\hat{x_k^i}, \hat{u_k^i}) + \epsilon.$$

Now use the fact that $\texttt{strong\_feasible}_k \subset \texttt{weak\_feasible}_k \subset \texttt{feasible}_k$ along with Lem. 5.13 to observe that there is an $n' > n$ such that there does not exist any $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho^*}_{-k})$ and $i > n'$ such that

$$J_k(x_k^i, u_k^i) \geq J_k(x_k, u_k) + \epsilon.$$

Take the limit on i to get that there is no $\rho_k \in \texttt{strong\_feasible}_k(\overline{\rho^*}_{-k})$ with

$$J_k(x_k^*, u_k^*) \geq J_k(x_k, u_k) + \epsilon.$$

This inequality is true for any $\epsilon > 0$. It therefore implies that any $\overline{\rho^*} \in \hat{R}$ is a Nash equilibrium. ■

Theorem 5.15 (Nash equilibrium: Asymptotic optimality). *For any $\overline{\rho} = \{\rho_1, \ldots, \rho_N\} \in \hat{R}$, the cost $J_k(x_k, u_k)$ converges to some limit $c_k \geq 0$ for all $k \leq N$.*

Proof. Let us first note that after a few iterations, all agents generate non-trivial traces (i.e., of duration strictly greater than zero), indeed the probability of this not happening goes to zero exponentially quickly. We therefore assume in the sequel that this is the case.

Consider agent k at iteration i. We have $G_k^{i-1} \subseteq G_k^i$ and $\rho_k^i \in \texttt{feasible}_k$ which implies

$$J_k(x_k^i, u_k^i) \leq J_k(x_k^{i-1}, u_k^{i-1}).$$

Hence the costs form a non-increasing sequence that is lower bounded by zero and thus $J_k(x_k^i, u_k^i)$ converges to some limit, say c_k, so long as ρ_k^i converges to some limit point. ■

Note that Alg. 5.3 is probabilistically complete from Lem. 5.9, i.e., the algorithm converges to a solution in R_{NE} with probability one as the number of samples in G_k goes to infinity for all k. Also, note that each agent only transmits its trajectories to other agents at most twice per iteration, i.e., the communication complexity is simply $\mathcal{O}(2N)$ (this is reduced to $\mathcal{O}(N)$ if agents cache trajectories of other agents).

5.3.4 *Pareto optimum solution*

An algorithm to compute the social optimum is simple. We construct the product of the dynamics of each agent in Eqn. 2.3, call the new state $\bigoplus_k x_k$. The algorithm then samples the product graph $\bigoplus_k G_k$ at every iteration and uses the standard RRT* algorithm on this combined system. By virtue of Thm. 36 in [KF11b], this approach immediately converges to the Pareto optimum defined in Def. 2.10. Using an analysis similar to Thm. 5.14 we get the following theorem.

Theorem 5.16 (Pareto optimum: Asymptotic optimality). *Any limit point in $\hat{R}$ of the product dynamics is a social optimum.*

5.4 Experiments

This section presents simulation experiments motivated from urban autonomous driving for Alg. 5.2. Please refer [ZOCF14a] for experiments on Alg. 5.3.

5.4.1 *Setup*

Consider a Dubins vehicle with control on acceleration as a model for the dynamics of both the robot and the environment with the dynamics given by

$$\begin{aligned}\dot{x}(t) &= v(t)\cos(\theta(t)),\\ \dot{y}(t) &= v(t)\sin(\theta(t)),\\ \dot{v}(t) &= u_1(t),\\ \dot{\theta}(t) &= v(t)\,u_2(t),\end{aligned}$$

where the state is (x, y, v, θ) with bounds on acceleration, i.e., $|u_1(t)| \leq 1$ and turning rate, i.e., $|v(t)\,u_2(t)| \leq c$. Following a similar analysis as given in [CL07], we can see that time optimal paths between two states $(x_1, y_1, v_1, \theta_1)$ and $(x_2, y_2, v_2, \theta_2)$ for the dynamics given above can be split into two cases, (i) constrained by the shortest length Dubins curve, and, (ii) constrained by change in velocity. As an approximation, we only consider the first case here, i.e., the `steer` procedure returns an infinite cost if the increment in velocity is larger than what can be achieved along the shortest length Dubins curve. Note that this does not affect the completeness guarantees of Sec. 5.2.2. Also note that Dubins curves for any pair (x_1, y_1, θ_1) and (x_2, y_2, θ_2) are composed of straight lines and maximum turning rate segments and can be computed efficiently [Dub57].

Safety rules used in these examples are the same as the safety rules in Sec. 3.4.2. We however add an additional rule which is motivated by the fact that drivers on road do not like to slow down for others. We include a rule which encourages drivers to maintain the speed above V_{nom}.

$$\psi_3 = \mathsf{G}\,\neg(\mathsf{true}, slow)$$

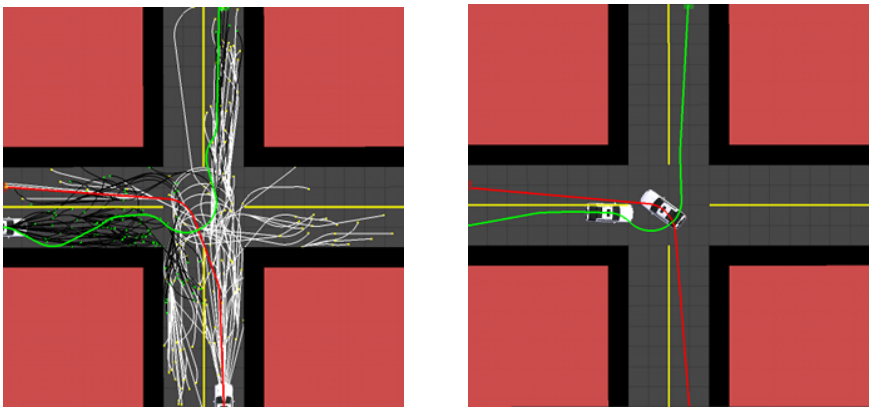
The priorities are set as $\varpi(\psi_1) = 1$ for the sidewalk rule, $\varpi(\psi_{2,1}) = \varpi(\psi_{2,2}) = 2$ for the direction and lane changing rules and $\varpi(\psi_3) = 3$ for the speed rule.

5.4.2 *Examples*

In Figs. 5.3 and 5.4, the Kripke structure maintained by **R** is shown in white while the Kripke structure maintained by **E**, i.e., K_e is shown in black. The current trajectory for **R** returned by Alg. 5.2 is plotted in red while the best response to this trajectory is plotted using green. Stationary obstacles are shown in red, while $\mathcal{X}_{sw}$ is shown in black. Right and left lanes are demarcated using yellow lines. Goal regions for both players, $\mathcal{X}_{r,G}, \mathcal{X}_{e,G}$ are shown in red and green respectively. Both players start with the same initial velocity V_{nom}.

Case 1: We first consider a scenario with both **R** and **E** at a cross-road junction to demonstrate the incremental nature of the algorithm. Fig. 5.3a shows the Kripke structures and the solution after 400 samples. It returns paths with cost 2.4231 for **R** which is close to optimal, but in order to satisfy the task specification without colliding with **R**, player **E** breaks the slow driving rule and incurs a cost of 6.819. Fig. 5.3b shows this path after 1000 samples.

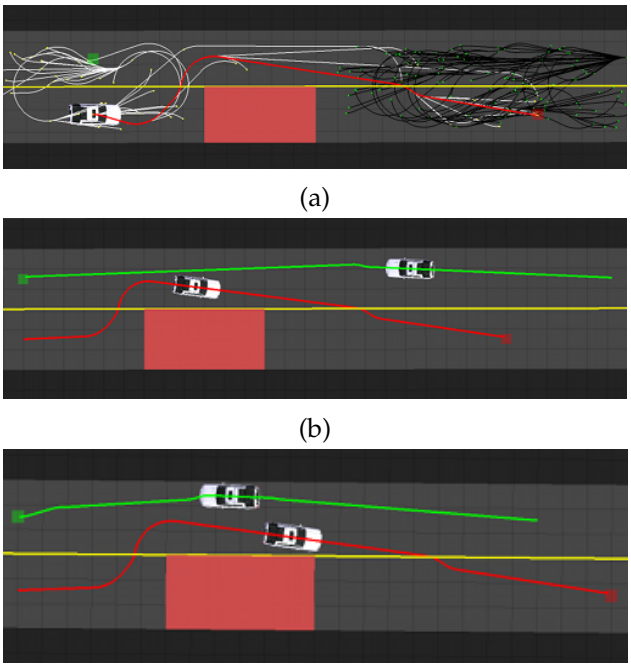
Case 2: In this example, we consider a single-lane road with an obstacle in the lane of **R**. Fig. 5.4a shows the algorithm after 500 samples where the environment cannot find any trajectory that reaches its goal without colliding with the trajectory that **R** would like to execute. With additional computation time, as shown in Fig. 5.4b, **E** obtains a trajectory that has a cost of 3.65 without breaking any safety rules. On the other hand, **R** incurs a penalty for breaking the lane



(a) (b)

Figure 5.3: Fig. 5.3a sub-optimal trajectories with sparse Kripke structures. As shown in Fig. 5.3b, with more samples, the algorithm converges to a trajectory for **E** that has a much smaller cost. Both cars are drawn at the same instant in time.

changing rule and obtains a best cost of 13.95.



(a)

(b)

(c)

Figure 5.4: Fig. 5.4a shows the Kripke structures when player **E** which starts at the root of the black tree does not have any trajectory to reach the goal region. The algorithm converges to the trajectories shown in Fig. 5.4b. On the other hand, as shown in Fig. 5.4c, if **E** starts closer to the obstacle, it is forced to obey the Stackelberg equilibrium and has to slow down to let **R** cross the obstacle.

***Case 3*:** Consider a modification of Case 2 where **E** starts slightly closer to the obstacle. In Fig. 5.4c, **R** now forces the Stackelberg equilibrium upon **E** and by still choosing the same trajectory. In order to avoid a collision, i.e., generate a valid best response that still satisfies the task specification, **E** slows down and incurs a cost for breaking rule ψ_3 to let **R** cross across the obstacle.

CHAPTER 6

Conclusions and Future Directions

This thesis introduced a collection of problems in the area of planning and control of autonomous agents using formal specifications. It identified a number of scenarios that are typical in urban autonomous driving, such as obeying traffic regulations which might make the underlying task infeasible, interacting with other drivers, merging on highways, negotiating cross-roads etc. The central idea pursued in this thesis is that low-level control synthesis needs to be integrated with high-level decision making to enable efficient, optimal and complete algorithms. We looked at three major ideas that develop this concept further.

The first part of the thesis uses ideas from sampling-based motion-planning algorithms to construct sequences of Kripke structures that are good approximations of the continuous-time dynamics. These Kripke structures also encode information about the temporal properties of continuous trajectories and can hence be used efficiently in model checking style algorithms. Specifically, we also ensured that a trace of the Kripke structure uniquely maps to the optimal trajectory that connects the two states and thus the temporal properties of the trajectory are encoded in the change in the atomic propositions across that transition in the Kripke structure.

The second part of the thesis considered task specifications that are infeasible to begin with and become feasible only if a few specifications, also known as "safety rules", are violated. These specifications were expressed using the finite fragment of Linear Temporal Logic which lent us a large number of automata-based model checking techniques for efficient verification of the specifications. We defined the "level of unsafety" as a metric to quantify the violation of the safety rules and used sampling-based techniques to construct trajectories of the dynamical system that minimize the level of unsafety while still satisfying the task specifications. In a related direction, we also considered languages such as process algebras that can be used to encode simple task specifications such as charging an electric vehicle, or pick-up and drop-off problems for mobility-on-demand. In addition to providing a number of theoretical guarantees such as asymptotic optimality and probabilistic completeness, the anytime nature of these algorithms make them extremely amenable to real-time implementations. We discussed results from both computational experiments and implementations on an autonomous platform.

The third part of the thesis delves into multi-agent control synthesis under temporal specifications. We modeled the interaction between various agents performing their tasks in a shared domain as a differential game between the agent and the environment. Using ideas from model checking and sampling-based algorithms, we devised algorithms that converge to established game theoretic notions of equilibria such as Stackelberg and non-cooperative Nash equilibria. It was shown using a number of examples that these equilibria also display "rational", i.e., human-like behaviors while driving on busy roads.

There are a number of future directions that one can explore which are motivated by the problems considered in this thesis.

- *Stochastic agents*: This area is the prime reason why it is unlikely that we will see large-scale deployment of autonomous urban navigation systems in the near future. A self-driving car has to interact with an enormous number of un-modeled scenarios, the problems considered in this thesis form only a small subset of those. Situations such as a busy pedestrian cross-walk, small lanes, where pedestrians and vehicles share the road together etc. are extremely hard to model. Roughly, in addition to preserving the notion that these disturbances are stochastic, we would like to impose certain "common-sense restrictions" upon their behaviors so that the autonomous agents are not entirely conservative. There is a growing literature on incorporating stochastic agents in planning using temporal verifications, see for example [WUB+13, WF12, CKSW13].

- *Multi-agent control*: Excursions along the game theoretic approaches considered in this thesis lead to a variety of problems in multi-agent control synthesis, which although tangential to the problem of autonomous urban driving, are still exciting problems in robotics. For example, consider [KDB11, USD+13, CDSB12] demonstrates a top-down approach, i.e., it decomposes a given LTL specification into local specifications for various agents. Along similar lines, a number of recent works bottom-up synthesis, e.g., [GD13]. These problems are of particular importance, for example, in verification and synthesis of embedded controllers in a large autonomous system. Efficiently checking sub-systems for partial satisfaction of specifications is useful if one can provide guarantees upon the performance of the larger inter-connected system, e.g., [Sti95].

  One might also consider general task specifications where groups of robots might interact in a co-operative way to complete sub-tasks and then again become self-interested for the remainder of the task using logics such as Alternating Time Logic [AHK02].

- *Human-in-the-loop verification*: On long journeys through the Australian outback, truck drivers regularly suffer from fatigue and boredom. Even if the roads are fairly straight, it is seen that most accidents start as long oscillations induced by drowsiness, which grow in amplitude until the vehicle goes out of control. Can an autonomous agent detect these — fairly minor, even benign — deviations from "standard" behavior? Tacking these problems requires that we have fairly good models of "human behavior". Instead of being merely stochastic, as is popular in contemporary literature, these models can be thought of as another agent trying to accomplish an aligned task, i.e., the interaction between the autonomous vehicle and the driver is a cooperative game.

  There are also a number of related issues in this area such as feedback loops between an autonomous agent enforcing certain specifications and the human driver trying to enforce contradictory behaviors. In such scenarios, we need to design algorithms for autonomous agents that *relinquish control* to the human driver.

References

[ABSP11] J.P. Aubin, A. Bayen, and P. Saint-Pierre. *Viability theory: New directions*. Springer-Verlag, 2011. 12

[AHK02] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002. 57

[ASF09] A. Arsie, K. Savla, and E. Frazzoli. Efficient routing algorithms for multiple vehicles with no explicit communications. *IEEE Trans. on Automatic Control*, 54(10):2302–2317, 2009. 12

[Bae05] Jos CM Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005. 35

[BCD97] M. Bardi and I. Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Birkhäuser, 1997. 12

[BEGL99] Francesco Buccafurri, Thomas Eiter, George Gottlob, and Nicola Leone. Enhancing model checking in verification by AI techniques. *Artificial Intelligence*, 112(1-2):57 – 104, 1999. 12

[BFS99] M. Bardi, M. Falcone, and P. Soravia. Numerical methods for pursuit-evasion games via viscosity solutions. *Annals of the International Society of Dynamic Games*, 4:105 – 175, 1999. 12

[BGK+11] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *Proc. of Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 326–340. Springer-Verlag, 2011. 12

[BIS09] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: Autonomous vehicles in city traffic*, volume 56. Springer, 2009. 9

[BK+08] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008. 16, 17, 19

[BO95] Tamer Basar and Geert Jan Olsder. *Dynamic noncooperative game theory*. SIAM, 1995. 12, 21, 22

[BPG93] MH Breitner, HJ Pesch, and W Grimm. Complex differential games of pursuit-evasion type with state constraints, Part 2: Numerical computation of optimal open-loop strategies. *J. of Optimization Theory and Applications*, 78(3):443–463, 1993. 12

[Buc89] S. J. Buckley. Fast motion planning for multiple moving robots. In *Int. Conf. on Robotics & Automation*, 1989. 12

[BWF+13] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Workshop on Alg. Foundations of Robotics*, pages 475–491. Springer, 2013. 43

[CCJ72] CL Chen and J Cruz Jr. Stackelburg solution for two-person games with biased information patterns. *IEEE Trans. on Automatic Control*, 17(6):791–798, 1972. 22, 45

[CCT+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. In *IEEE Conf. on Decision and Control*, 2013. 48, 49

[CDSB12] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Trans. on Robotics*, 28(1):158–171, 2012. 57

[CKSW13] Taolue Chen, Marta Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Quantitative Evaluation of Systems*. Springer, 2013. 57

[CL07] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In *IEEE Conf. on Decision and Control*, pages 2379–2384, 2007. 46, 54

[CQSP99] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre. Set-valued numerical analysis for optimal control and differential games. *Annals of the International Society of Dynamic Games*, 4(1):177 – 247, 1999. 12

[CRST08] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In *Verification, Model Checking and Abstract Interpretation*, pages 52–67. Springer-Verlag, 2008. 11

[CY98] Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events. *IEEE Trans. on Automatic Control*, 43:1399– 1418, 1998. 12

[DF11] Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *Formal Methods*, pages 12–26, Berlin, Heidelberg, 2011. Springer-Verlag. 12

[DKCB11] Xu Chu Ding, Marius Kloetzer, Yushan Chen, and Calin Belta. Automatic deployment of robotic teams. 18(3):75–86, 2011. 35

[DSBR11] Xu Chu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. MDP optimal control under temporal logic constraints. In *Proc. of IEEE Conf. on Decision and Control and European Control Conf.*, pages 532–538, 2011. 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*, pages 497–516, 1957. 29, 39, 46, 54

[ED05] Matthew G Earl and Raffaello D'andrea. Iterative MILP methods for vehicle-control problems. *IEEE Trans. on Robotics*, 21(6):1158–1167, 2005. 35

[ELP87] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(6):477–521, 1987. 12

[Eme97] E Allen Emerson. Model checking and the mu-calculus. *DIMACS series in discrete mathematics*, 31:185–214, 1997. 35

[Fai11] Georgios E Fainekos. Revising temporal logic specifications for motion planning. In *Int. Conf. on Robotics & Automation*, 2011. 11

[Fok00] Wan Fokkink. *Introduction to process algebra*. Springer, 2000. 20, 35

[FSS11] Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and broadcast time of random geometric graphs in arbitrary dimensions. In *Algorithms and Computation*, pages 190–199. Springer, 2011. 49

[FTO+08] Luke Fletcher, Seth Teller, Edwin Olson, David Moore, Yoshiaki Kuwata, Jonathan How, John Leonard, Isaac Miller, Mark Campbell, Dan Huttenlocher, et al. The MIT–Cornell collision and why it happened. *J. of Field Robotics*, 25(10):775–807, 2008. 10

[GD13] Meng Guo and Dimos V Dimarogonas. Reconfiguration in motion planning of single-and multi-agent systems under infeasible local LTL specifications. In *IEEE Conf. on Decision and Control*, 2013. 57

[GLL+99] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *Int. Journal of Computational Geometry & Applications*, 9(04n05):471–493, 1999. 12

[GP02] Elsa L. Gunter and Doron Peled. Temporal debugging for concurrent systems. In *Proc. of Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 431–444. Springer-Verlag, 2002. 18

[GPVW95] Rob Gerth, Doron Peled, Moshe Y Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. of the Int. Symposium on Protocol Specification, Testing and Verification*, 1995. 18

[Hau12] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *Workshop on Alg. Foundations of Robotics*, 2012. 12

[Isa99] Rufus Isaacs. *Differential games: A mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Dover Publications, 1999. 12

[KB06] Marius Kloetzer and Calin Belta. Ltl planning for groups of robots. In *Proc. of Int. Conf. on Networking, Sensing and Control*, pages 578–583, 2006. 11

[KDB11] Marius Kloetzer, Xu Chu Ding, and Calin Belta. Multi-robot deployment from ltl specifications with reduced communication. In *Proc. of IEEE Conf. on Decision and Control and European Control Conf.*, 2011. 57

[KF08] Sertac Karaman and Emilio Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *Amerian Control Conference*, pages 2003–2009, 2008. 35

[KF11a] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for a class of pursuit-evasion games. In *Workshop on Alg. Foundations of Robotics*, pages 71–87. Springer, 2011. 48

[KF11b] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. of Robotics Research*, 30(7):846–894, 2011. 11, 28, 38, 48, 50, 52, 53

[KF12a] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Amerian Control Conference*, 2012. 11

[KF12b] Kangjin Kim and Georgios Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *Int. Conf. on Intelligent Robots and Systems*, 2012. 11

[KF13] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Int. Conf. on Robotics & Automation*, pages 5041–5047, 2013. 27, 46

[KFS12] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. On the revision problem of specification automata. In *Int. Conf. on Robotics & Automation*, 2012. 11

[KGFP07] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *Int. Conf. on Robotics & Automation*, pages 3116–3121, 2007. 11, 35

[KRKF09] Sertac Karaman, Steven Rasmussen, Derek Kingston, and Emilio Frazzoli. Specification and planning of UAV missions: a process algebra approach. In *Amerian Control Conference*, pages 1442–1447, 2009. 35

[KZ86] K. Kant and S.W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *Int. J. of Robotics Research*, 5(3):72 – 89, 1986. 12

[LH98] S.M. LaValle and S.A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Trans. on Robotics*, 14(6):912–925, 1998. 12

[LHT+08] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Aalbert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. *J. of Field Robotics*, 25(10):727–774, 2008. 9

[LWAB10] Morteza Lahijanian, Joseph Wasniewski, Sean B Andersson, and Calin Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Int. Conf. on Robotics & Automation*, 2010. 11

[MBT05] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. on Automatic Control*, 50(7):947–957, 2005. 12

[MP95] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995. 18

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An open-source Robot Operating System. In *Workshop on Open-Source Software, ICRA*, 2009. 32, 39

[Rai01] Tuomas Raivio. Capture set computation of an optimally guided missile. *J. of Guidance, Control and Dynamics*, 24(6):1167–1175, 2001. 12

[Sch02] Philippe Schnoebelen. The Complexity of Temporal Logic Model Checking. *Advances in Modal Logic*, 4:393–436, 2002. 35

[Set96] J.A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Science*. Cambridge University Press, 1996. 12

[Sip12] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 3rd edition, 2012. 16

[SL02] G. Sanchez and J.C. Latombe. On delaying collision checking in PRM planning – Application to multi-robot coordination. *Int. J. of Robotics Research*, 21:5 – 26, 2002. 12

[SLL02] T. Simeon, S. Leroy, and J.-P. Lauumond. Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE Trans. on Robotics*, 18(1):42 – 49, 2002. 12

[Sor00] Pierpaolo Soravia. Optimal control with discontinuous running cost: eikonal equation and shape-from-shading. In *IEEE Conf. on Decision and Control*, volume 1, 2000. 45

[Sou99] P.E. Souganidis. Two-player, zero-sum differential games and viscosity solutions. *Annals of the International Society of Dynamic Games*, 4(1):69 – 104, 1999. 12

[STBR11] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints. *Int. J. of Robotics Research*, 30(14):1695–1708, 2011. 11

[Sti95] Colin Stirling. Local model checking games. In *CONCUR'95: Concurrency Theory*, pages 1–11. Springer, 1995. 57

[THK+13] Jana Tumova, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Hybrid Systems: Computation and Control*, 2013. 25

[TMD+06] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the DARPA Grand Challenge. *J. of Field Robotics*, 23(9):661–692, 2006. 9

[TP06] Paulo Tabuada and George J Pappas. Linear time logic control of discrete-time linear systems. *IEEE Trans. on Automatic Control*, 51(12):1862–1877, 2006. 11

[USD+13] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *Int. J. of Robotics Research*, 32(8):889–911, 2013. 57

[USDB12] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, and Calin Belta. Robust multi-robot optimal path planning with temporal logic constraints. In *Int. Conf. on Robotics & Automation*, 2012. 11

[VBJP10] Ondrej Vanek, B Bosansky, Michal Jakob, and Michal Pechoucek. Transiting areas patrolled by a mobile adversary. In *IEEE Symposium on Computational Intelligence and Games*, pages 9–16, 2010. 12

[VW94] Moshe Y Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994. 17

[WF12] Tichakorn Wongpiromsarn and Emilio Frazzoli. Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In *IEEE Conf. on Decision and Control*, 2012. 57

[WTM10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon control for temporal logic specifications. In *Hybrid Systems: Computation and Control*, pages 101–110, 2010. 10

[WUB+13] Tichakorn Wongpiromsarn, Alphan Ulusoy, Calin Belta, Emilio Frazzoli, and Daniela Rus. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *Int. Conf. on Robotics & Automation*, 2013. 57

[XA08] E.K. Xidias and N.A. Aspragathos. Motion planning for multiple non-holonomic robots: A geometric approach. *Robotica*, 26:525–536, 2008. 12

[ZM13] M. Zhu and S. Martínez. Distributed coverage games for energy-aware mobile sensor networks. *SIAM Journal on Control and Optimization*, 51(1):1–27, 2013. 12

[ZOCF14a] M. Zhu, M. Otte, P. Chaudhari, and E. Frazzoli. Distributed anytime game-theoretic learning for multi-robot motion planning - Part I : Trajectory based algorithms. *arXiv:1402.2708*, 2014. 49, 53

[ZOCF14b] Minghui Zhu, Michael Otte, Pratik Chaudhari, and Emilio Frazzoli. Game theoretic controller synthesis for multi-robot motion planning Part I : Trajectory based algorithms. In *Int. Conf. on Robotics & Automation*, 2014. 49