Constructive Formal Control Synthesis through Abstraction and Decomposition

Eric Kim

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-118
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-118.html

August 16, 2019

Constructive Formal Control Synthesis through Abstraction and Decomposition

by

Eric Shinwon Kim

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Murat Arcak, Co-chair
Processor Sanjit A. Seshia, Co-chair
Professor Roberto Horowitz

Summer 2019

Constructive Formal Control Synthesis through Abstraction and Decomposition

Abstract

Constructive Formal Control Synthesis through Abstraction and Decomposition

by

Eric Shinwon Kim

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Murat Arcak, Co-chair

Processor Sanjit A. Seshia, Co-chair

Control synthesis is the problem of automatically constructing a control strategy that induces a system to exhibit a declared behavior. Synthesis algorithms vary widely across different classes of system dynamics and specifications. While continuous optimization has traditionally been used to construct stabilizing controllers for physical systems modeled with differential equations, temporal logic synthesis for finite state machines heavily leverages discrete algorithms and data structures. Hybrid systems are a class of systems that exhibit both continuous and discrete behaviors, which are necessary to capture phenomena such as impacts for legged robots and congestion shockwaves in freeways. Tractable control synthesis remains elusive because hybrid systems violate many of the fundamental topological assumptions made by prior algorithms for purely continuous or discrete systems. This thesis exploits compositionality and system structure to provide a suite of algorithmic and theoretical techniques to tackle acute computational bottlenecks in hybrid control synthesis.

The first half of this thesis provides a framework for engineers to model control systems and construct algorithmic pipelines for control synthesis. By explicitly capturing system structure, this framework gives users the flexibility to rapidly iterate over and leverage a library of optimizations for control synthesis. We demonstrate this framework in the context of abstraction-based control, a synthesis workflow that translates continuous systems into finite state machines by throwing away high precision information. Different optimization techniques such as multi-scale grids, lazy abstraction, and decomposed synthesis, can all be expressed as modifications to a computational pipeline. We demonstrate computational gains while synthesizing safe motion primitives for numerous robotic examples.

The second half addresses distributed control synthesis where multiple controllers act as agents that seek to jointly satisfy a specification and are restricted by some communication topology. We introduce parametric assume-guarantee contracts as a formalism to derive guarantees about the closed loop behavior of a collection of interacting components. Dynamic contracts allow contract parameters to change at runtime and enable coordination of multiple

interacting sub-systems. These results are demonstrated in the context of a freeway ramp meter and an adjacent arterial network.

Contents

Acknowledgements

My years at Berkeley have been some of the most intellectually and personally fulfilling ones of my life. It is because of the people I met and the lessons they've taught me that I'll always remember this period fondly.

First and foremost I'd like to thank my advisors Murat Arcak and Sanjit Seshia who have both guided me in my growth from a student to a researcher. Without them both, pursing this path at the intersection of control systems and formal methods would have been a much more difficult. Murat's guidance helped me map larger abstract problems into manageable sub-problems. As a younger graduate student, I recall thinking that "I don't know what there is to do after this paper". Murat helped dispel this fear because under him I learned how to be more comfortable with dealing with research uncertainty and how to both critically and constructively appraise my own work. I first spoke with Sanjit while making my graduate school decision. My main background was in control theory but I was interested in expanding into formal methods. Speaking with him gave me the confidence that I would be able to pursue this path at Berkeley. It's through his emphasis on making my research contributions clear to to broader research community that I learned how to put my work in context and identify concrete research goals.

I'd like to thank Roberto Horowitz who as a member of my qualifying exam committee and involvement on the traffic project emphasized importance the of transitioning research into practice. Alberto Sangiovanni-Vincentelli provided constructive feedback on my research as a member of on my qualifying exam committee and afterward. Thank you to Pravin Varaiya, Alex Kurzhanskiy, and Gabriel Gomes for their sage-like wisdom and guidance in the weekly traffic meetings.

Day to day, most of my interactions were with fellow Berkeley graduate students who've helped me with the professional and personal parts of grad student life. In my first year Sam Coogan and Dorsa Sadigh guided me through the controls and formal methods literature and through the minutiae of graduate research. I also want to thank the Arcak lab for fruitful chats and lunch outings on Euclid: Ana Rufino-Ferreira, John Maidens, Mindy Perkins, Stanley Smith, Pierre-Jean Meyer, Mikhael Burov, Galaxy Yin, and Alex Devonport. Thanks to the learn and verify group, Marcell Vasquez-Chanlatte, Markus Rabe, Ankush Desai, Shromona Ghosh, Daniel Fremont, Edward Kim, Hadi Ravanbakhsh, Tommaso Dreossi, and Ben Caulfield, who have helped me understand formal methods and for their help refining presentations. To Matt Weber, I couldn't have asked for a better friend and co-GSI for EECS149/249. Shirley Salanio was also an invaluable resource for navigating grad school bureaucracy and never failed to answer my desperate last minute questions.

My collaborators outside of Berkeley have also had a substantial impact on my research. Thank you to Calin Belta and Sadra Sadraddini for their fruitful collaboration on the formal methods in traffic project. To my Germany colleagues Majid Zamani, Mahmoud Khaled, and Felix Gruber thank you for our joint work on SCOTS and its variants, which constitute a substantial part of this thesis. Thanks to Shinichi Shiraishi and Baek-Gyu Kim at Toyota ITC from whom I learned about industrial research and connected vehicles.

To my fellow "control freaks" Roel, Jaime, Jason, and Cathy, it was truly special starting

the PhD journey with you. Berkeley wouldn't have been the same without your friendship through the linear systems course, prelims, and beyond. I'll always remember our trips and wandering conversations, most of which ended in late night Top Dogs. With your concern and devotion to addressing long term societal impacts of our research, I'm sure the future is in safe hands.

To Adam and Colin, I'm glad we met during graduate school visit days and all (independently!) decided to come to Berkeley and live together. Along with Dexter, Sloka, and Christine, you helped turn the Halfwit house into a home. I'll fondly remember Dominion and D&D sessions, Kips trivia, and brunches all over the Bay Area with Ilina, Lucy, and Doug.

To Yeban, your love, enthusiasm, and unwavering support throughout our many days both together and apart mean the world to me. After many adventures along the West coast together, I can't wait to explore the rest of the world with you.

Of course, I wouldn't have been able to even start my PhD journey without the steady love and encouragement of my entire family, from my father Jungho, to mother Cheesue, sister Sera, and my many grandparents, uncles, aunts, and cousins. They've encouraged and challenged me from a young age to be my best self, academically and personally. As much as I've learned in grad school, it doesn't compare to the many intangible life lessons I've learned from them.

Chapter 1

Introduction

1.1 Designing Reliable Cyber-Physical Systems

Advances in computing hardware, communication, and sensing expand the capabilities of existing systems and also facilitate new forms of automation. Smart grids and intelligent cities can optimize the allocation of scarce resources and orchestrate the coordination of many heterogeneous systems. Completely new applications are also possible such as autonomous aerial and ground vehicles. Augmenting systems with more control modalities enables them to be more efficient, safer, and endows them with exciting new capabilities.

The act of engineering systems is a delicate dance between endowing them with extending their functionality and certifying that they behave correctly and reliably. While novel capabilities attract interest and investment in a new technology, societal trust in those very technologies erodes when they exhibit unexpected or unintended behavior. Losing this trust is detrimental because it often leads to the "trough of disillusionment" in Figure 1.1. Every technology is ultimately judged by its impact on the real world. Systems with direct agency over the physical world can exhibit many catastrophic failure modes; electric grids can experience blackouts, autonomous vehicles can execute unsafe maneuvers, and an air vehicle's autopilot system can cause a crash. The cost of failure is especially acute when human lives are jeopardized.

Systems must be certified to a much higher standard when failure is not an option. The burden of ensuring control systems are intelligent and safe falls on the algorithm designer. Before they can be deployed, safety critical autonomous systems need to satisfy numerous hard and soft requirements such as:

- Correctness: The system must behave as intended.
- Interpretability: A control system's decisions must be justified and easy to understand.
- Robustness: The controller must perform as expected despite discrepancies between the model and the true environment.

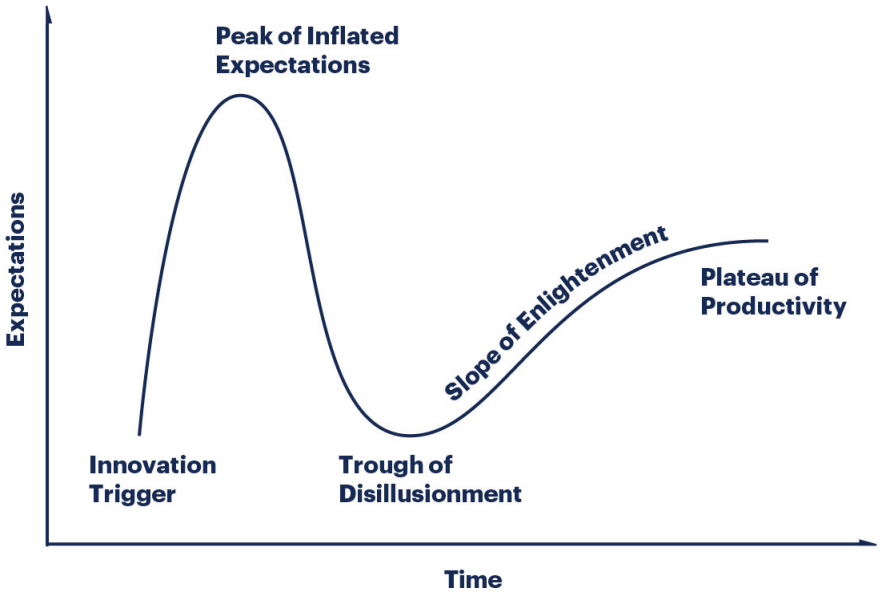


Figure 1.1: The Gartner Hype Cycle describes how the expectations for new emerging technologies undergo distinct phases as they mature. Verification and synthesis tools are necessary if one wants to leap over the trough of disillusionment and directly to the plateau of productivity.

- Modularity: Extensible and easy to modify/interact with others.

Translating a designer's intent into a correct implementation is notoriously difficult. Even after restricting interest to the "cyber" part of cyber-physical systems, security vulnerabilities regularly appear in mission critical software. Combining the two to create a cyber-physical system only increases the surface area of issues and concerns that a control designer must address. Furthermore, useful models for a system's physical behaviors are imperfect representations of the real world.

In light of the costs associated with development and validation of autonomous systems, there is an acute need for methods to bridge the gap between an engineer's *intent* and the *implementation* of the final product. A formal specification is a mathematical description of desirable behaviors that captures *what* a system should do instead of *how* to do it. Ideally, the system design process incorporates methods to systematically and algorithmically reason about whether the specification satisfies the system.

Tools for Cyber-Physical System Design

A control synthesizer is an algorithm or tool whose role is to take a formal specification and a system model and automatically construct a control algorithm that causes the system to enforce the specification. A controller designed to enforce the specification ensures that all possible system behaviors satisfy it. A well designed control synthesizer will accelerate the

	Control Theory	**Formal Methods**
Bottlenecks	Uncountable state space	Combinatorics
Structural Assumptions	Continuity	Discrete
Workhorse Algorithms	Continuous optimization	SAT/SMT, BDDs

Table 1.1: Successful algorithms from the control theory and formal methods communities exploit structural assumptions to abstract away critical bottlenecks.

design process by lifting the engineer's level of reasoning from low level controller design to declarative specifications.

The general robust control synthesis problem is easy to state mathematically. Translating that mathematical problem into an efficient implementation on the other hand is nontrivial because there is a gap between the theory and the algorithms used to solve it. The key to successful development of design tools and algorithms is exploiting system structure, whenever it is available.

The control theory and formal methods communities have both developed rich theoretical frameworks and powerful tools to systematically reason about system properties and dynamics. These tools make critical assumptions about the system and specification as shown in Table 1.1. These assumptions are often reasonable for their respective domains, but need to be dropped when tackling cyber-physical systems. The tools for controller design for continuous systems are rooted in linear algebra and continuous optimization. Stability theory leverages convex optimization and sum of squares programming to search for and verify Lyapunov functions. Receding horizon control uses a continuous optimizer in the loop to generate control inputs that enforce constraints online. Unlike control theory, the theoretical foundation of formal methods revolves instead around logic, computability theory, and automata theory. Tools for verification and synthesis include model checkers, Boolean satisfiability (SAT) solvers and satisfiability modulo theories (SMT) solvers, which have each found successful applications in hardware verification, circuit synthesis, compilers, and program synthesis.

Cyber-physical systems present a unique challenge because they often lack readily exploitable mathematical structure. Hybrid spaces with continuous and discrete components don't exhibit commonly assumed topological properties like continuity or finiteness. Tools often have these assumptions embedded within them and are not readily generalized to tackle cyber-physical systems.

One can view the process of designing reliable algorithms for autonomous systems as an exercise in taming uncertainty and complexity. Different systems and specifications contain domain specific structure, so algorithms designed for one system may not be applicable to another one. This lack of flexibility is a hindrance for computationally challenging problems, where it is often necessary to let the user rapidly identify and algorithmically exploit structure through analysis or experimentation. This thesis exploits compositionality and system structure to provide a suite of algorithmic and theoretical techniques to tackle acute

	Part 1	Part 2
Controllers	Centralized	Distributed
Time Horizon	Short	Long
Specification Classes	Safety + Reach	Temporal logic
Composition	Algorithm components	Control Systems
Sub-system Composition rules	Flexible	Restrictive
Notation Style	Logical	Set-based

Table 1.2: Key distinctions between approaches to controller synthesis in each half of this thesis.

computational bottlenecks in control synthesis for hybrid systems.

Thesis Statement

Efficient control synthesis algorithms are enabled by a modular and constructive theoretical framework that can algorithmically declare, identify, and exploit structure in the dynamics and specification.

1.2 Thesis Organization and Contributions

This thesis is broken apart into two parts with different approaches to controller synthesis. The two parts are complementary yet distinct in how they model systems, capture structure, and compose systems. These differences are highlighted in Table 1.2.

Part 1: Constructive Algorithms for Control Synthesis

The first half of this thesis focuses on centralized control synthesis. A centralized controller has access to full state information and can coordinate decisions across multiple controllable inputs. We restrict ourselves to a specific variant of control synthesis, called "abstraction-based control synthesis", where a continuous system is translated into a finite state machine that mimics the original dynamics. By restricting one's interest to a finite domain, this approach circumvents several thorny undecidability issues [86].

This part's goal is to lay a theoretical foundation for developing extensible tools for controller synthesis with finite abstractions. Successfully tackling high dimensional control synthesis will require a variety of theoretical and algorithmic techniques and a method to combine them. We show that the standard control synthesis algorithm can be decomposed into smaller components and furthermore can be restructured to exploit structural properties in the system and specification. By adopting a small, core framework it is easier for synthesis algorithms to dynamically reconfigure themselves at runtime and explore trade-offs between

different requirements such as tractability and completeness. This will ultimately provide a compositional foundation for one to combat acute computational bottlenecks that arise in abstraction-based control.

- **Chapter 3 Relational Interfaces for Control Systems**

  We present a compositional framework for representing systems, specifications, and synthesis algorithms. It leverages the theory of relational interfaces by Tripakis et al. [90] as a foundation, and adapts it to represent controller synthesis tasks. Interfaces are stateless input-output components and can be viewed as a system or a specification. They can capture features such as non-deterministic outputs and inputs that block. Operators in the relational interface theory take existing interfaces and construct new ones. We deconstruct many control synthesis operations, such as the robust controlled predecessor, and reveal that they can be viewed as composite operators stitched together from a small set of atomic operators.

  Contributions: This chapter provides a formal link between relational interfaces and controller synthesis. The original theory of relational interfaces was motivated by managing and reasoning about software requirements and design and not for synthesis. One novel contribution is the introduction of a controlled predecessor operator that takes advantage of a decomposed control system representation.

- **Chapter 4 Abstracting Control Systems**

  A system's abstraction summarizes its core behaviors in an abridged form. One can reduce the complexity of the controller synthesis problem by solving it over a simpler model. Proper abstractions faithfully represent the concrete system's behaviors. Reasoning over abstractions yields results that can be extrapolated back into statements about the concrete system. Interfaces are equipped with the notion of a refinement partial order, which formalizes the relationship between concrete and abstract components.

  Operators like interface composition and variable hiding are monotone with respect to the refinement relation and are used to move "horizontally" across the order. Other operators like coarsening and refinement are used to move vertically within the order and construct abstractions. The "direction" of new composite operators can easily be established through simple reasoning about the cumulative directions of their constituent operators. This allows one to guarantee the ability to refine discrete controllers back to concrete ones via the refinement relation.

  Contributions: We demonstrate a variety of modifications to the control synthesis pipeline such as dynamic coarsening and flexible abstraction-refinement.

- **Chapter 5 Implementation and Benchmarks for Finite Abstraction-based Control Synthesis**
  This chapter applies the insights from the prior two chapters to the abstraction-based control synthesis pipeline. It includes details about how quantizers are implemented in practice, demonstrates how abstractions can be constructed by computing forward reachable sets, and showcases a computationally constrained control predecessor that is aware of machine memory usage. A collection of benchmarks showcase how these modifications to the synthesis pipeline reduce computation time, memory, and the number of sample required to construct abstractions.

  Contributions: We demonstrate novel techniques that have not appeared in prior tools. These tools also contain some implicit assumptions which, aided through the relational interface formulation, we reveal and relax.

Part 2: Distributed Control Synthesis with Formal Objectives

The second half of this thesis addresses distributed control, where control inputs are not determined by a single centralized controller but instead by a collection of them. Distributed controllers commonly arise in multi-agent robotics or in networked control systems such as vehicular traffic networks. A distributed controller is often preferred to a centralized one when computing a centralized controller is intractable or when the communication overhead for a large controller is high.

When multiple systems with distributed controllers are interconnected, they induce new behaviors. Naively and haphazardly interconnecting these systems can lead to unintended and undesirable effects. In a distributed control setting, sub-controllers may have restricted access to information about the global state or be unable to coordinate actions with others. We use compositional reasoning to certify that these interacting components enforce global behaviors.

- **Chapter 6 Parametric Assume Guarantee Reasoning**

  An assume-guarantee contracts is a high level representation of a system's behavior. They capture input-output properties over signals and can be expressed in temporal logic or as finite gain conditions. Whenever the system's assumptions on the environment are satisfied, it ensures that the guarantees are satisfied. A parametric contract consists of a collection of contracts indexed with a parameter domain, which can be thought of as different environmental scenarios. We show that reasoning over the parameter domain enables one to derive tight guarantees about the behavior of interconnected systems.

  Contributions: This chapter bridges results and concepts from assume-guarantee reasoning to robust control theory. We generalize the small gain theorem, a classical result from robust control, into a parametric assume-guarantee counterpart. This new small

gain theorem enables one to reason about contracts encoded in linear temporal logic with continuous parameters.

- **Chapter 7 Dynamic Contracts and Coordination**

  This chapter adopts the parametric assume-guarantee reasoning rules from Chapter 6 and leverages them for distributed controller synthesis. Contracts for control synthesis ensure that multiple systems do not inadvertently violate a specification by having each sub-system agree to adhere to a restricting sub-system behaviors. Static contracts are computed offline do not react to runtime conditions. This results in conservative guarantees and sub-optimal behaviors in practice, especially when the contracts were designed with a conservative environment model. Dynamic contracts consist of a library of static contracts. They update the restrictions on sub-system behaviors in reaction to runtime conditions, yielding both tighter guarantees about the global system behavior and permitting more aggressive actions. A contract coordinator is a specially designed state machine that ensures that each sub-component satisfies a formal specification. We show how dynamic contracts reduce delays on a distributed traffic control example, while also guaranteeing that a temporal logic specification is satisfied.

  Contributions: Dynamic contracts bridge the gap between offline controller design and real-time decision making. Previously, contracts did not account for real-time state information and were unnecessarily conservative to ensure satisfaction of formal guarantees with distributed controllers.

- **Chapter 8: Identifying Communication requirements**

  It is not possible to enforce certain specifications with a distributed control architecture without any coordination amongst controllers, especially when systems contain coupled dynamics and objectives. Even when all controllers have global state information, they can still be uncertain about what actions other agents will perform. Collision avoidance is one such example where a centralized coordinator is required to resolve this uncertainty. The coordinator can be viewed as an agent that breaks symmetries or imposes a priority amongst agents, but in practice can be implemented as a distributed consensus protocol. We characterize where it is necessary for agents to communicate with one another to satisfy a safety objective. This characterization can be generalized to the case when there are communication delays. The coordination region is constructed for an intersection collision avoidance example.

  Contributions: The problem of identifying communication requirements in a control theoretic setting had not previously been tackled. This novel formulation helps bridge the gap between distributed and centralized controllers, especially in a safety critical setting.

Chapter 2

Background

In this chapter we introduce the mathematical framework for discrete-time controller synthesis and highlight key computational bottlenecks that arise during implementation.

2.1 Discrete-Time Control Systems

A discrete time interval is an ordered sequence of values. The open interval $I = [a, b)$ where $a < b$, $a \in \mathbb{Z}_{\geq 0} \cup \{-\infty\}$ and $b \in z \cup \{\infty\}$ denotes a set of time values $\{a, a+1, \ldots, b-1\}$. The closed interval $[a, b]$ includes b, that is, $[a, b] = [a, b-1)$ When an interval consists of a single point, a more concise notation is adopted $[a] = [a, a]$.

Consider a space $\mathcal{P}$ representing some collection of values. The space of signals $\mathcal{P}[\cdot]$ is given by a Cartesian product indexed by elements of the interval I:

$$\mathcal{P}[\cdot] = \prod_{k=a}^{b} \mathcal{P}. \tag{2.1}$$

For a signal $p[\cdot] \in \mathcal{P}[\cdot]$, let $p[k]$ represent its value at time $k \in I$. Similarly $p[a, b)$ and $p[a, b]$ respectively represent slices of the signal along intervals $[a, b)$ and $[a, b]$.

The set $\mathcal{P}^*$ represents the collection of all signals of finite length and with respect to a known start time (0 in the case below).

$$\mathcal{P}^* = \bigcup_{i \in \mathbb{N}} \left(\prod_{k=0}^{i} \mathcal{P} \right).$$

The set $\mathcal{P}^\omega$ represents the collection of all signals of infinite length.

$$\mathcal{P}^\omega = \prod_{k=0}^{\infty} \mathcal{P}.$$

Let $\mathcal{X}$ be a set of states and $\mathcal{U}$ be a set of control inputs. Variables $x[k] \in \mathcal{X}$ $u[k] \in \mathcal{U}$ represent value of the state and control input at time step k. We model discrete time control

systems with the difference inclusion $F : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ that imposes a constraint on the state evolution over time.

$$x[k+1] \in F(x[k], u[k]). \tag{2.2}$$

This is a model that can capture behaviors like non-determinism in the system dynamics or blocking states which encode regions where the system model breaks down. Both of these phenomena are encoded within the cardinality of $|F(x[k], u[k])|$ for a fixed state-input pair.

- When $|F(x[k], u[k])| = 1$, next state $x[k+1]$ is deterministically chosen.
- $|F(x[k], u[k])| > 1$, next state $x[k+1]$ is non-deterministically chosen. We view this non-determinism as adversarial.
- $|F(x[k], u[k])| = 0$, the system blocks because no next state $x[k+1]$ may be chosen. This can represent regions where the model F breaks down.

2.2 Specifications and Controller Synthesis

In order to perform control synthesis, one needs to provide the algorithm both with dynamic constraints and a desired behavior that ought to be satisfied. This desired behavior is encoded as a specification, which is a constraint encoded as a subset of a signal space.

Definition 1 (Specification)**.** *A specification $\phi \subseteq \mathcal{X}^{\omega}$ is a subset of the state signal space.*

The specification is said to be *formal* when it is written in a mathematical language and designed in such a way that they can be algorithmically verified or as target behaviors for control synthesis tools. Examples of such specification languages include linear temporal logic [71] and signal/metric temporal logic [52].

Two particularly relevant specifications for control synthesis encode safety and reach objectives. An element of the signal space $x[0, \infty]$ satisfies a

- safety objective with safety set $S \subseteq \mathcal{X}$ if $x[k] \in S$ for all $k \in \mathbb{N}$.
- reach objective with target set $T \subseteq \mathcal{X}$ if $x[k] \in T$ for some $k \in \mathbb{N}$.
- reach-avoid objective with safety set $S \subseteq \mathcal{X}$ and target set $T \subseteq \mathcal{X}$ if there exists some $k \in \mathbb{N}$ where $x[k] \in T$ and $x[k'] \in S$ for all $k' \in [0, k)$.

Controller synthesis is the problem of taking a control system and constructing a controller such that the closed loop behavior satisfies some specification. We first define a memoryless controller, then generalize to controllers which may contain memory.

Definition 2 (Controller)**.** *A controller for a system with state space $\mathcal{X}$ and $\mathcal{U}$ is characterized by a map C from finite sequences of observed states to a collection of admissible inputs*

$$C : \mathcal{X}^* \to 2^{\mathcal{U}}$$

Such a controller is said to be memoryless if $C : \mathcal{X} \to 2^{\mathcal{U}}$ takes a state as input rather than a sequence of states. Because controllers may output a collection of admissible values, at runtime any of these values can be chosen.

Definition 3 (Closed Loop System Behaviors)**.** *A state-input signal is an element of the closed loop behavior set if for all $k \in \mathbb{N}$ the dynamics constraint $x[k+1] \in F(x[k], u[k])$ and the controller constraint $u[k] \in C(x[0,k])$ are satisfied.*

The set of closed loop behaviors contains more than one element when the controller or the dynamics exhibit non-determinism. A controller is said to enforce a specification ϕ from an initial state $x[0]$ if all potential closed loop behaviors from that state are contained within ϕ.

We are now ready to pose the controller synthesis problem.

Definition 4 (Control Synthesis)**.** *Given a system $F : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ and a specification $\phi \subseteq \mathcal{X}^\omega$, construct a controller $C : \mathcal{X}^* \to 2^{\mathcal{U}}$ and a collection of initial states $\mathcal{X}_0 \subseteq \mathcal{X}$ such that all closed loop behaviors from $\mathcal{X}_0$ satisfy the specification.*

Ideally, the collection of initial states $\mathcal{X}_0$ is as large as possible. That is, if there exists a satisfying control strategy from some initial state then that state is contained within $\mathcal{X}_0$.

One common metaphor underlying many control synthesis algorithms is a zero-sum game between a controller, which tries to enforce the specification, with an adversarial environment that tries to violate it. The environment in this case is embodied through the non-determinism encoded in the dynamics. A winning strategy that guarantees specification satisfaction can be converted into a controller. Such a strategy can be constructed by leveraging the principle of dynamic programming to temporally break apart the synthesis problem into smaller problems that each correspond to a game played over a single time step. This yields a sequence of states that encodes when and where the controller or environment wins.

The robust control predecessor is an operator that encodes the game over a single time step. It takes a set of states Z and outputs a set of states $\texttt{cpre}(F, Z)$ from which a controller could ensure that the system F will be in Z. The controller must be robust to any non-determinism in the dynamics F. While both Z and $\texttt{cpre}(F, Z)$ could be interpreted as subsets of $\mathcal{X}$, Z is associated with a time $k+1$ while $\texttt{cpre}(F, Z)$ is associated with time k.

Definition 5 (Robust Control Predecessor Operator)**.** *The robust controlled predecessor operator takes as input the dynamics $F : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ and a set of states $Z \subseteq \mathcal{X}$ and yields another set of states*

$$\texttt{cpre}(F, Z) = \{x : \exists u \text{ such that } \varnothing \neq F(x, u) \subseteq Z\}. \tag{2.3}$$

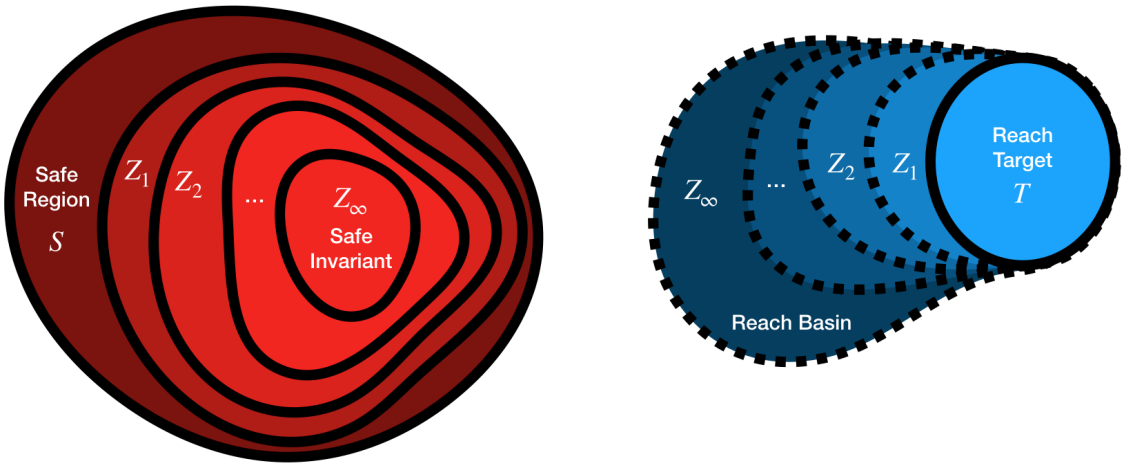


Figure 2.1: Visualizations of the safety and reach game iterations as constructing a collection of state space subsets. The safety game starts with the safe region S and shrinks it until a safe invariant is computed, while for the reach game the initial target T is expanded until a reach basin is computed. The fixed point Z_∞ is not always guaranteed to be reached in a finite number of iterations.

The operator evolves backwards in the sense that the set of states Z is associated with a time step k

It encodes the collection of states x from which there exists an assignment to u that guarantees that all potential next states $F(x, u)$ within will satisfy Z, despite any non-determinism contained in F. The requirement $\varnothing \subset F(x, u)$ also ensures that a next state exists and the system execution does not block.

Using the controlled predecessor, we can solve for a region for which the target T can be reached via the iteration:

$$Z_0 = T \tag{2.4}$$
$$Z_{i+1} = \texttt{cpre}(F, Z_i) \cup T \tag{2.5}$$

The following iteration characterizes a maximal controlled invariant set that ensures the safety constraint S is satisfied:

$$Z_0 = S \tag{2.6}$$
$$Z_{i+1} = \texttt{cpre}(F, Z_i) \cap S \tag{2.7}$$

Both of these iterations can be visualized in Figure 2.1. The reach game begins with the target T and then outputs a sequence of sets that grows until a reach basin is computed. Likewise, the safety game begins with the safe set S and shrinks until a safe invariant is computed.

While in general the above iterations are not guaranteed to reach a fixed point in a finite number of iterations, they are under certain technical conditions such as when the state space is finite [86].

Illustrative Dubins Vehicle Example

As a simple, instructive example consider a planar vehicle that is tasked with reaching a desired location. Let $x = \{p_x, p_y, \theta\}$ be the collection of state variables and $u = \{v, \omega\}$ be a collection input variables to be controlled. Let $x^+ = \{p_x^+, p_y^+, \theta^+\}$ represent state variables at a subsequent time step.

The discrete time dynamics are given by the collection of constraints

$$p_x^+ == p_x + v\cos(\theta) \qquad (F_x)$$
$$p_y^+ == p_y + v\sin(\theta) \qquad (F_y)$$
$$\theta^+ == \theta + \frac{v}{L}\sin(\omega) \qquad (F_\theta)$$

where $L = 1.4$ is a constant representing the length of the vehicle. The continuous state space is the hyperrectangle $[-2, 2] \times [-2, 2] \times [-\pi, \pi)$, where the last component is periodic so $-\pi$ and π represent identical values. The input space for the forward velocity v is $\mathcal{D}(v) \equiv \{0.25, 0.5\}$ and steering angle ω is $\mathcal{D}(\omega) \equiv \{-1.5, 0, 1.5\}$.

Let F collectively represent the constraints (F_x) - (F_θ) encoding the system dynamics. The target region T is $[-0.4, 0.4] \times [-0.4, 0.4] \times [-\pi, \pi)$; that is, the vehicle's position must reach a square but the orientation does not matter.

An approximate solution to the reach game with target T is depicted in Figure 2.2. It is guaranteed to be an underapproximation of the true solution to the reach game. It is generated with the techniques subsequently introduced in the first half of this thesis. Further details are provided in Chapter 5.

2.3 Algorithmic Challenges

Many algorithmic issues are obfuscated from the elegant mathematical summary of control synthesis.

1. **Non-termination and undecidability**: Each game iteration is associated with a time step. If a fixed point is reached, then one may extrapolate the game's results to an infinite time horizon. The iterations in the safety, reach, and reach-avoid games are unfortunately not guaranteed to terminate or reach a fixed point in general. Moreover, the problem of identifying if a state space subset is reachable is undecidable, even for a simple class of hybrid systems [42].

2. **Computation** How is the controlled predecessor computed and what are its runtime requirements?

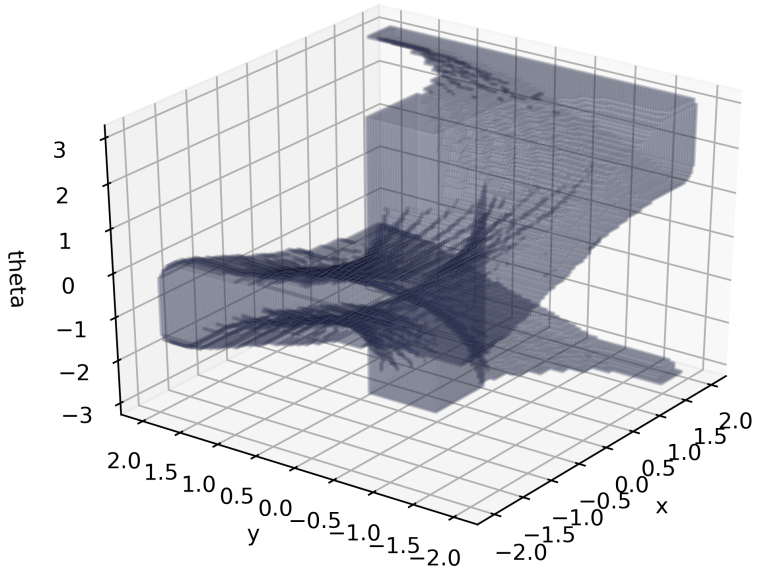


Figure 2.2: Approximate solution to the Dubins vehicle reach game visualized as a subset of the state space.

3. **Representation**: What data structures are used to represent F, Z_i, and regions S and T? The domain of these sets can be infinite and it's difficult to efficiently encode them in memory. How can one leverage efficient data structures that capture salient structural properties of these sets? How can one mitigate the cost of increased memory requirements incurred as the state-input space grows?

This thesis tackles the above problems through a variety of approaches. The first half of this thesis shows a modular framework for developing control synthesis algorithms that reflect domain specific structure. It uses relational interfaces [90] to represent the controlled predecessor `cpre`$(\cdot)$ and iterations (2.5) and (2.7) as a computational pipeline. By explicitly representing it within a small yet powerful framework, it becomes easier to identify structural properties and modify this pipeline. Different modifications can encode heuristics to extract computational gains or to encode favorable theoretical properties such as algorithm termination. The second half of this thesis decomposes the control system and specification. Unlike the first half, the controllers are distributed and either do not have access to another system's state or cannot coordinate their decisions with other controllers. Each of the closed loop systems are then interconnected and "stitched" together. One can guarantee that they satisfy a global specification using assume-guarantee reasoning.

Part I

Constructive Algorithms for Control Synthesis

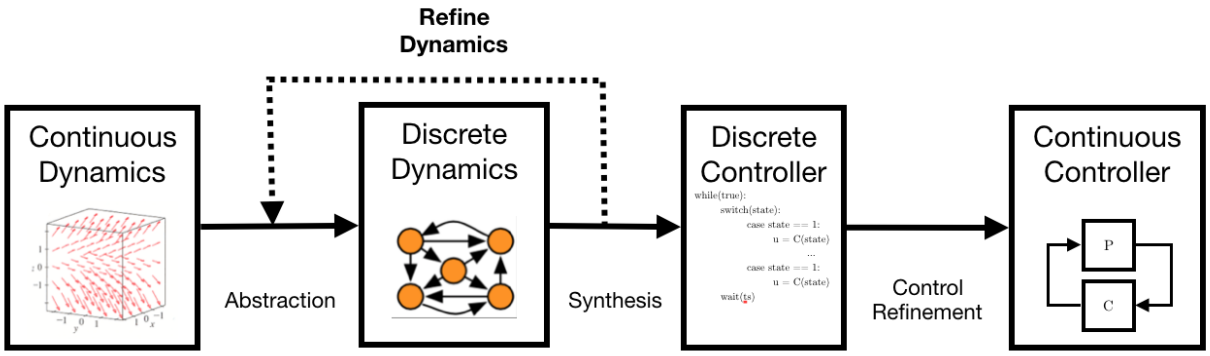


Figure 2.3: Traditional abstraction-based control synthesis pipeline. The controller refinement step only yields meaningful results if the discrete abstraction faithfully mimics the continuous dynamics.

A control synthesizer translates a declarative specification into an implementation that fulfills those requirements. If properly implemented into the autonomy design process, synthesizers can accelerate development and reduce time between design iterations.

The control synthesizer ideally outputs a *robust* controller. Errors in the the system dynamics model, uncertainty in the specification, and noise in the state observation can all cause the specification to be violated. The synthesizer can account for these error sources by explicitly reasoning about them while constructing the controller. Synthesis in the robust control setting reduces down to solving a zero-sum game between two agents. One agent is the controller that seeks to enforce a specification. The aforementioned errors are aggregated into another agent representing an adversarial environment that seeks to violate the specification.

In the first half part of the thesis, we focus on safety, reach, and reach-avoid specifications. Controllers enforcing these specifications can be viewed as motion primitives that can serve as building blocks for higher level algorithms such as graph based path planners [43] and for temporal logic specifications [70]. Solutions to safety, reach, and reach-avoid games are played over a series of time steps. They exhibit a temporal structure that can be exploited by the principle of dynamic programming, which breaks apart the problem into a collection of sub-problems over shorter time horizons [9]. The Hamilton-Jacobi-Isaacs (HJI) partial differential equation characterizes the solutions to continuous time versions of these games [7]. The strength of the HJI characterization is its generality. It is an elegant mathematical solution that can accommodate nonlinear and time-varying dynamics and objectives [32]. This generality can be a liability in practice because the algorithms to solve the HJI equations must accommodate a variety of systems and are not readily specialized to specific use cases. The level set toolbox by Mitchell and Templeton [62] numerically solves the HJI equations over a discrete grid, but runs into memory bottlenecks as the state dimension grows. The HJI solution is represented as a lookup table representing a function over a discrete grid.

Control synthesis via finite abstractions presents an alternative approach to numerically solving the HJI partial differential equation over a discrete grid. It is motivated by the

following question:

If discretization is ultimately necessary to implement a control system algorithm, are there any benefits to discretizing the original continuous mathematical problem statement as well?

Finite abstraction of continuous dynamical systems has been an active area of research from the very birth of the hybrid systems community [3, 86]. Finite abstractions allow one to circumvent thorny decidability issues [42] when mixing continuous and discrete dynamics and also to synthesize controllers for temporal logic specifications.

Figure 2.3 depicts the traditional abstraction-based control synthesis pipeline which occurs over three stages.

1. Abstracting the continuous state system into a finite automaton whose underlying transitions faithfully mimic the original dynamics [86], [95].
2. Synthesizing a discrete controller by leveraging data structures and symbolic reasoning algorithms to mitigate combinatorial state explosion.
3. Refining the discrete controller into a continuous one. Feasibility of this step is ensured through the abstraction step.

A system relationship formalizes the ability to extrapolate properties from an abstraction to the concrete system. Different system relationships enable extrapolation of different kinds of properties, such as behavior satisfaction or controllability. If one can establish a bisimulation relation between an abstract system and the original one, then verifying a property on the abstract system is equivalent to doing so for the original system. The bisimulation relationship thus establishes a necessary and sufficient condition for specification satisfaction. A bisimulation relation isn't always guaranteed to exist except for certain restricted classes of systems such as timed automata as shwon by Alur and Dill [2]. Alternating bisimulation relations are another system relation used for control synthesis [86] rather than verification. While (alternating) bisimulation relations can only be constructed for a limited class of hybrid systems, other system relationships are more general but also give weaker guarantees. An (alternating) bisimulation relation can be relaxed into an (alternating) simulation relation which only encodes a sufficient condition. If a simulation relation exists, then abstraction satisfaction of a property is a sufficient condition for the original system to satisfy it. Approximate (bi)simulation relations are another relaxation that are utilized when systems evolve over a metric space [33, 94, 35, 34, 72, 77]. Reissig et al. [76] developed feedback refinement relations as an alternative to alternating simulation relations. It has a slightly stricter condition that has more favorable theoretical properties. Namely, the refined controller for feedback refinement relations is simpler to compute and represent in memory.

Related Work

There are a variety of tools that automate the abstraction, control synthesis, refinement pipeline depicted in Figure 2.3. These tools `PESSOA` [60], `SCOTS` [78], `MASCOT` [44], `ROCS` [55],

`ARCS` [16], `TuLiP` [93], `conPAS2` [51] systematically translate continuous dynamics into finite approximations and synthesize controllers through discrete reasoning. Two common computational bottlenecks arise in the finite abstraction approach. First, many existing algorithms to construct finite abstractions are sample heavy; some even require an enumerative traversal of the state-input space which grows exponentially with dimension. Second, even if an abstraction were successfully constructed the synthesis algorithms do not always scale to systems of higher dimensions.

Recent publications have solutions that exploit common topological and algebraic properties of dynamical systems. One solution by Hsu et al. [44] is to employ multi-scale grids and hierarchical models that capture notions of locality in the state space. Another solution, by Nilsson et al. [65] and Li and Liu [55] is known as abstraction-refinement or lazy abstraction. It incrementally constructs an abstraction while performing the control synthesis step. A third solution is to represent systems as a collection of components and decompose abstraction and synthesis algorithms with respect to the interconnection structure [61] [38]. These solutions have been developed in isolation and were not previously interoperable.

Methodology

The existing control synthesis formalism does not readily lend itself to algorithmic modifications that reflect and exploit structural properties in the system and specification. We use the theory of relational interfaces by Tripakis et al. [90] as a foundation and augment it to express control synthesis pipelines. Interfaces are used to represent both system models and constraints. A small collection of atomic operators allows one to construct computational pipelines in control synthesis by manipulating interfaces. This collection is powerful enough to reconstruct many existing algorithms by stitching together atomic operators into composites. New operators can easily be added to encode desirable heuristics that exploit structural properties in the system and specifications. Interfaces come equipped with a refinement partial order that formalizes when one interface abstracts another. Interface composition and variable hiding are monotone with respect to the refinement order and are used to move horizontally across the order. Coarsening and refinement operators are used to move vertically and construct abstractions. The "direction" of new composite operators can easily be established through simple reasoning about the cumulative directions of their constituent operators. This thesis focuses on preserving the refinement relation and sufficient conditions to refine discrete controllers back to concrete ones. Additional guarantees regarding completeness, termination, precision, or decomposability can be encoded, but impose additional requirements on the control synthesis algorithm and are beyond the scope of this section.

Contributions

This half of the thesis bridges the gap between theory and implementation by incorporating compositionality into the theoretical foundation for control synthesis. To our knowledge, the application of relational interfaces to robust abstraction-based control synthesis is new.

The framework's building blocks consist of a collection of small, well understood operators that are nonetheless powerful enough to express many prior techniques. Encoding these techniques as relational interface operations forces one to simplify, formalize, or remove implicit assumptions that are embedded in existing tools. The framework also exhibits numerous desirable features.

1. It enables compositional tools for controller synthesis by leveraging a theoretical foundation with compositionality built into it. A compositional algorithmic framework enables users to explicitly take advantage of system specific structure in the dynamics and specification.

2. It enables a declarative approach to control synthesis by enforcing a strict separation between the high level algorithm from its low level implementation. We rely on the availability of an underlying data structure to encode and manipulate predicates. Low level predicate operations, while powerful, make it easy to inadvertently violate the refinement property. Conforming to the relational interface operations minimizes this danger.

3. The framework explicitly captures the data flow in the algorithm. Doing so makes it easier to introduce algorithmic and dynamic modifications the control synthesis pipeline such as dynamically tuning hyper-parameters to account for memory constraints.

This framework is domain agnostic and applicable to robust control synthesis problems over both continuous and discrete domains.

Organization

This half of the thesis consists of three chapters.

- **Chapter 3** Introduces relational interfaces describes how they model systems, constraints, and algorithms.

- **Chapter 4** Introduces the interface refinement order and a collection of operators used to traverse it.

- **Chapter 5** Applies the insights of the prior two chapter and contains a wealth of domain specific optimizations applied to the finite abstraction-based control synthesis pipeline. It also contains a collection of examples that showcase computational gains compared to the standard pipeline.

Chapter 3

Modeling Control Systems, Constraints, and Algorithms with Relational Interfaces

In this half of the thesis, systems are symbolic descriptions to represent the dynamic behavior of an object. For instance, a ball's movement is governed by a collection of differential equations encoding the laws of physics, while a program's execution is constrained by the underlying hardware. They can be represented in a variety of ways including:

- Source code
- Mathematical equations and expressions
- Simulink models

One common facet of each of these models is that they use a compositional modeling language to encode complex phenomena via the interaction of simple components. In the case of source code the simple components are defined by the grammar of the programming language while for mathematical equations and expressions it is the atomic mathematical operations and functions.

We desire an expressive framework to capture many salient features of compositional design and analysis. The theory of relational interfaces [90] is a simple, yet powerful, framework for modeling large complex systems by connecting smaller components. Interfaces can be visualized as input-output blocks and will be used to represent both systems and specifications. Interfaces can also encode requirements and assumptions. The theory comes equipped with a collection of operators to construct new interfaces from existing ones. We show how the standard control synthesis procedure is implicitly represented by composing a sequence of operations together.

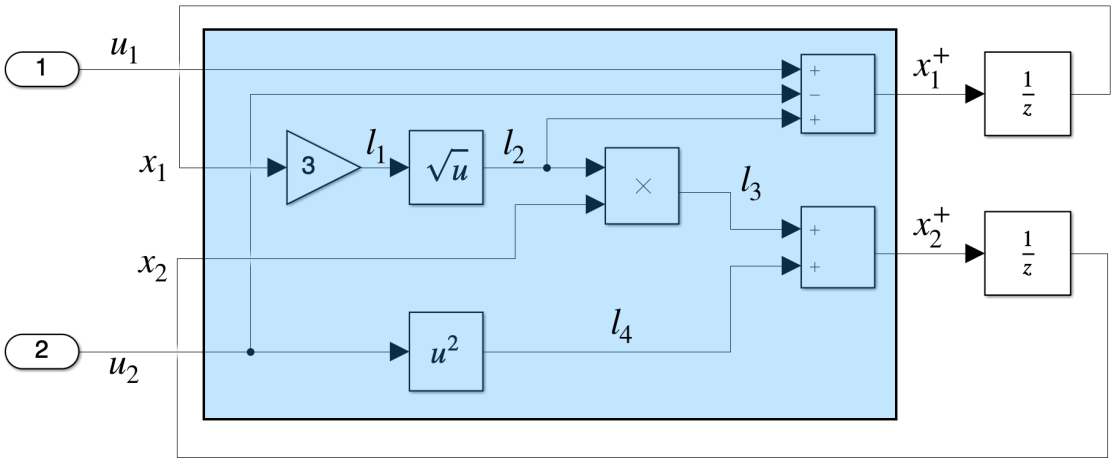


Figure 3.1: Discrete time control system encoded as a Simulink block diagram here.

Introductory Example

In the following simple example, we demonstrate that many control systems can be expressed as the composition of smaller atomic components and also expose subtle issues that arise when components exhibit non-determinism or blocking inputs. Consider the following discrete time control system depicted as the blue box in Figure 3.1:

$$x^+ := f(x, u) \tag{3.1}$$

with two continuous current states $x_1, x_2 \in \mathbb{R}$, two control inputs $u_1, u_2 \in \mathbb{R}$, and two next states $x_1^+, x_2^+ \in \mathbb{R}$. Note that ":=" represents the assignment operation and is distinct from a mathematical assertion "=" or a condition evaluation "==". Let $f(x, u)$ be concretely given by

$$\begin{pmatrix} x_1^+ \\ x_2^+ \end{pmatrix} := \begin{pmatrix} \sqrt{3x_1} - u_2 + u_1 \\ x_2\sqrt{3x_1} + u_2^2 \end{pmatrix}. \tag{3.2}$$

The encoding map (3.1) directly maps the variables u_1, u_2, x_1, x_2 to a next state x_1^+, x_2^+, but this obfuscates some internal structure that is more clear when the system is viewed as the blue box in Figure 3.1. Latent variables $l_1 := 3x_1$, $l_2 := \sqrt{3x_1}$, $l_3 := x_2\sqrt{3x_1}$, and $l_4 := u_2^2$ capture intermediate computations, with l_2 is also shared across updates for both x_1^+ and x_2^+. These latent variables correspond with wires that are fully contained inside the blue box or the function body's internal variables. The representation immediately below includes

these latent variables and is a counterpart to that of Equation (3.2).

$$l_1 := 3x_1 \tag{3.3}$$
$$l_2 := \sqrt{l_1} \tag{3.4}$$
$$l_3 := x_2 l_2 \tag{3.5}$$
$$l_4 := u_2^2 \tag{3.6}$$
$$x_1^+ := l_2 - u_2 + u_1 \tag{3.7}$$
$$x_2^+ := l_3 + l_4 \tag{3.8}$$

Interfaces are analogous to the blue box in Figure 3.1 and the blocks it contains. They are memoryless and encode the system's transition relation. Two interfaces are connected in series if the output of one interface feeds into the input of another, such as the gain and square root interfaces. Two components are connected in parallel if neither output is connected to the other interface's input, such as the summation components. Composing smaller interfaces by connecting them yields another larger interface.

The unit delay blocks outside the blue box are not considered interfaces. They only appear to break any algebraic loops and introduce state in the context of controller synthesis, but they are not required for encoding the dynamics. Moreover, they can be mimicked for any finite time horizon through series composition of the dynamics with itself. This is analogous to function composition $x[k+2] = f(f(x[k], u[k]), u[k+1])$.

Motivation for Logical System Representation

Numerous issues arise when taking a set-based formulation from the previous chapter and immediately above. For instance, the assignment operations (3.3) - (3.8) are not robust to arbitrary rearrangements and place requirements on the execution order. It's unclear what happens when $x_1 < 0$, which causes the square root's output to be undefined and for the interface to block. Even if the square root's input were non-negative, its implementation may output either a negative or positive value because there are two choices for l_2 to satisfy $l_2^2 = l_1$.

We transition to a logical framework to cope with these issues. Predicates can accommodate both non-determinism and undefined outputs in a unified notation. They are functions that output a Boolean value and can be interpreted as set indicated function or as constraints to be satisfied. We can replace $x^+ := f(x, u)$ above with a predicate representation $F(x, y, x^+)$ which only accepts those values of x, u, x^+ where there exists an assignment to l_2 that satisfies each of the following constraints:

$$\begin{aligned} x_1^+ &== l_2 - u_2 + u_1 \\ x_2^+ &== x_2 l_2 + u_2^2 \\ l_2 &\in \{|\sqrt{3x_1}|, -|\sqrt{3x_1}|\} \\ x_1 &\geq 0. \end{aligned}$$

Notation

Relational interfaces are expressed as logical predicates. We briefly introduce predicates and the operations used to manipulate them; a formal introduction is provided in [46]. Let $\top$ denote logical true and $\bot$ denote false. Operators $\neg, \wedge, \vee$ respectively represent negation, conjunction, and disjunction. The implication $a \Rightarrow b$ is shorthand for the formula $\neg a \vee b$. The standard $=$ asserts a statement that two objects are mathematically equivalent; set equivalence $\equiv$ is used when those two are sets. In contrast, the operator $==$ checks whether two objects are equivalent, returning true if they are and false otherwise. A special instance of $==$ is logical equivalence $\Leftrightarrow$.

Variables are denoted by lower case letters. Each variable v is associated with a domain of values $\mathcal{D}(v)$ that is analogous to the variable's type. A composite variable is a set of variables and is analogous to a bundle of wrapped wires. From a collection of variables $v_1, \ldots, v_M$ a composite variable v can be constructed by taking the union $v \equiv v_1 \cup \ldots \cup v_M$ and the domain $\mathcal{D}(v) \equiv \prod_{i=1}^{M} \mathcal{D}(v_i)$. Note that the variables $v_1, \ldots, v_M$ above may themselves be composite. As an example if v is associated with an M-dimensional Euclidean space $\mathbb{R}^M$, then it is a composite variable that can be broken apart into a collection of atomic variables $v_1, \ldots, v_M$ where $\mathcal{D}(v_i) \equiv \mathbb{R}$ for all $i := 1, \ldots, M$. The technical results herein do not distinguish between composite and atomic variables.

Predicates are functions that map variable assignments to a Boolean value. Predicates that stand in for expressions are denoted with capital letters. Boolean valued expressions like "$x \in \{4, 5, 12\}$" and "$y == \sin(x)$" are predicates. The variables contained in those expressions are unassigned in the sense that they are not associated with a single value. Once all of a predicate's variables are assigned it returns a Boolean value. Predicates without full variable assignments yield newer predicates, e.g. assigning $y = 1$ in "$(y == \sin(x))$" yields the predicate "$(1 == \sin(x))$". Assignment of a composite variable $v \equiv v_1 \cup \ldots \cup v_M$ means that every v_i is assigned to an element in $\mathcal{D}(v_i)$.

Predicates can construct sets via set builder notation. A single predicate can instantiate different sets if the domains differ, e.g. $\{x \in \mathcal{D}(x) | P(x)\}$ and $\{(x, y) \in \mathcal{D}(x) \times \mathcal{D}(y) | P(x)\}$ are distinct sets but associated with the same predicate.

The standard Boolean operations can be applied to a predicate's Boolean output to construct new predicates. The negated predicate $\neg P(v)$ is true for an assignment to v if and only if $P(v)$ is false. Predicates P and Q are logically equivalent (denoted by $P \Leftrightarrow Q$) if and only if $P \Rightarrow Q$ and $Q \Rightarrow P$ are true for all variable assignments. The universal quantifier $\forall$ and existential quantifier $\exists$ eliminate variables and yield new predicates. For instance, $\exists w P$ and $\forall w P$ are predicates that do not depend on w. Existential quantification is analogous to projecting a set to a lower dimensional domain. If the variable w is actually a composite variable $w \equiv w_1 \cup \ldots \cup w_N$ then $\exists w P$ is simply a shorthand for $\exists w_1 \ldots \exists w_N P$.

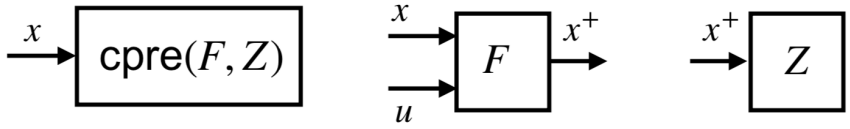


Figure 3.2: Three examples of relational interfaces with labeled inputs and outputs. The dynamics interface F takes the current state x and controllable input u as inputs. It outputs a subsequent state x^+ that is chosen non-deterministically. The non-determinism is constrained by the the pair (x, u) and represents uncertainty in the dynamics model. Sink interface Z encodes a collection of values to x^+ that do not block. An invalid assignment to x^+ would cause an error. The sink $\texttt{cpre}(F, Z)$ is constructed from F and Z via the relational interface operator $\texttt{cpre}(\cdot)$.

3.1 Relational Interfaces

Relational interfaces are predicates augmented with annotations about each variable's role as an input or output. They abstract away a component's internal implementation and only encode an input-output relation that characterizes that component's behaviors.

Definition 6 (Relational Interface)**.** *A relational interface consists of a predicate M, a set of input variables i, and a set of output variables o. We denote this triple as $M(i, o)$.*

For an interface $M(i, o)$, we call (i, o) its input-output *signature*. An interface is a sink if it contains no outputs and has signature like $(i, \varnothing)$, and a source if it contains no inputs with signature $(\varnothing, o)$. Sinks and sources can be interpreted as sets, while generic interfaces can be interpreted as relations. Figure 3.2 depicts three interfaces that arise during control synthesis.

Because interfaces effectively encode relations through their predicates, they can capture features such as blocking inputs (i.e., disallowed inputs or inputs that cause errors) or non-deterministic outputs. Certain assignments to the input variables may also cause system executions to block, which occurs for that input if there does not exist an output that satisfies the interface's input-output relation. Blocking is a critical property that we leverage to declare requirements. Sink interfaces can impose constraints through non-blocking inputs. That is, any input assignment that causes a sink to block violates the constraint. Outputs on the other hand exhibit non-determinism, which will be treated as an adversary. When one interface's outputs are connected to another interface's inputs (done with an operator that will soon be introduced), the outputs seek to cause blocking whenever possible. This is reminiscent of the game interpretation of control synthesis.

The original motivation for relational interfaces was as a compositional design language to reason about software and hardware requirements. While relational interfaces can be used to model control systems and constraints, the theory is oblivious to concepts such as controllable input and control system state. We view this as a powerful feature, rather than a shortcoming, because it allows atomic operators to be reused in a variety of scenar-

ios. To provide useful results however, control synthesis algorithms should only implement semantically meaningful sequences of operators.

3.2 Atomic Operators

Operators manipulate relational interfaces. They take existing interfaces and interface variables as inputs and transform them into another interface. Composite operators can also be defined by combining existing ones.

The first operator, output hiding, removes interface outputs.

Definition 7 (Output Hiding [90])**.** *The output hiding operator* $\texttt{ohide}(w, M)$ *over interface* $M(i, o)$ *and set of outputs* w *yields an interface with signature* $(i, o \setminus w)$*.*

$$\texttt{ohide}(w, M) = \exists w M \tag{3.9}$$

Existentially quantifying out w ensures that the input-output behavior over the unhidden variables is still consistent with potential assignments to w. The operator $\texttt{nb}(\cdot)$ is a special variant of $\texttt{ohide}(\cdot)$ that hides all outputs, yielding a sink encoding all non-blocking inputs to the original interface.

Definition 8 (Nonblocking Inputs Sink)**.** *Given an interface* $F(i, o)$*, the nonblocking operator* $\texttt{nb}(F)$ *yields a sink interface with signature* $(i, \varnothing)$ *and predicate*

$$\texttt{nb}(F) = \exists o F. \tag{3.10}$$

If $F(i, \varnothing)$ *is a sink interface, then* $\texttt{nb}(F) = F$ *yields itself. If* $F(\varnothing, o)$ *is a source interface, then* $\texttt{nb}(F) = \bot$ *if and only if* $F \Leftrightarrow \bot$*; otherwise* $\texttt{nb}(F) = \top$*.*

The interface composition operator takes multiple interfaces and "collapses" them into a single input-output interface. It can be viewed as a generalization of function composition in the special case where each interface encodes a total function (i.e., deterministic output and inputs never block). [1]

Definition 9 (Robust Interface Composition)**.** *Let* $M_1(i_1, o_1)$ *and* $M_2(i_2, o_2)$ *be interfaces with disjoint output variables* $o_1 \cap o_2 \equiv \varnothing$ *and*

$$i_1 \cap o_2 \equiv \varnothing \tag{3.11}$$

signifying that outputs of interface M_2*'s may not be fed back into inputs of* M_1*. Define new composite variables*

$$io_{12} \equiv o_1 \cap i_2 \tag{3.12}$$

$$i_{12} \equiv (i_1 \cup i_2) \setminus io_{12} \tag{3.13}$$

$$o_{12} \equiv o_1 \cup o_2 \tag{3.14}$$

[1] Note that the interface composition operator is distinct from operator composition. The former takes interfaces as inputs while the latter is a higher-order operation analogous to function composition.

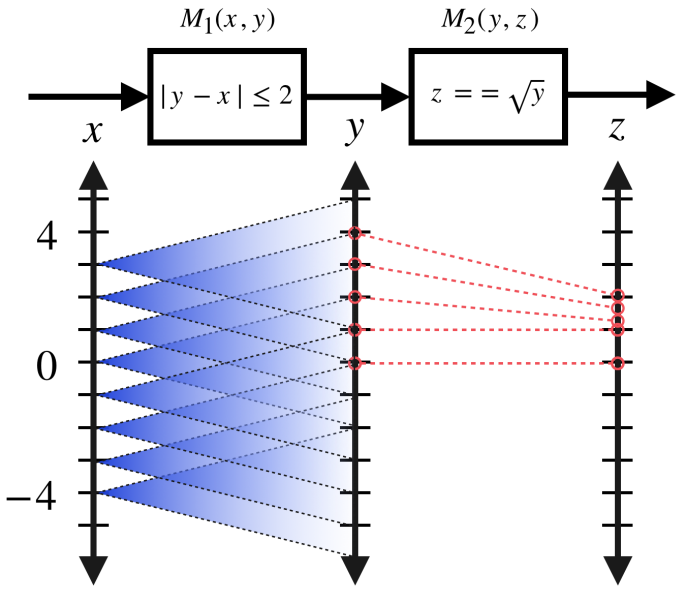


Figure 3.3: Visualization of the propagation of nonblocking inputs under series composition from Definition 9. The interfaces from Example 1 are used. Interface M_2's nonblocking set $\texttt{nb}(M_2) = (y \geq 0)$ is a subset of the possible outputs of M_1 which is the entire real line. Input values $x < 2$ to M_1 may lead M_2 to block, even though M_1 wouldn't have blocked otherwise. The additional constraint $\forall o_{12}(M_1 \Rightarrow \texttt{nb}(M_2)) = (x \geq 2)$ prunes inputs to M_1 that could induce M_2 to block under an adversarial and nondeterministic output of M_1.

and the composed interface

$$\textit{comp}(M_1, M_2) = M_1 \wedge M_2 \wedge \forall o_{12}(M_1 \Rightarrow \textit{nb}(M_2)) \tag{3.15}$$

with signature (i_{12}, o_{12}). *The interface subscripts may be swapped if instead the outputs of* M_2 *are fed into* M_1.

We say that M_1 and M_2 were composed in parallel if $io_{12} \equiv \varnothing$ holds in addition to Equation (3.11). Equation (3.15) under parallel composition reduces down to $M_1 \wedge M_2$ (Lemma 6.4 in [90]) and the composition operator is both commutative and associative. If $io_{12} \not\equiv \varnothing$, then the interfaces are composed in series and the composition operator is only associative. Figure 3.7 depicts the series composition of F and Z. If each interface has deterministic outputs and represents a function, function composition is a special instance of series composition of interfaces. Any acyclic interconnection can be composed into a single interface through a systematic application of Definition 9's binary composition operator.

Non-deterministic outputs are interpreted to be *adversarial*. Series composition of interfaces has a built-in notion of robustness to account for M_1's non-deterministic outputs and blocking inputs to M_2 over the shared variables io_{12}. The series composition changes the role of variables $io_{12} \subseteq i_2$ from inputs to outputs. The term $\forall o_{12}(M_1 \Rightarrow \texttt{nb}(M_2))$ in Equation (3.15) is a predicate over the composition's input set i_{12}. It ensures that if a potential

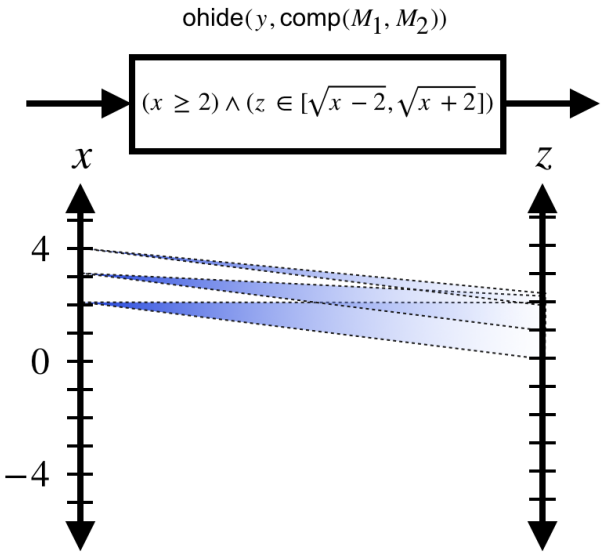


Figure 3.4: Depiction of $\texttt{ohide}(y, \texttt{comp}(M_1, M_2))$ with M_1 and M_2 from Figure 3.3. The new interface only input-output assignments to x and z that satisfy the expression $(x \geq 2) \wedge z \in [\sqrt{x-2}, \sqrt{x+2}]$.

output of M_1 may cause M_2 to block, then $\texttt{comp}(M_1, M_2)$ must preemptively block.

$$\texttt{nb}(\texttt{comp}(M_1, M_2)) = \exists o_{12}(M_1 \wedge M_2) \wedge \forall o_{12}(M_1 \Rightarrow \texttt{nb}(M_2)) \tag{3.16}$$

Example 1 below explains the role of the right-most term for a series composition interfaces and is accompanied by Figure 3.3 and 3.4.

Example 1. *Consider an interface $M_1(x, y)$ with predicate $M_1(x, y) = (|y - x| \leq 2)$, which feeds into an interface $M_2(y, z)$ with predicate $M_2(y, z) = (z == \sqrt{y})$. M_2's nonblocking inputs $\boldsymbol{nb}(M_2)(y)$ are $(y \geq 0)$. Substituting into the term $\exists o_{12}(M_1 \wedge M_2)$ from Equation (3.16) yields two equivalent expressions*

$$\begin{aligned} &\exists y \exists z (|y - x| \leq 2 \wedge z == \sqrt{y}) \\ &\quad \Leftrightarrow (x \geq -2) \end{aligned}$$

because for any $x \geq -2$ the assignments $y := x + 2$ and $z := \sqrt{x+2}$ satisfy the expression. However the series composition is not robust to an adversarial assignment to y. For instance, $x = -1, y = -1$ satisfy M_1's constraints but $y = -1$ is not a valid input to M_2. Substituting into the term $\forall o_{12}(M_1 \Rightarrow \boldsymbol{nb}(M_2))$ from (3.16) yields a tighter constraint on inputs.

$$\begin{aligned} &\forall y \forall z (|y - x| \leq 2 \Rightarrow y \geq 0) \\ &\quad \Leftrightarrow \forall y (|y - x| > 2 \vee y \geq 0) \\ &\quad\quad \Leftrightarrow (x \geq 2). \end{aligned}$$

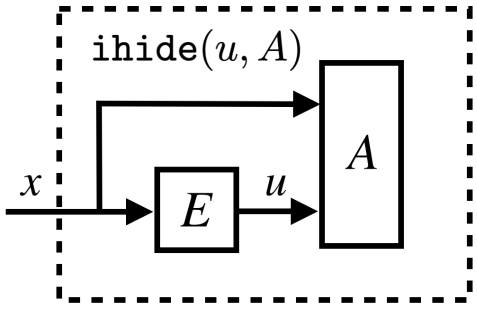


Figure 3.5: Interpretation of $\texttt{ihide}(\cdot)$ operator as an angelic environment E that chooses a variable assignment to u in reaction to a variable assignment to x. Note that $\texttt{ihide}(u, A)$ and $\texttt{ohide}(u, E)$ are equivalent interfaces.

Any input $x < 2$ is disallowed because there exists a strictly negative y that satisfies $|y - x| \leq 2$. Thus,

$$\begin{aligned} \texttt{comp}(M_1, M_2) &= (|y - x| \leq 2) \wedge (z == \sqrt{y}) \wedge (x \geq 2) \\ \texttt{ohide}(y, \texttt{comp}(M_1, M_2)) &= (x \geq 2) \wedge z \in [\sqrt{x-2}, \sqrt{x+2}] \\ \texttt{nb}(\texttt{comp}(M_1, M_2)) &= (x \geq 2) \end{aligned}$$

The final atomic operator is input hiding, which may only be applied to sinks. If the sink is viewed as a constraint, an input variable is "hidden" by an angelic environment that chooses an input assignment to satisfy the constraint. This operator is analogous to projecting a set into a lower dimensional space.

Definition 10 (Hiding Sink Inputs)**.** *The input hiding operator $\texttt{ihide}(w, M)$ over sink interface $M(i, \varnothing)$ and inputs w yields an interface with signature $(i \setminus w, \varnothing)$.*

$$\texttt{ihide}(w, M) = \exists w M \tag{3.17}$$

Unlike the composition and output hiding operators, this operator is not included in the standard theory of relational interfaces [90] and was added to encode a controller predecessor introduced subsequently in Equation (3.21).

Useful Composite Operators

One extremely useful operator is to swap one interface variable with another.

Definition 11 (Input and Output Renaming)**.** *Consider an interface M with signature (i, o). The input renaming operation to swap i for $\hat{i}$ is defined as*

$$\texttt{irename}(M, i, \hat{i}) = \texttt{ohide}(i, \texttt{comp}(Q_i, M)) \tag{3.18}$$

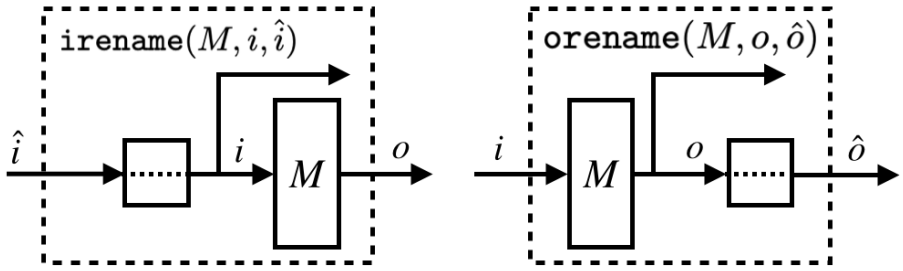


Figure 3.6: Inputs and outputs can be renamed by composing an identity interface and hiding the original variable.

where $Q_i(\hat{i}, i) = (i == \hat{i})$ *is an identity interface. Similarly, the output renaming operation to swap* o *for* $\hat{o}$ *is defined as*

$$\texttt{orename}(M, o, \hat{o}) = \texttt{ohide}(o, \texttt{comp}(M, Q_o)) \tag{3.19}$$

where $Q_o(o, \hat{o}) = (o == \hat{o})$ *is an identity interface.*

Renaming in practice is often quicker to compute if implemented directly rather than via combining the `ohide`$(\cdot)$ and `comp`$(\cdot)$ operators. This can be implemented by swapping variables in the predicate representation.

Another useful composite operator is domain filtering, which removes values from an interface's input domain. It reduces an interface's input domain by composing it in parallel with a sink interface that encodes a smaller domain. The operation is a special case of `comp`$(\cdot)$ that imposes additional requirements about the interface signatures.

Definition 12 (Domain Filtering)**.** *Consider an interface* $M(i, o)$ *and a sink interface* $I(j, \varnothing)$ *that acts as a filter. Let the input variable sets satisfy* $j \subseteq i$*, meaning that the filter has fewer inputs. The domain filtering operation*

$$\texttt{dfilter}(I, M) = \texttt{comp}(I, M) \tag{3.20}$$

discards any input-output assignments to M *that violate* I*.*

3.3 Control Synthesis as Robust Interface Composition

Recall the set-based definition of the controlled predecessor from Equation (2.3)

$$\begin{aligned}\texttt{cpre}(F, Z) &= \{x : \exists u \text{ such that } \varnothing \neq F(x, u) \subseteq Z\} \\ &= \left\{ x : \exists u \text{ such that } \begin{array}{c} \exists x^+ \text{ such that } x^+ \in F(x, u) \\ \text{and} \\ x^+ \in Z \text{ holds } \forall x^+ \in F(x, u) \end{array} \right\}\end{aligned}$$

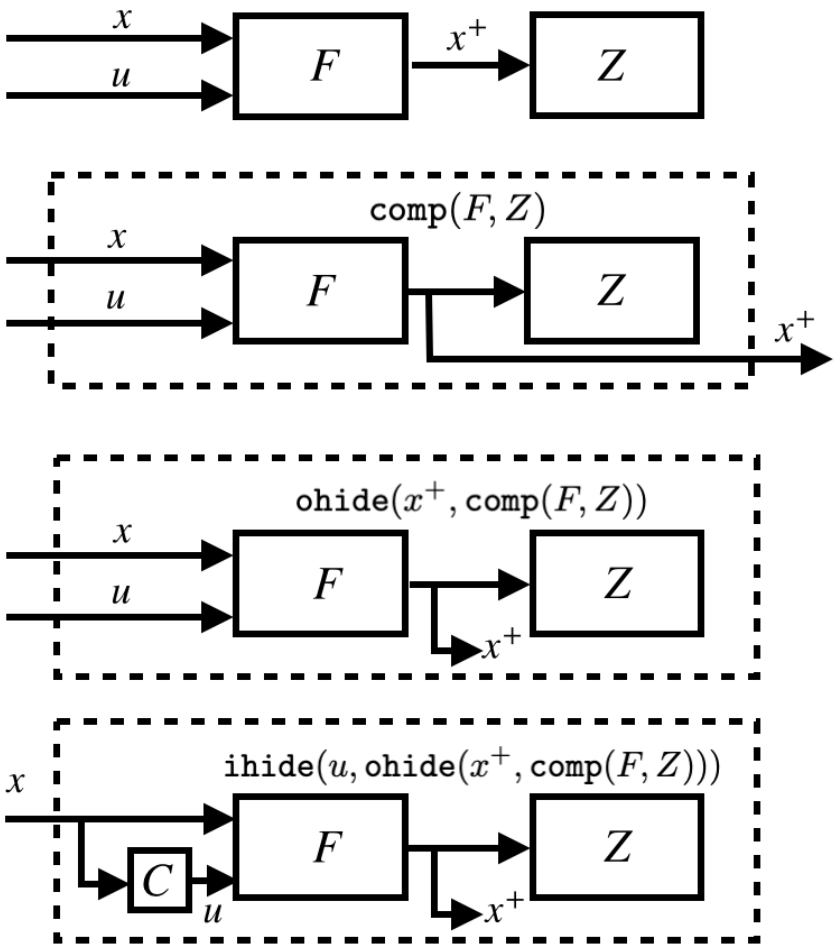


Figure 3.7: Control predecessor as a sequence of interface operations. Interface $F(x \cup u, x^+)$ represents system dynamics and sink interface $Z(x^+, \varnothing)$ represents a target set of states at the next time step. The C interface represents a controller that takes the state x as input and constrains the potential control values u.

Translating this to a logical expression yields the expression that encodes a predicate condition on the variables x, u, and x^+.

$$\exists u(\exists x^+ F \wedge \forall x^+(F \Rightarrow Z))$$

We now have all of the necessary atomic operators to express $\texttt{cpre}(F, Z)$ through a sequence of relational interface operators, whose collective effect is depicted visually in Figure 3.7.

Proposition 1. *The controlled predecessor* $\texttt{cpre}(F, Z)$ *from Equation* (2.3) *is equivalent to*

$$\texttt{cpre}(F, Z) = \texttt{ihide}(u, \texttt{ohide}(x^+, \texttt{comp}(F, Z))) \tag{3.21}$$

Proof. Applying the definitions of $\texttt{comp}(\cdot)$, $\texttt{ihide}(\cdot)$, and $\texttt{ohide}(\cdot)$ yields the expression (3.22). One can safely move the $\exists x^+$ inside the parenthesis as in (3.23) because $\forall x^+(F \Rightarrow Z)$ is a predicate that is independent of x^+.

$$\exists u \exists x^+(F \wedge Z \wedge \forall x^+(F \Rightarrow Z))) \tag{3.22}$$
$$= \exists u(\exists x^+(F \wedge Z) \wedge \forall x^+(F \Rightarrow Z))) \tag{3.23}$$

To show equivalence of the above expressions with Equation (2.3), we simply need to show equivalence of the following two predicates that depend on x and u:

$$\exists x^+(F \wedge Z) \wedge \forall x^+(F \Rightarrow Z) \tag{3.24}$$
$$\exists x^+ F \wedge \forall x^+(F \Rightarrow Z). \tag{3.25}$$

It is easy to see that (3.24) implies (3.25) because $\exists x^+(F \wedge Z)$ implies $\exists x^+ F$. To show the reverse, suppose (3.25) is satisfied for a pair x and u. Any x^+ chosen to satisfy $\exists x^+ F$ must also satisfy the constraint imposed by the sink Z. Otherwise, the clause $\forall x^+(F \Rightarrow Z)$ would be violated, contradicting satisfaction of (3.25). Therefore, (3.24) and (3.25) are equivalent which completes the proof. □

Both the safety game and reach game are possible to encode with relational interface operators.

Definition 13 (Safety Game)**.** *Given a safety set interface $S(x, \varnothing)$ and dynamics F, the safety fixed point is defined as the fixed point of the iteration*

$$Z_0 = S \tag{3.26}$$
$$Z_i = \texttt{safe}(F, Z_i, S) = \texttt{comp}(\texttt{cpre}(F, Z_{i-1}), S) \tag{3.27}$$

Let Z_∞ denote the fixed point of the iteration, which occurs when $Z_i == \texttt{safe}(F, Z_i, S)$ holds. If such a fixed point exists, then it can be viewed as a subset of the state space where a controller can force closed loop system behaviors to lie in S forever. It is an invariant set. The iterations of a safety game can also be viewed as a collection of interfaces generated by applying relational interface operators, as depicted in Figure 3.8. The composition operation over sinks encodes a conjunction in Equation (3.27). Recalling that sink interfaces correspond to requirements to be satisfied, one may interpret each of the S interfaces in Figure 3.8 as interfaces that monitor for constraint violations at every time step. This can be viewed as an instance of domain filtering as in Definition 12. That is, the $\texttt{comp}(\cdot)$ in (3.27) can be substituted with $\texttt{dfilter}(\cdot)$.

To encode a reach game, we use another operator $\texttt{refine}(\cdot)$, which is introduced later in Definition 20 of Section 4.3. The $\texttt{refine}(\cdot)$ operation reduces down to set union when the inputted interfaces are sinks that encode sets.

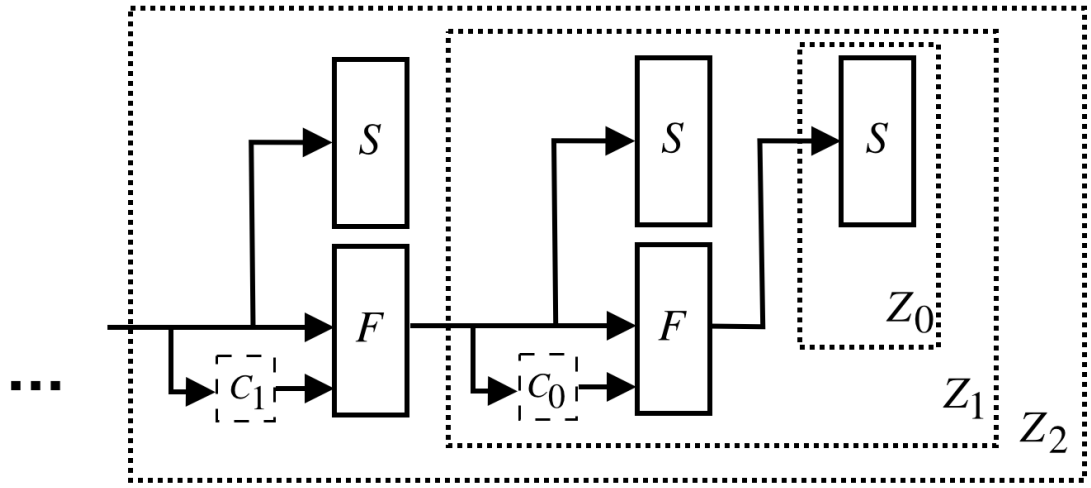


Figure 3.8: Depiction of safety game iteration as a collection of interfaces generated by repeated application of Equation (3.27). Starting from $Z_0 = S$, a sequence of sink interfaces Z_i is generated. This reflects each iteration of the safety game. The dynamics F and safety set S are depicted without an iteration/time index, which is appropriate if they are time invariant. In contrast, a different control interface C_i is constructed via the `ihide`$(\cdot)$ operator at each iteration. The control interfaces have dashed outlines because they are never explicitly constructed; we simply depict the affects of the `ihide`$(\cdot)$ operator.

Definition 14 (Reach Game)**.** *Given a target set interface $T(x, \varnothing)$ and dynamics F, the reach fixed point is defined as the fixed point of the iteration*

$$Z_0 = \bot \tag{3.28}$$
$$Z_i = \texttt{reach}(F, Z_i, T) = \texttt{refine}(\texttt{cpre}(F, Z_{i-1}), T) \tag{3.29}$$

The fixed point Z_∞ of the reach game (if it exists) can be viewed as a set of initial states for which a controller can cause the system to eventually enter T.

The reach-avoid game can be encoded in a similar manner to the safety and reach games. The reach-avoid fixed point represents the set of initial states for which a controller can cause the system to eventually enter T while never exiting the safe region S beforehand.

Definition 15 (Reach-Avoid Game)**.** *Given a target set T, safety set S, and dynamics F, the solution to the reach-avoid game is defined as the fixed point of the iteration*

$$Z_0 = \bot \tag{3.30}$$
$$Z_i = \texttt{reachavoid}(F, Z_i, S, T) = \texttt{refine}(\texttt{comp}(\texttt{cpre}(F, Z_{i-1}), S), T) \tag{3.31}$$

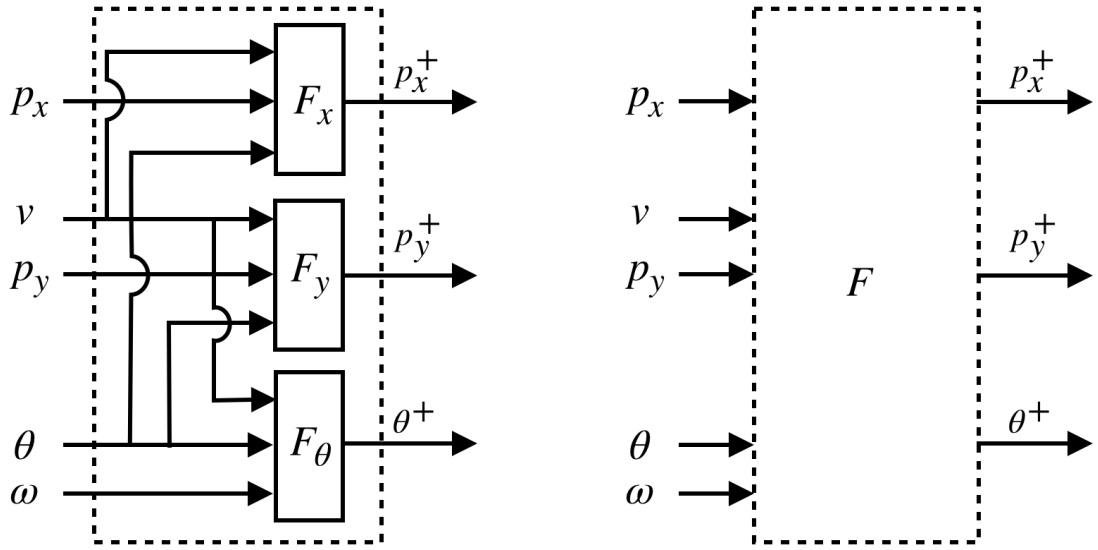


Figure 3.9: Decomposed dynamics for a Dubins vehicle. The interface F represents the monolithic dynamics (right) and is the result of a parallel composition of three internal components (left).

3.4 Compositional Models and Decomposed Control Predecessors

The game iterations above all rely on computing $\texttt{cpre}(\cdot)$, which so far has been defined for a monolithic system F. Many large systems are actually constructed as a collection of smaller components. One of the most common structural properties exhibited by continuous domain control systems are a state space encoded as a Cartesian product and dynamics characterized by a collection of component-wise updates. Consider a collection of state variables $x \equiv \{x_1, \ldots, x_N\}$ and $x^+ \equiv \{x_1^+, \ldots, x_N^+\}$ and $u \equiv \{u_1, \ldots, u_M\}$. The dynamics $F(x, u, x^+)$ could be denoted by some collection of smaller components to update each state variable x_i^+ for $i = 1, \ldots, N$. This decomposition can be encoded as a simple parallel composition of systems with $\texttt{comp}(\cdot)$ and is logically equivalent to

$$F(x, u, x^+) = \bigwedge_{i=1}^{N} F_i(x, u, x_i^+). \tag{3.32}$$

In the language of relational interfaces, we can construct the monolithic interface from Equation (3.32) as a parallel composition $F = \texttt{comp}(F_1, \ldots, F_N)$. An example of such a monolithic interface is depicted in Figure 3.9 for the Dubins vehicle.

A decomposed control predecessor avoids computing the monolithic system altogether. Instead, it rearranges itself to reflect the system's decomposition structure. The original

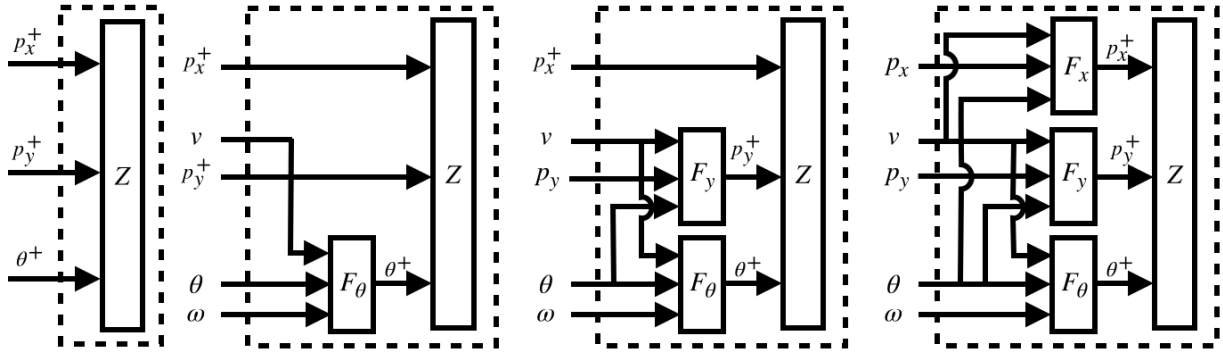


Figure 3.10: Visualization of a decomposed control predecessor for the decomposed Dubins vehicle dynamics in Figure 3.9. A sequence of interfaces that ultimately yields an interface that is equivalent to the monolithic $\texttt{ohide}(x^+, \texttt{comp}(F, Z))$. From the initial sink interface $Z(p_x^+ \cup p_y^+ \cup \theta^+, \varnothing)$ in the far left, the center left interface $\texttt{ohide}(\theta^+, \texttt{comp}(F_\theta, Z))$ is obtained by composing with F_θ and hiding θ^+. Composing and hiding the F_y interface and p_y^+ variable yields the center right interface. The far right interface is equivalent to the monolithic $\texttt{ohide}(x^+, \texttt{comp}(F, Z))$.

controlled predecessor

$$\texttt{cpre}(F, Z) = \texttt{ihide}(u, \texttt{ohide}(x^+, \texttt{comp}(F, Z)))$$

was given for a monolithic interface F encoding the control system dynamics. To use this existing $\texttt{cpre}(\cdot)$ for system like (3.32), one could take the interfaces $F_1, \ldots, F_N$, compute their parallel composition, and substitute for F above. Even with the component-wise abstraction method described above, this could be computationally expensive. Constructing F is also premature because knowledge of the specification can drastically simplify the computation of a controller. Experimental evidence (albeit in other domains) suggests that full knowledge of the system dynamics may be unnecessary once a task is fixed [6].

The decomposed control predecessor avoids constructing F by instead using an alternative to $\texttt{cpre}(\cdot)$ that leverages the decomposition structure. This occurs by breaking apart the term $\texttt{ohide}(x^+, \texttt{comp}(F, Z))$. The key intuition is provided in Figure 3.10 on the Dubins vehicle example. More generally, for a system composed in parallel $\texttt{ohide}(x^+, \texttt{comp}(F, Z))$ can instead be replaced with the equivalent representation

$$\texttt{ohide}(x_1^+, \texttt{comp}(F_1, \ldots, \texttt{ohide}(x_N^+, \texttt{comp}(F_N, Z)) \tag{3.33}$$

that avoids constructing F. Equivalence of the decomposed controlled predecessor with the original monolithic one can be shown via the associativity of $\texttt{comp}(\cdot)$ and the fact that outputs $x^+ \equiv \{p_x^+, p_y^+, \theta\}$ are not shared across interfaces.

Empirical runtimes for the Dubins vehicle example appear later in Section 5.7. They demonstrate that preemptively constructing the monolithic system dynamics F is unnecessary and increases synthesis runtimes.

Domain Filtering

Domain filtering is an operation that eliminates an interface's input-output information for a collection of input values. This operation is especially useful if one knows beforehand that some input-output assignments to an interface are superfluous or unnecessary. Modeling the system dynamics for a global state space is useful if the specification is unknown but may be unnecessary for specific problem instances.

1. **Embedding Safety into the Dynamics:** For safety games, a certain region of the state space is known to be unsafe so modeling the dynamics in that region is unnecessary. `SCOTS` [78] incorporates this insight during the abstraction step by allowing the user to preemptively reject any input samples that violate the safety constraint. This approach is reasonable if the safety constraint is known beforehand and static, but comes at the cost of flexibility. The `SCOTS` approach would not work if the safety region were for instance time varying. Recall Figure 3.8's depiction of the safety game. It shows the system F is always composed in parallel with the (in this case static) safety constraint S. Equation (3.27) encoded the safety game operator $\texttt{safe}(F, Z_i, S) = \texttt{dfilter}(\texttt{cpre}(F, Z_{i-1}), S)$ which indicates that the safe region S is composed with $\texttt{cpre}(F, Z_{i-1})$ after it is computed. The domain filtering method rearranges this computation to be $\texttt{cpre}(\texttt{dfilter}(F, S), Z_{i-1})$. This iteration has no notion of safety but the safe set is hard coded into the dynamics via $\texttt{dfilter}(F, S)$.

2. **Filtering with Lifted Control Predecessor Projections:** An approach appearing in [48] combines domain filtering with notions from the decomposed control predecessor. Each interface that encodes a portion of the dynamics is filtered by a set that overapproximates the potential predecessor states. This set is generated from the game iteration interface Z_i and its construction is best explained by example. The interface F_x from the Dubins vehicle is filtered through these three steps:

   a) Project the current sink interface Z_i with signature $(p_x^+ \cup p_y^+ \cup \theta^+, \varnothing)$ to a lower dimension interface Z_i^x with the input hiding operator $\texttt{ihide}(p_y^+ \cup \theta^+, Z_i)$. This sink interface's signature is $(p_x^+, \varnothing)$. The variables $\{p_y^+, \theta^+\}$ are hidden because they are not outputted by F_x.

   b) Compute the filter interface with $\texttt{ohide}(p_x^+, \texttt{comp}(F_x, Z_i^x))$. Any input that is accepted by this filter satisfies a necessary condition for satisfaction of Z_i.

   c) Apply the domain filter to F_x, eliminating state-input pairs that violate the filter's necessity condition.

   The steps above also generalize to construct domain filtered versions of F_y and F_θ.

3.5 Reducing the Complexity of the Control Synthesis Pipeline

The controlled predecessor $\texttt{cpre}(\cdot)$ appearing in Equation (3.21) is oblivious to the domains of variables x, u, and x^+. This generality is useful for describing a problem and serving as a blank template, but lacks sufficient details to translate it into an algorithmic implementation. In existing control synthesis implementations, many structural assumptions are implicitly encoded in the tool's design choices. Whenever problem structure exists, pipeline modifications refine the general control synthesis algorithm into a form that reflects the specific problem instance. They also allow a user to explicitly inject preferences into a problem and reduce computational bottlenecks or to refine a solution.

The decomposed control predecessor and domain filtering techniques above yield the same result as the original pipeline. They manage to reduce computational complexity by rearranging the algorithmic procedure used to compute the result. While this approach yields noticeable benefits, it ultimately runs into performance bottlenecks if the original problem exhibits inherent memory lower bounds. The next chapter introduces the dual notions of abstraction and refinement. Abstraction reduces a complicated problem into its core, while refinement introduces additional details. Using these two allows one to modulate the complexity of the problem and the conservatism of the synthesizer's output.

Chapter 4

Abstracting and Refining Control Systems

Model complexity implicitly encodes a tradeoff between utility and tractability. Computing the intermediate game iterations from Equation (2.5) and Equation (2.7) exactly is often intractable for high-dimensional nonlinear dynamics. One method to construct tractable algorithms for control synthesis is through the notion of abstraction. At its essence abstraction entails removing unnecessary information and distilling a problem to its core. A good abstraction procedure captures a problem's salient structure while discarding extraneous information. For continuous systems tasked with satisfying a safety, reach, or reach-avoid objective, small perturbations of the dynamics or the target sets do not substantially alter the solution. Throwing away fine grained local information can be a reasonable trade-off if it leads to noticeable computational gains, but determining the effectiveness of this approach is domain specific.

In this chapter, we introduce the notion of a hierarchy of models and formalize what it means to systematically discard information. We define a special kind of interface called a quantizer that is an abstraction of the identity interface. The model hierarchy ensures that reasoning performed over abstract models can be extrapolated to their concrete counterparts. Traversing this hierarchy allows us to explore the tradeoffs between algorithmic tractability and fidelity of the control synthesizer's output. Chapter 5 applies the core theoretical framework in this chapter to the finite abstraction, synthesis, refinement pipeline from Figure 2.3.

4.1 Interface Refinement

We formalize the notion of an abstract interface. At its essence, it encodes the principle where abstract interfaces are more aggressive with blocking and exhibit more output nondeterminism. Abstractions are ideally simpler than the original interface, but are more conservative. Thus if a property can be established for an abstract interface, it should also

hold for the concrete interface as well. When a controller synthesis algorithm constructs a controller enforcing a property on abstract system dynamics, the results can be extrapolated or refined back to control the original system dynamics.

Definition 16 (Interface Refinement Relation)**.** *Let $M(i, o)$ and $\hat{M}(\hat{i}, \hat{o})$ be interfaces. $\hat{M}$ is an abstraction of M if and only if $i \equiv \hat{i}$, $o \equiv \hat{o}$, and substituting predicates M and $\hat{M}$ into*

$$\boldsymbol{nb}(\hat{M}) \Rightarrow \boldsymbol{nb}(M) \tag{4.1}$$

$$\Big(\boldsymbol{nb}(\hat{M}) \wedge M\Big) \Rightarrow \hat{M} \tag{4.2}$$

yields predicates equivalent to $\top$, i.e., they are satisfied for all variable assignments. This relationship is denoted by $\hat{M} \preceq M$.

Definition 16 imposes two main requirements between the concrete interface M and abstract interface $\hat{M}$. Equation (4.1) encodes the condition where if $\hat{M}$ accepts an input, then M must also accept it; that is, the abstract component is more aggressive with rejecting invalid inputs. Second, if both systems accept the input then the abstract output set is a superset of the concrete function's output set. The abstract interface is a conservative representation of the concrete interface because the abstraction accepts fewer inputs and exhibits more non-deterministic outputs. If both the interfaces are sink interfaces, then $\hat{M} \preceq M$ reduces down to $\hat{M} \subseteq M$ when $M, \hat{M}$ are interpreted as sets. If both are source interfaces then the set containment direction is flipped and $\hat{M} \preceq M$ reduces down to $M \subseteq \hat{M}$.

The refinement relation $\preceq$ encodes a direction of conservatism such that any reasoning done over the abstract models is sound and can be generalized to the concrete model.

Theorem 1 (Informal Substitutiability Result [90])**.** *For any input that is allowed for the abstract model, the output behaviors exhibited by an abstract model contains the output behaviors exhibited by the concrete model.*

If a property on outputs has been established for an abstract interface, then it still holds if the abstract interface is replaced with the concrete one. Informally, the abstract interface is more conservative so if a property holds with the abstraction then it must also hold for the true system.

The refinement relation satisfies the required reflexivity, transitivity, and antisymmetry properties to be a partial order [90]. Figure 4.1 depicts a collection of interfaces in this order and their relationships. This order has a bottom element $\perp$ which is a universal abstraction. Conveniently, the bottom element $\perp$ signifies both Boolean false and the bottom of the partial order. This interface blocks for all input values and always induces an error. In contrast, boolean $\top$ plays no special role in the partial order. This interface never blocks but introduces undesirable non-determinism into the system. While $\top$ exhibits totally non-deterministic outputs, it also accepts all inputs. A blocking input is considered worse than non-deterministic outputs in the refinement order.

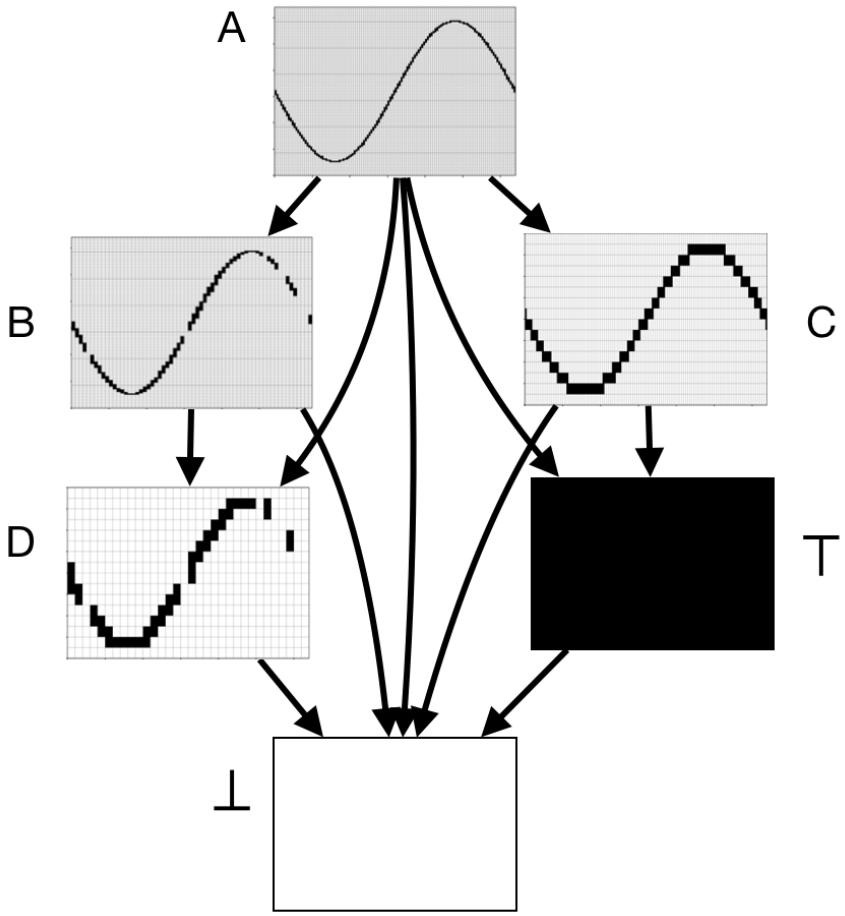


Figure 4.1: Example depiction of the refinement partial order. Each small plot on the left depicts darkened input-output pairs that satisfy an interface's predicate. Inputs (outputs) vary along the horizontal (vertical) axis. Because B blocks on some inputs but A accepts all inputs $B \preceq A$. Interface C exhibits more output non-determinism than A so $C \preceq A$. Similarly $D \preceq B$, $D \preceq C$, $\top \preceq C$, etc. Note that $B \not\preceq C$ and $C \not\preceq B$ because C exhibits more output non-determinism and B blocks for more inputs. The false interface $\bot$ is a universal abstraction. The true interface $\top$ plays no special role in this partial order and in fact is incomparable with interfaces B and D. Table 5.1 later shows how certain interfaces were obtained through coarsening.

All aforementioned interface operators preserve the properties of the refinement relation of Definition 16, in the sense that they are monotone with respect to the refinement partial order. Proofs for Theorem 2 and 3 below are provided in [90]. The proof for the input hiding operator is trivial and is omitted.

Theorem 2 (Composition Preserves Refinement)**.** *Let $\hat{A} \preceq A$ and $\hat{B} \preceq B$. If the composition is well defined, then $\texttt{comp}(\hat{A}, \hat{B}) \preceq \texttt{comp}(A, B)$.*

Theorem 3 (Output Hiding Preserves Refinement)**.** *If $A \preceq B$, then for any variable w*

$$\texttt{ohide}(w, A) \preceq \texttt{ohide}(w, B). \tag{4.3}$$

Theorem 4 (Input Hiding Preserves Refinement)**.** *If A, B are both sink interfaces and $A \preceq B$, then $\texttt{ihide}(w, A) \preceq \texttt{ihide}(w, B)$ for any variable w.*

One can think of using interface composition and variable hiding to horizontally (with respect to the refinement order) navigate the space of all interfaces. The synthesis pipeline encodes one navigated path and monotonicity of these operators yields guarantees about the path's end point. Composite operators such as $\texttt{cpre}(\cdot)$ chain together multiple incremental steps. Furthermore since the composition of monotone operators is itself a monotone operator, any composite constructed from these parts is also monotone. In contrast, the coarsening and refinement operators introduced later in Definition 18 and Definition 20 respectively are used to move vertically and construct abstractions. The "direction" of new composite operators can easily be established through simple reasoning about the cumulative directions of their constituent operators.

4.2 Interface Abstraction via Quantization

The core idea behind translating continuous dynamics into an approximate finite representation relies on discretizing both time and space. Many control synthesis tools construct discrete time system models from continuous time models by fixing a sampling time step and assume that controlled inputs have a zero order hold [76]. Spatial discretization or coarsening is achieved by use of a quantizer interface that implicitly aggregates points in a space into a partition or cover. It is possible to construct many different kinds of quantizer interfaces, and their implementations depend on the data structure to represent finite abstractions.

Definition 17 (Quantizer Interface)**.** *A quantizer $Q(i, o)$ is any interface that abstracts the identity interface $(i == o)$ associated with the signature (i, o).*

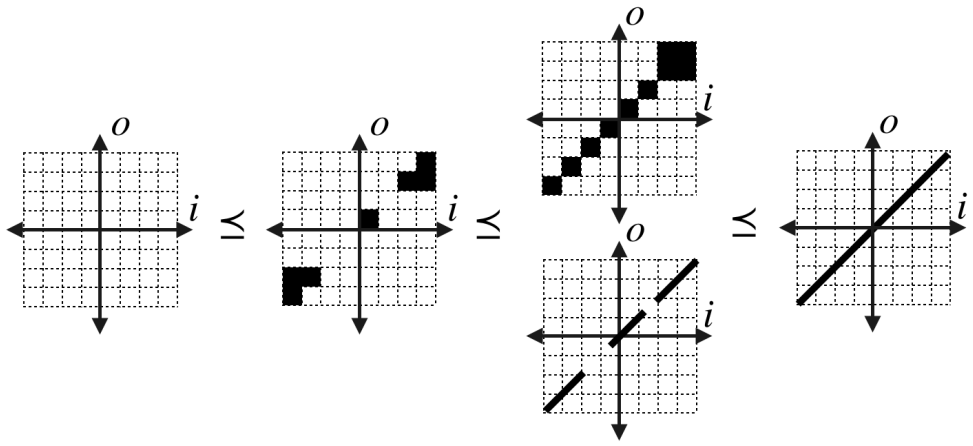


Figure 4.2: An identity interface associated with predicate $(i == o)$ is depicted at the very right and the universal interface $\perp$ at the left. The identity interface refines all other ones depicted above.

Figure 4.2 depicts some non-trivial interfaces that abstract the identity interface. The abstraction occurs either by increasing output non-determinism, by filtering the input domain, or some combination of both.

The coarsening operator takes an interface by connecting a quantizer interface in series with the target interface. Its main use is to decrease the complexity of the system representation and make synthesis more computationally tractable.

Definition 18 (Input and Output Coarsening)**.** *Consider an interface $M(i, o)$ and input quantizer $Q(\hat{i}, i)$. Input coarsening*

$$\texttt{icoarsen}(M, Q(\hat{i}, i)) = \texttt{ohide}(i, \texttt{comp}(Q(\hat{i}, i), M)) \tag{4.4}$$

yields a new interface with signature $(\hat{i}, o)$. Similarly, given an output quantizer $Q(o, \hat{o})$, output coarsening

$$\texttt{ocoarsen}(M, Q(o, \hat{o})) = \texttt{ohide}(o, \texttt{comp}(M, Q(o, \hat{o}))) \tag{4.5}$$

yields a new interface with signature $(i, \hat{o})$.

Coarsening reduces the number of non-blocking inputs and increases the output non-determinism. The $\texttt{ohide}(\cdot)$ is only required because the $\texttt{comp}(\cdot)$ series composition operator exposes the original variables (i/o for input/output hiding) as output variables to the new interface. The corollary below readily follows from the fact that quantizers abstract the identity interface, Theorem 2, and Theorem 3.

Corollary 1. *Input and output coarsening operations $\texttt{icoarsen}(\cdot)$ in (4.4) and $\texttt{ocoarsen}(\cdot)$ in (4.5) are monotone operations with respect to the interface refinement order $\preceq$.*

4.3 Constructing Abstractions through Reachable Set Overapproximations

Shared refinement [90] is an operation that takes two interfaces and merges them into a single interface. In contrast to coarsening, it makes interfaces more precise. It is extensively used in the abstraction step to convert dynamical systems models into a form that's amenable to the relational interface operations.

Black box functions, Simulink models, and source code files are all common representations of control systems, yet they do not readily lend themselves to the predicate operations used to encode the relational interface operators. Many tools get around this issue by constructing system abstractions by starting from the universal abstraction $\perp$, then iteratively refining it with a collection of smaller interfaces that represent input-output samples. The predicate operations over finite domains are guaranteed to be computable. This property is a key motivator behind abstracting transition relations over continuous spaces an approximate and finite counterpart. There are many data structures that could be used to represent finite domain predicates including lookup tables, binary valued n-dimensional arrays, trees, bitmaps, and Boolean circuits.

Directly translating the common control system representations into any of the above data structures is non-trivial. An alternative method is to first generate a collection of smaller interfaces that represent input-output samples, then construct a larger interface by iteratively merging (or collapsing) the collection.

Shared refinement is another atomic operator [90] that merges the information contained in multiple interfaces. Interfaces can be successfully merged whenever they do not contain contradictory information. The shared refinability condition below formalizes when such a contradiction does not exist.

Definition 19 (Shared Refinability [90])**.** *Let $M_1(i, o)$ and $M_2(i, o)$ be two interfaces that operate on the same set of inputs and outputs. We say that they are shared refinable if for all inputs i.*

$$(\boldsymbol{nb}(M_1) \wedge \boldsymbol{nb}(M_2)) \Rightarrow \exists o(M_1 \wedge M_2). \tag{4.6}$$

For any inputs that do not block for all interfaces, the corresponding output sets have a non-empty intersection. Figure 4.3 depicts two examples of interface pairs that do and don't meet the shared refinability condition.

Given a collection of shared refinable interfaces, they can be combined into a single one that encapsulates all of their information.

Definition 20 (Shared Refinement Operation [90])**.** *The shared refinement operation combines two shared refinable interfaces and yields a new interface corresponding to the predicate.*

$$(\boldsymbol{nb}(M_1) \vee \boldsymbol{nb}(M_2)) \wedge (\boldsymbol{nb}(M_1) \Rightarrow M_1) \wedge (\boldsymbol{nb}(M_2) \Rightarrow M_2) \tag{4.7}$$

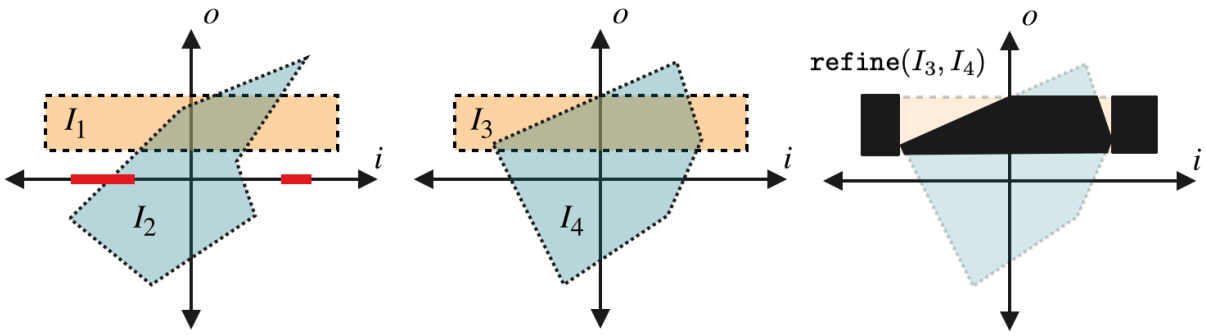


Figure 4.3: Example of pairs of interfaces that are and are not shared refineable. Inputs (outputs are depicted on the horizontal (vertical) axis. (Left) The interfaces are not shared refinable because for certain input assignments (highlighted in red) the sets of possible outputs are disjoint. The output sets contradict one another and the interfaces cannot be merged. (Center) For each input assignment that does not cause I_3 and I_4 to block, the corresponding output sets are not disjoint. (Right) Interface $\texttt{refine}(I_3, I_4)$ in black.

More generally given a set of shared refinable interfaces $M_1, \ldots, M_N$ we define:

$$\textit{refine}(M_1, \ldots, M_N) = \left(\bigvee_{i=1}^{N} \textit{nb}(M_i) \right) \wedge \bigwedge_{i=1}^{N} (\textit{nb}(M_i) \Rightarrow M_i) \tag{4.8}$$

The left term expands the set of accepted inputs. The right term signifies that if an input was accepted by multiple interfaces, the output must be consistent with each of them. Its effect is to use multiple interfaces to move up the refinement order as will be depicted in Example 4. The shared refinement operation reduces down to a disjunction for sink interfaces and an conjunction for source interfaces.

Theorem 5 (Shared Refinement Yields a Least Upper Bound [90])**.** *Given a collection of shared refinable interfaces, the shared refinement operation yields the least upper bound with respect to the refinement partial order in Definition 16.*

Violation of the shared refinability requirement Part 4.6 can be detected if the outputted interface is not refinement of the inputted ones. This is a useful check for ensuring that reach set overapproximations are computed correctly.

Reachable Set Overapproximations are Abstractions

A common method to construct finite abstractions is through simulation and overapproximation of forward reachable sets. This technique appears in tools such as PESSOA [60], SCOTS [78], MASCOT [44], ROCS [55] and ARCS [16]. By covering a sufficiently large portion of the interface input space, one can construct larger composite interfaces from smaller ones via shared refinement.

Sample interfaces are constructed in a sequence of steps.

1. **Input-Output Sampling:** Smaller interfaces are constructed by sampling regions of the input space, constructing the associated forward reachable set, then constructing an interface encoding a collection of input-output pairs. Restrict attention to a subset of the interface input space.

2. **Output Overapproximation:** Computing the exact forward reachable set may be intractable, in which case an over-approximation may be substituted while preserving the abstraction.

3. **Quantize:** If desired, quantize the results of the previous two steps so that the interface can be stored as a finite data structure.

Each of the previous steps take as input an interface and output an abstraction of that interface. The refinement step merges the information contained within many sample interfaces. The refinement step result is abstracted by all of the inputted sample interfaces.

4. **Refine:** Merge quantized samples to construct a larger interface that represents a quantization of the original function. Its execution can be interleaved with sample generation process; it does not need to wait for all samples to be generated before starting to merge.

Figure 4.4 depicts a dataflow graph where the above operations serve as methods to move down the interface partial order. The shared refinement operation combines multiple interfaces to construct an abstraction of the original concrete interface.

Input-Output Sampling

A sink interface $I(i, \varnothing)$ acts as a filter that prunes inputs outside of the relevant input region. The source interface $I'(\varnothing, i')$ composed with $F(i', o)$ generates a forward reachable set associated with the set of inputs. Both $I(i, \varnothing)$ and $I'(\varnothing, i')$ encode the same subset of the $\mathcal{D}(i) \equiv \mathcal{D}(i')$ space. The sampled interface is an *abstraction* in the sense that the original interface refines the sampled interface.

Proposition 2 (Input-Output Sample Interface)**.** *Let $I(i, \varnothing)$ be an interface such that*

$$I \Rightarrow \mathtt{nb}(F) \tag{4.9}$$

is valid. Define the renamed interface $F' = \mathtt{irename}(F, i, i')$. Similarly, let interface $O = \mathtt{ohide}(i', \mathtt{comp}(I', F'))$ be a source that encodes the collection of potential output assignments from the collection of input assignments that satisfy I. Input-output interface $IO(i, o) = \mathtt{comp}(I, O)$ corresponds with the predicate $I \wedge O$ because the two are composed in parallel. then $IO \preceq F$ and the sample interface IO is an abstraction of F.

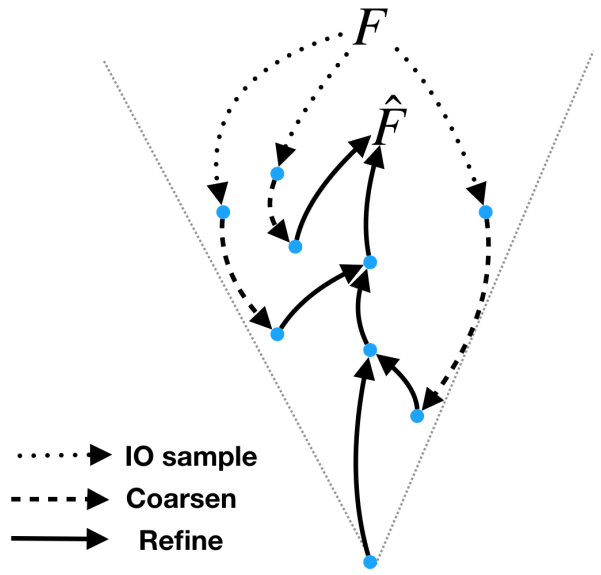


Figure 4.4: Visualization of finite abstraction algorithms as a dataflow graph traversing the refinement partial order. Nodes represent interfaces and edges signify data dependencies for interface manipulation operators. Multiple refine edges point to a single node because refinement combines multiple interfaces. Input-output (IO) sample and coarsening are unary operations so the resulting nodes only have one incoming edge. IO sample edges represent lines 3-5 in Algorithm 2. Interface F refines all others, and the final result is an abstraction $\hat{F}$.

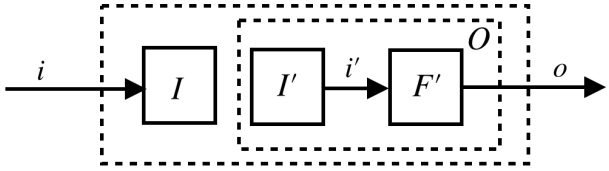


Figure 4.5: An interface representing an input-output sample of F. Here, F' is identical to F except with its input renamed i'. Outputs o do not depend on inputs i, which distinguishes this interface from a filtered interface.

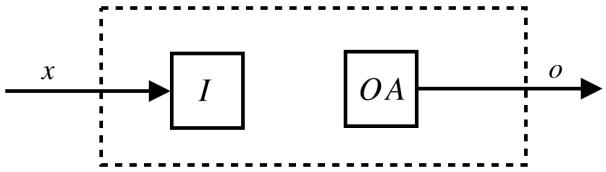


Figure 4.6: Instead of computing an exact $O = \texttt{comp}(I', F')$, one can replace it with any OA such that $OA \preceq O$. Recall that $OA \preceq O$ for sink interfaces is equivalent to $O \subseteq OA$ from a set perspective.

Proof. Condition (4.1) is satisfied by assumption. Condition (4.2) is satisfied because any assignment to i, o that satisfies F also satisfies IO. □

Critically, the interfaces generated via different input sample regions are shared refinable with one another.

Proposition 3. *Shared refinability is preserved under input-output sampling. That is, IO_1 and IO_2 as constructed in Proposition 2 with interfaces I_1, I_2 are shared refinable.*

Proof. Note that $I_j = \texttt{nb}(IO_j)$. Substituting IO_1 and IO_2 into the shared refinement condition yields the following collection of equivalent predicates

$$(I_1 \wedge I_2) \Rightarrow \exists o(IO_1 \wedge IO_2)) \tag{4.10}$$
$$(I_1 \wedge I_2) \Rightarrow \exists o(I_1 \wedge \exists i'(I_1' \wedge F') \wedge I_2 \wedge \exists i'(I_2' \wedge F')) \tag{4.11}$$
$$(I_1 \wedge I_2) \Rightarrow (I_1 \wedge I_2) \wedge \exists o\,(\exists i'(I_1' \wedge F') \wedge \exists i'(I_2' \wedge F')) \tag{4.12}$$
$$(I_1 \wedge I_2) \Rightarrow \exists o\,(\exists i'(I_1' \wedge F') \wedge \exists i'(I_2' \wedge F')) \tag{4.13}$$

Now suppose that an assignment to i satisfies $I_1 \wedge I_2$. From Equation (4.9), there must exist as assignment to o such that $F(i, o)$ is satisfied. Pick any such o and let $i' == i$. These assignments satisfy $I_j' \wedge F'$ for $j \in 1, 2$, signifying that the shared refinement conditions above are true. □

Output Overapproximation for Dynamic System Interfaces

The computation of the output interface $O = \texttt{comp}(I', F')$ is analogous to computing a function's image and is intractable for general systems. One may instead safely use an interface $OA(\varnothing, x^+)$ that overapproximates the forward reachable set interface $O(\varnothing, x^+)$. The interfaces satisfy the refinement relation $OA \preceq O$ signifying that OA is an abstraction of O; recall that this relation is identical to $O \subseteq OA$ when the source interfaces are interpreted as sets or $O \Rightarrow OA$ when using the predicate interpretation.

Proposition 4. *An input-output interface $IO = \texttt{comp}(I, OA)$ constructed via an overapproximation (i.e. $OA \preceq O$) satisfies the refinement relation $\texttt{comp}(I, OA) \preceq F$.*

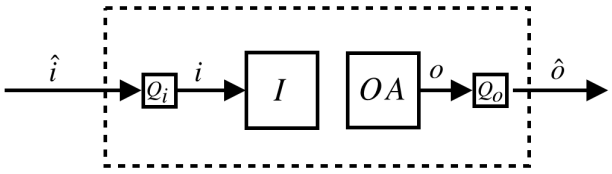


Figure 4.7: Appending quantizers coarsens the input-output sample interface.

The above proposition is easy to prove by observing that $\texttt{comp}(I, OA) \preceq \texttt{comp}(I, O)$, Proposition 2, and applying the transitivity property of the refinement relation partial order.

Proposition 5. *Given an interface F, its renamed version $F' = \texttt{irename}(F, i, i')$, and two interfaces I_1, I_2, let $OA_j(\varnothing, o)$ be a source interface that satisfies*

$$\exists i'(I_j' \wedge F') \Rightarrow OA_i. \tag{4.14}$$

That is, OA_j is a superset of the set encoded by $\exists i'(I_j' \wedge F')$. The interfaces $IO_j = \texttt{comp}(I, OA_j)$ for $j = 1, 2$ are shared refinable.

Proof. Consider the predicate

$$(I_1 \wedge I_2) \Rightarrow \exists o \left(\exists i'(I_1' \wedge F') \wedge \exists i'(I_2' \wedge F')\right)$$

appearing in Proposition 3's proof. Substituting in OA_1 and OA_2 above yields

$$(I_1 \wedge I_2) \Rightarrow \exists o \left(OA_1 \wedge OA_2\right).$$

Validity of this formula is trivial after noting that Equation (4.14) is true. □

Coarsening

To make the sample interface *finite*, the interface inputs and outputs are coarsened with quantizers. An individual sampled abstraction is not useful for synthesis because it is restricted to a local portion of the interface input space. However after sampling, many finite interfaces are merged through shared refinement.

Making Control Synthesis Concrete

The next chapter showcases a collection of concrete modifications to the control synthesis pipeline for the abstraction-based control synthesis case. These modifications reconstruct the traditional abstraction-based control pipeline, as well as encoding domain specific heuristics that empirically lead to favorable computational outcomes. The relational interfaces framework is simple, but powerful. It uncovers some implicit assumptions in existing tools and either remove them or make them explicit. Minimizing the number of assumptions ensures applicability to a diverse collection of systems and specifications and compatibility with future algorithmic modifications.

Chapter 5

Implementation and Benchmarks for Finite Abstraction-based Control Synthesis

We apply the insights of the previous two chapters and apply it to the abstraction-based control pipeline. They will provide a methodology to guide the control synthesizer development. The most common use cases are continuous dynamical systems that are abstracted over a rectangular grid; this chapter contains a collection of domain specific methods that apply to this use case. Each of these methods resemble similar techniques in `PESSOA` [60], `SCOTS` [78], `MASCOT` [44], `ROCS` [55] and `ARCS` [16], but by adopting the relational interfaces framework we uncover some implicit assumptions and generalize or remove them.

Following a discussion on the data structures used to encode predicates, we will survey the domain specific techniques which cover the quantizer implementation, different abstraction and sampling methods, and using forward reach set overapproximations for abstraction. Each of these techniques are independent and can be seamlessly combined when desired. We present a variety of examples such as a vehicular traffic network and lunar lander, as well as the previously introduced Dubins vehicle. They each highlight reductions in memory usage, runtimes, samples, or some combination thereof. These techniques are included in a toolbox `Redax` available at [1]. It is written in `python 3.6` and uses the `dd` package as an interface to `CUDD`[84], a library in `C/C++` for constructing and manipulating binary decision diagrams (BDD). All experiments (except for the sparsity-aware abstraction example) are run on a single core of a 2013 Macbook Pro with 2.4GHz Intel Core i7 and 8GB of RAM.

5.1 Predicate Data Structures

Relational interfaces are stored with data structures that encode predicates. We assume that the standard predicate manipulation operations (conjunction, negation, variable elimination) exist and are available. The relational interface operators are implemented with these low

level predicate operations, but come with the refinement preservation guarantees Theorem 2-4. Low level predicate operations, while powerful, make it easy to inadvertently violate the refinement property. Conforming to the relational interface operations minimizes this danger.

There are two broad classes of data structures used to represent discrete sets and transition relations. *Explicit* data structures such as hash tables, graphs, bitmaps, or multi-dimensional arrays represent elements as distinct locations in memory. The memory requirements for explicit data structures generally scale linearly with the number of elements in the set, but the number of elements in the set grows exponentially with the dimension. Implementing the predicate operations on explicit data structures is generally straightforward.

Symbolic data structures represent the sets and transition relations implicitly as constraints. These data structures allow one to avoid the linear scaling associated with explicit data structures. Shallow and simpler symbolic data structures commonly appear as (in)equality constraints in optimization tools or satisfiability checkers. Boolean functions can be used to represent sets or relations. Decision trees and binary decision diagrams (BDD) [15] are symbolic data structures that compress the Boolean function by detecting patterns. Software implementations of BDD are equipped with the standard predicate operations [84]. We opt to use BDDs because they empirically exhibit the best performance on the benchmark examples in this chapter.

5.2 Multi-Precision Quantizers

A key benefit of quantizing (or "abstracting") continuous dynamics into finite representations is that the predicate operations are finitely computable whereas for continuous representations they are not in general. While the loss of precision during the quantization step means that the result of control synthesis is conservative, the granularity of the quantizers can be modified. On the other hand, finer granularity solutions can cause the memory and computational requirements to store and manipulate the symbolic representation to exceed machine resources. To counter this, one may inject coarser quantizers directly into the synthesis pipeline to simplify computations and reduce memory usage whenever it becomes apparent that the fixed point computation will require more computational resources than are available.

We show a particular quantizer that is geared towards dynamic or variable precision abstractions such as those that appear in tools `ROCS`, `ARCS`, and `MASCOT` [16, 44]. Afterwards, we show how the quantizer can be used to coarsen interfaces via series composition. For clarity, our quantizer will build off a one-dimensional quantizer that translates a real number within a compact interval into a finite vector of bits. Remark 1 will briefly explain how to construct quantizers for higher dimension spaces and unbounded domains.

Certain data structures like trees or binary decision diagrams (BDDs) [15] are natural candidates for capturing and encoding a space's hierarchical decomposition. A sequence of bits can be used to traverse that data structure to arrive at a subset of that space. More bits

$N = 0$	$---$			
$N = 1$	$\bot--$		$\top--$	
$N = 2$	$\bot\bot-$	$\bot\top-$	$\top\bot-$	$\top\top-$
$N > 2$	$\vdots$			

Figure 5.1: Varying length N of a bit vector leads to different granularity covers of an interval. The don't care term "$-$" signifies that the value of a certain bit does not matter. Bit vectors of finite length are implicitly appended with an infinite sequence of don't cares.

allow one to specify finer granularity sets. As an illustrative example, consider an interval $[0, 1]$ that is to be covered. Let N represent a number of bits to construct a cover of $[0, 1]$, and assign a unique bit sequence to identify each set in the cover. Figure 5.1 depicts for varying N how an interval can be covered by a collection of small intervals, while minimizing overlaps. A greater N signifies that the cover can be constructed with finer intervals. A bit vector $o_1 o_2 \ldots o_N$ of length N can be used to specify an interval of width 2^{-N} such as

$$\left[\sum_{k=1}^{N} 2^{-k} o_k, \sum_{k=1}^{N} 2^{-k} o_k + 2^{-N}\right]. \tag{5.1}$$

If $N = \infty$, then any value in $[0, 1]$ can be encoded via the infinite weighted sum $\sum_{k=1}^{\infty} 2^{-k} o_k$. For $N = 0$, the interval is $[0, 1]$ itself. Figure 5.1 implicitly appends don't care terms $-$ to the end of bit vectors. This allows finite length bit vectors (intervals) to be identical to the disjunction (union) of longer bit vectors (smaller intervals) with a common prefix.

A quantizer acts by truncating a bit vector to a specified finite number of bits and outputting the result. The effect of truncating bits is to implicitly widen or coarsen that interval. A quantizer $Q_N(o, \hat{o})$ that only retains the first N bits satisfies the following formula and is depicted in Figure 5.2

$$Q_N(o, \hat{o}) = Q_N(o_1 \cup \ldots, \hat{o}_1 \cup \ldots) \tag{5.2}$$

$$= \bigwedge_{k=1}^{N} (o_k == \hat{o}_k) \tag{5.3}$$

Even though the quantizer Q_N only depends on the N most significant bits, it implicitly receives and emits infinite bit sequences. It ignores higher precision input bits and non-deterministically assigns values to higher precision output bits. Equality with the Equation (5.3) is shown via the following simple equalities:

$$Q_N(o, \hat{o}) = \bigwedge_{k=1}^{N} (o_k == \hat{o}_k) \wedge \top \wedge \top \tag{5.4}$$

$$= \bigwedge_{k=1}^{N} (o_k == \hat{o}_k) \wedge \bigwedge_{k=N+1}^{\infty} (\neg o_k \vee o_k) \wedge \bigwedge_{k=N+1}^{\infty} (\neg \hat{o}_k \vee \hat{o}_k). \tag{5.5}$$

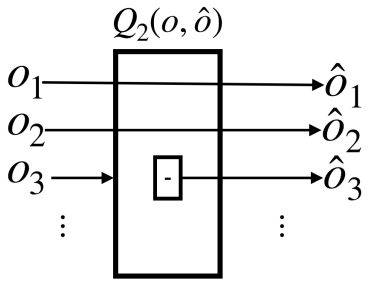


Figure 5.2: A bit quantizer in Equation (5.3) with $N = 2$. Higher significant bits are outputted but are replaced with a don't care term "$-$" that can non-deterministically be true or false.

The input and output domains have identical cardinality but the quantizer is not a function due to the non-deterministic output. It is easy to see that for two quantizers Q_N, Q_M with precisions $N, M \in \mathbb{N} \cup \{\infty\}$ and $N \leq M$ the relation $Q_N \preceq Q_M$ holds. The quantizer is multi-precision in the sense that multiple quantizers of varying precisions can be composed with one another. A collection of quantizers composed in series also yields a single quantizer with the minimum precision.

Table 5.1 summarizes how certain interfaces in Figure 4.1 were constructed by applying the coarsening operation with different precision quantizers. Applying the output coarsening operation increases output non-determinism resulting in thicker black bands vertically. The input coarsening operation thickens the black bands horizontally but also causes the blocking region shown as white gaps to grow.

Applying the bit quantizer definition above to the coarsening operations yields insight into how it alters the original interface and a more efficient encoding.

Example 2 (Output Quantizer)**.** *Consider an interface $F(i, o)$ where $o = o_1 \cup o_2 \cup \ldots$ is a composite variable representing an infinite bit vector. Let $o_{msb} = o_1 \cup \ldots \cup o_N$ be the most significant bits and $o_{lsb} = o_{N+1} \cup \ldots$ be the least significant bits.*

$$\texttt{ocoarsen}(M, Q(o, \hat{o})) \tag{5.6}$$
$$= \texttt{ohide}(o, \texttt{comp}(M, Q(o, \hat{o}))) \tag{5.7}$$
$$= \exists o \left(\bigwedge_{k=1}^{N} (o_k == \hat{o_k}) \wedge F \wedge \forall o \left(F \Rightarrow \exists \hat{o} \left(\bigwedge_{k=1}^{N} (o_k == \hat{o_k}) \right) \right) \right) \tag{5.8}$$
$$= \exists o \left(\bigwedge_{k=1}^{N} (o_k == \hat{o}_k) \wedge F \wedge \top \right) \tag{5.9}$$
$$= \exists o_{msb} \left(\bigwedge_{k=1}^{N} (o_k == \hat{o}_k) \wedge \exists o_{lsb} F \right) \tag{5.10}$$

The last predicate has signature $(i, \hat{o}_{msb})$ and is equivalent to the predicate $\exists o_{lsb} F$ after the o

bit are replaced with their respective $\hat{o}$ bits. In other words, the output non-determinism is increased.

The effect of input coarsening is more complicated than that of output coarsening.

Example 3 (Input Quantizer)**.** *Consider an interface $F(i, o)$ where $i \equiv i_1 \cup i_2 \cup \ldots$ is a composite variable representing an infinite bit vector. Let $i_{msb} = i_1 \cup \ldots i_N$ be the most significant bits and $i_{lsb} = i_{N+1} \cup \ldots$ be the least significant bits. The input coarsening operator yields the following equivalent prediates.*

$$\texttt{icoarsen}(F, Q(\hat{i}, i)) \tag{5.11}$$

$$= \texttt{ihide}(i, \texttt{comp}(Q(\hat{i}, i), F)) \tag{5.12}$$

$$= \exists i \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \wedge F \wedge \forall i \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \Rightarrow \texttt{nb}(F) \right) \right) \tag{5.13}$$

$$= \exists i \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \wedge F \wedge \forall i_{msb} \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \Rightarrow \forall i_{lsb} \texttt{nb}(F) \right) \right) \tag{5.14}$$

$$= \exists i \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \wedge F \right) \wedge \forall i_{msb} \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \Rightarrow \forall i_{lsb} \texttt{nb}(F) \right) \tag{5.15}$$

$$= \exists i_{msb} \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \wedge \exists i_{lsb} F \right) \wedge \forall i_{msb} \left(\bigwedge_{k=1}^{N} \left(i_k == \hat{i}_k \right) \Rightarrow \forall i_{lsb} \texttt{nb}(F) \right) \tag{5.16}$$

The last predicate has signature $(\hat{i}_{msb}, o)$ and is equivalent to the predicate $\exists i_{lsb} F \wedge \forall i_{lsb}(\texttt{nb}(F))$ after all i bits are replaced with their respective $\hat{i}$ bits. The $\exists i_{lsb} F$ term increases output non-determinism by taking a union of outputs generated by different lower significant bit assignments. Term $\forall i_{lsb}(\texttt{nb}(F))$ imposes that if some input assignments can block, then all other input assignments with identical most significant bits are also blocked.

These examples show that it is more efficient to directly eliminate the lower significant bit variables through variable quantification rather than compute Equation (5.8) and Equation (5.13).

Remark 1. *A multi-dimensional quantizer simply consists of multiple scalar quantizers. The encoding for the interval $[0, 1]$ can be rescaled to an interval $[a, b]$. Additional bits can be added to account for quantizers that do not contain a power of two number of bins, or for quantizers that extend beyond a finite interval to the entire real line $\mathbb{R}$.*

Original Interface	Original Grid	New Interface	New Grid	Input Quant.	Output Quant.
A	128×128	C	128×16	None	Q_4
B	64×128	D	32×16	Q_5	Q_4
C	128×16	$\top$	128×1	None	Q_0
D	32×16	$\bot$	1×16	Q_0	None

Table 5.1: Interfaces from Figure 4.1 were obtained via applying input and output coarsen operations, e.g., interface D is obtained by applying quantizers Q_5 and Q_4 to B's inputs and outputs respectively. The new grid column indicates the finest grid of the new interface obtained by applying the specified quantizers. The actual grid may be coarser, e.g., interfaces $\top$ and $\bot$ actually live on a 1×1 grid.

5.3 Abstraction through Forward Reachable Set Overapproximations

Computing the exact input-output sample interface for general nonlinear systems is not always feasible. Many computationally efficient finite abstraction algorithms exist to instead *over-approximate* the behavior of the original continuous space control system. They leverage properties of the system dynamics such as Lipschitz constants, growth bounds [76], matrix measures [58], and mixed monotonicity [22] to generate appropriate bounds.

Consider the deterministic system dynamics $F : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. Let $OA : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$ be any set-valued map that returns a superset of the true reachable set function of the control system. We present two possible realizations of OA that are tailored for different classes of dynamical systems. Both of these classes of dynamics have Euclidean state space $\mathcal{X} \equiv \mathbb{R}^n$ and control space $\mathcal{U} \equiv \mathbb{R}^m$. Let the two boxes $[a^x, b^x] \equiv \{x \in \mathbb{R}^n : a^x \leq x \leq b^x\}$ and $[a^u, b^u] \subseteq \{x \in \mathbb{R}^m : a^u \leq x \leq b^u\}$ represents that are inputted to OA.

- An over-approximation method for non-linear systems with perturbed measurements is presented in [76] and is called a "growth bound". Let the centers of the state and input box be $c^x = \frac{1}{2}(b^x + a^x)$ and $c^u = \frac{1}{2}(b^u + a^u)$, respectively. The radius of the state box is given by $r^x = \frac{1}{2}(b^x - a^x)$. A local error bound $\beta : \mathbb{R}^n \to {\mathbb{R}_{\geq 0}}^n$ propagates the uncertainty in each state coordinate independently based on the partition of the state-control input space. This error bound resembles a Lipschitz bound for continuous functions. The over-approximation function OA associated with this bound is defined by
$$OA([a^x, b^x], [a^u, b^u]) \equiv (F(c^x, c^u) + [-\beta(r^x), \beta(r^x)]) \tag{5.17}$$
where the $+$ represents the Minkowski set addition.

- An efficient estimate of the reachable set of nonlinear, monotone dynamical systems is presented in [63]. The authors consider discrete-time systems that are monotone with respect to the nonnegative convex cone ${\mathbb{R}_{\geq 0}}^n$ and with respect to the standard

element-wise partial order $\leq$. The over-approximation function OA for this class is given by

$$OA([a^x, b^x], [a^u, b^u]) \equiv \left\{x^+ : F(a^x, a^u) \leq x^+ \leq F(b^x, b^u)\right\}. \tag{5.18}$$

Monotone systems are a subclass of mixed monotone systems [22], for which there exists a generalized procedure that accounts for increasing or decreasing components of F.

5.4 Interface Sampling Methods

To our knowledge, all the control synthesis tools `PESSOA` [60], `SCOTS` [78], `MASCOT` [44], `ROCS` [55], and `ARCS` [16] construct finite abstractions with an enumerative algorithm. That is, they apply a number of operations that are proportional with the number of discrete states in the abstraction. The pseudo-code presented in Algorithm 1 is one common method to construct a system abstraction of $F(x \cup u, x^+)$. It uses forward reachable set overapproximations `OutputOverapprox`$(\cdot)$. It exhaustively traverses a collection of discrete states which each correspond to a set within a partition. Note that := signifies an assignment operation and is distinct from the operations for equality assertion = and equivalence checking ==.

Algorithm 1 Enumerative Abstraction

1: Fix spatial quantizers Q_i, Q_o
2: $\hat{F} := \bot$ ▷ Universal Abstraction
3: **for all** partitions $I \subseteq \mathcal{D}(i)$ induced by Q_i **do**
4: $\quad OA :=$ `OutputOverapprox`(F, I)
5: $\quad OA :=$ `ocoarsen`(OA, Q_o)
6: $\quad IO :=$ `comp`(I, OA)
7: $\quad \hat{F} := \hat{F} \vee IO$ ▷ Adds transitions. Assumes I disjointness.

Algorithm 1 has some desirable properties. It is an "anytime algorithm", so even if the loop terminates early, $\hat{F}$ is a valid abstraction, albeit one that may be of little use because there will be parts of the input space that are not sampled. The input grid traversal on line 3 is actually a nested for loop with depth equal to the dimension of the input space. This is the main source for the exponential runtime.

The abstraction $\hat{F}$ is constructed by iteratively adding transitions via the disjunction in line 7. This approach imposes design constraints on other parts of Algorithm 1. Specifically, disjointness of input sets is a key property enforced by line 3. Without disjointness, line 7 adds many more transitions than necessary and makes the interface exhibit unnecessary non-determinism, causing one to make coarser abstractions rather than finer ones! ROCS [55] and ARCS [16] do not fix a spatial quantizer like in Algorithm 1 and instead abstract systems by implementing splitting/bisection operations to adaptively construct partitions. However, both are still instances of an enumerative abstraction procedures.

Algorithm 2 (visualized in Figure 4.4) is a generalization of Algorithm 1 that is both more flexible and can exhibit dramatically reduced runtimes with comparable results. It is the result of modifications to specific parts of Algorithm 1. First, line 7's disjunction is changed to a shared refinement operation. Shared refinement reduces down to the disjunction under the input disjointness condition. Second, disjoint input sets are no longer generated by line 3's loop. Instead, at each iteration any arbitrary subset of the input space can be chosen. Third, the fixed quantization from line 1 is changed so that a desired quantization level can be chosen at each iteration. On iterations with a smaller input set a finer granularity can be chosen if desired, while for larger sets a coarser granularity can be chosen.

Algorithm 2 Abstraction Through Shared Refinement

1: $\hat{F} := \bot$ ▷ Universal Abstraction
2: **repeat**
3: Pick an input set $I \subseteq \mathcal{D}(i)$. ▷ Sink Interface
4: $OA := \texttt{OutputOverapprox}(F, I)$ ▷ Source Interface
5: $IO := \texttt{comp}(I, OA)$ ▷ Equal to $I \wedge O$
6: Pick desired quantizatizers Q_i, Q_o
7: $IO := \texttt{icoarsen}(IO, Q_i)$
8: $IO := \texttt{ocoarsen}(IO, Q_o)$
9: $\hat{F} := \texttt{refine}(\hat{F}, IO)$
10: **until** User specified condition

Example 4 illustrates the key idea behind how Algorithm 2 can leverage overlapping input sets to implicitly create more discrete states than the number of loop iterations.

Example 4. *Let $I_1 \wedge O_1$ and $I_2 \wedge O_2$ be two interfaces generated in the same manner as IO in Proposition 2. The shared refinement interface outputted by Algorithm 2*

$$(I_1 \vee I_2) \wedge (I_1 \Rightarrow O_1) \wedge (I_2 \Rightarrow O_2) \tag{5.19}$$

is logically equivalent to

$$(I_1 \wedge I_2 \wedge O_1 \wedge O_2) \vee (I_1 \wedge \neg I_2 \wedge O_1) \vee (\neg I_1 \wedge I_2 \wedge O_2). \tag{5.20}$$

If I_1 and I_2 correspond to disjoint sets, then this simplifies to the output of Algorithm 1

$$(I_1 \wedge O_1) \vee (I_2 \wedge O_2) \tag{5.21}$$

because $I_1 \wedge I_2 \Leftrightarrow \bot$, $I_1 \Rightarrow \neg I_2$ and $I_2 \Rightarrow \neg I_1$. If I_1 and I_2 are not disjoint, then (5.20) can be viewed as three reach set overapproximations $O_1 \wedge O_2$, O_1 and O_2 for three respective disjoint input sets $I_1 \wedge I_2$, $\neg I_1 \wedge I_2$, and $I_1 \wedge \neg I_2$. By leveraging overlapping input domains, Algorithm 2 has generated three discrete states despite only being provided two interfaces $I_1 \wedge O_1$ and $I_2 \wedge O_2$.

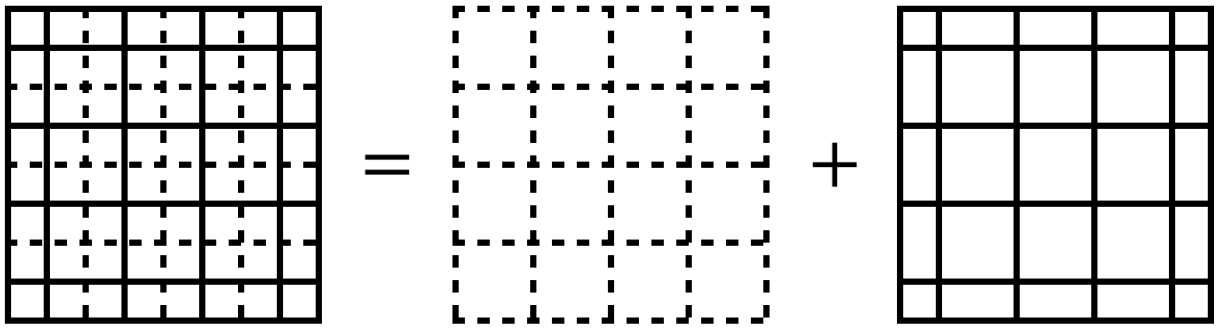


Figure 5.3: A 8×8 partition induced by overlaying a 4×4 partition and a 5×5 partition. Note that $4^2 + 5^2 < 8^2$. For grids of dimension N, this technique reduces the number of iterations by a factor of roughly 2^{N-1}. An initial factor of roughly 2^N is achieved by doubling the cell width along each dimension, but iterating over two grids reduces this factor to 2^{N-1}.

Suppose one needs to construct a finite abstraction over an 8×8 grid. Algorithm 1 would exhaustively iterate over the full 64 element grid. Each iteration constructs an input interface that is disjoint from the other ones. Switching to Algorithm 2 lets one use the overlapping property of shared refinement to reduce the number of iterations while still inducing a grid of the desired granularity. Figure 5.3 depicts a simple two dimensional version of the iteration procedure where a higher granularity grid is constructed by two coarser grids that are offset from one another. This technique generally yields a more conservative abstraction than the one obtained by a full granularity traversal, but leads to a reduction in abstraction runtime. The runtime is reduced by a factor that increases with the state dimension.

5.5 Sparsity-Aware Abstraction

Many dynamical systems consist of a collection of sub-components that exhibit low dimensional structure. Sparsity-aware abstraction avoids abstracting a monolithic system directly and instead abstracts its sub-components, then combines them together. Suppose that for all $i \in \{1, \ldots, N\}$, $\hat{F}_i$ is an abstract interface of F_i constructed using the techniques introduced in this section. Let $F = \texttt{comp}(F_1, \ldots, F_N)$ be a concrete monolithic interface that results from composing the sub-interfaces. Rather than compute $\hat{F}$ using sampling, one can instead compute $\hat{F} = \texttt{comp}(\hat{F}_1, \ldots, \hat{F}_N)$.

Section 5.7 showcases as an example the sparsity-aware abstraction of a large traffic network of up to 51 dimensions. It can be viewed as a counterpart to the decomposed control predecessor in Section 3.4, which avoided using the monolithic representation of the system dynamics F and instead had an incremental computation that reflected the underlying dependency structure.

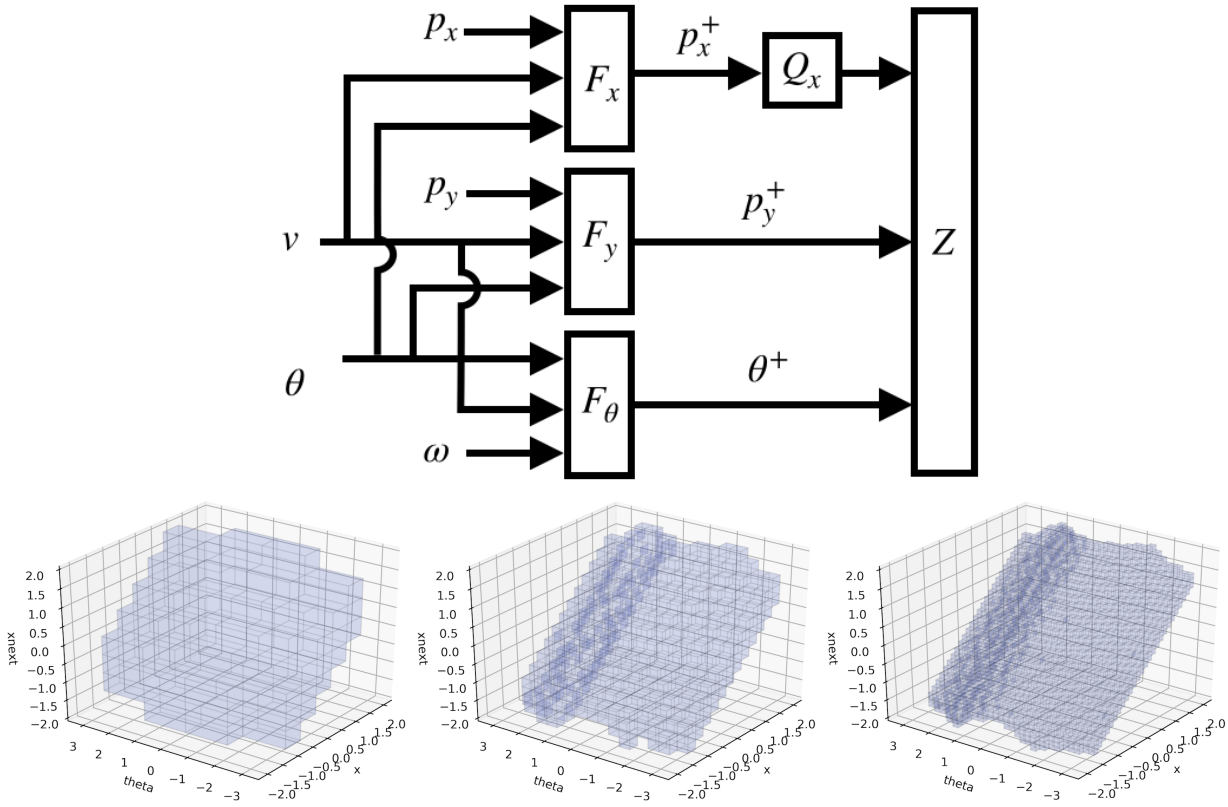


Figure 5.4: Coarsening along the p_x^+ variable during synthesis with Figure 3.10's example. The inserted quantizer can be interpreted as coarsening either F_x's output, Z's input, or both.

5.6 Computationally Constrained Synthesis

Tools for symbolic verification and synthesis extensively use the binary decision diagram data structure to represent predicates. BDDs compress predicates by identifying and eliminating redundant sub-structures. In practice, the sequence of interfaces Z_i generated by the fixed point iteration grows in complexity. This occurs even if the dynamics F and the target set T(or safe set S) have compact BDD representations. BDDs are implemented as graphs and their complexity is measured in the number of nodes in that graph.

The coarsening operators are one method to combat this growth in complexity. To maintain tractability, interfaces for both the winning region and dynamics are coarsened whenever the winning region becomes too complex. In the context of the control synthesis graph, this can be viewed as inserting a quantizer into the pipeline as depicted in Figure 5.4. The quantizer effectively reduces the amount of information sent across iterations of the fixed point iteration by reducing the size of Z's BDD encoding. Heuristics to determine when and how to quantize include:

- *Downsampling with progress*: The authors of [44] leverage the idea of dynamically coarsen-

ing the dynamics and sets Z_i while synthesizing controllers. Multi-layered synthesis starts with the coarsest dynamics and monitors progress (or lack thereof) in the fixed point calculation to decide when to switch amongst the levels. Whenever the game iteration is making progress and hasn't reached a fixed point, a coarser representation of the dynamics is used. The hypothesis is that a fixed point can be reached faster because the state space of a coarser representation has smaller cardinality.

The scheme in [44] precomputes abstractions of multiple granularities. Multi-layered coarsening also occurs along each state dimension uniformly and simultaneously.

- *Greedy quantization*: This heuristic entails selectively coarsening along certain variables or dimensions by checking at runtime which variable, when coarsened, would cause Z_i to shrink the least. This reward function can be leveraged in practice because coarsening is a much more computationally cheap operation than the composition operation. The winning region can be coarsened until the BDD encoding complexity (i.e., number of nodes) reduces below a desired threshold. The traditional reach game is theoretically guaranteed to reach a fixed point, but in practice may not reach one due to limited computational resources. In contrast, the coarsening scheme above is less computationally intensive, but is not guaranteed to reach a fixed point as long as quantizers can be dynamically inserted into the synthesis pipeline. It is guaranteed once the quantizers are always inserted at a fixed precision.

  Figure 5.10 shows this heuristic being applied to reduce memory usage at the expense of answer fidelity. A fixed point is not guaranteed as long as quantizers can be dynamically inserted into the synthesis pipeline, but is once quantizers are always inserted at a fixed precision. Figure 5.11 depicts the reach basin computed using greedy quantization on the Dubins vehicle example.

5.7 Examples

Double Integrator with Random State-Control Boxes

We illustrate the flexibility of Algorithm 2 with a simple double integrator with a drag term.

$$p^+ = p + Tv \tag{5.22}$$
$$v^+ = v + Ta - kT\text{sgn}(v)v^2 - Tg \tag{5.23}$$

where $T = 0.2$, $k = .1$, $g = 9.8$ and $\text{sgn} : \mathbb{R} \to \{-1, 1\}$ maps a number to its sign (with the convention $\text{sgn}(0) = 1$).

Figure 5.5 illustrates two sampling schemes when constructing the finite abstraction for Equation (5.23). The first is an enumerative sampling over 16384 discrete grid cells, which corresponds to Algorithm 1 as implemented in SCOTS or PESSOA. We also randomly sample boxes of the state and control space by varying their width along each dimension and shifting

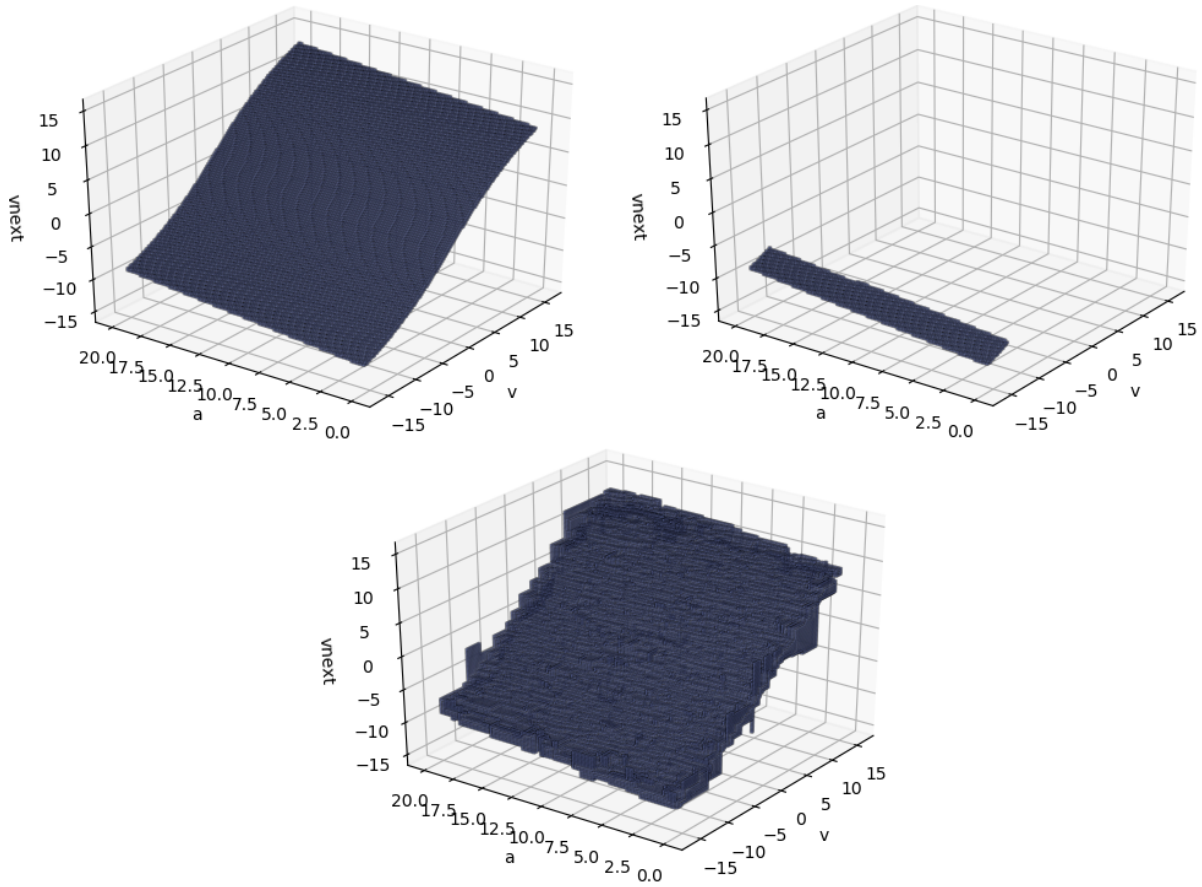


Figure 5.5: Constructing the double integrator abstraction's v^+ component from Equation (5.23). Algorithm 1 run over a full 16384 element grid (top left) and its intermediate result after 2000 iterations (top right). Randomly sampling boxes and applying Algorithm 2 yields a coarse but useful approximation after 2000 iterations (bottom).

them. Note that the shifts and widths do not align with a predetermined and fixed grid and many of the sampled boxes intersect with other samples. The random sampling technique illustrates flexibility of Algorithm 2 and is not meant to be optimal with respect to sample efficiency, but still shows how one does not need to sacrifice performance along these metrics compared to Algorithm 1.

The continuous state space is the hyperrectangle $[-10, 10] \times [-16, 16]$ and the continuous control input space is $[0, 20]$. Each state and control is allocated 7 bits to discretize the space into 128 subintervals. Given an interval, an adversarial non-determinism gets to choose which concrete value is used. For control inputs, this signifies imprecise actuation.

Figure 5.6 depicts the growth of the invariant set with additional samples. The invariant set generated with the 2000 sample interface closely resembles the one generated with the exhaustive grid traversal, even though the exhaustive traversal at the fixed granularity requires 16384 samples.

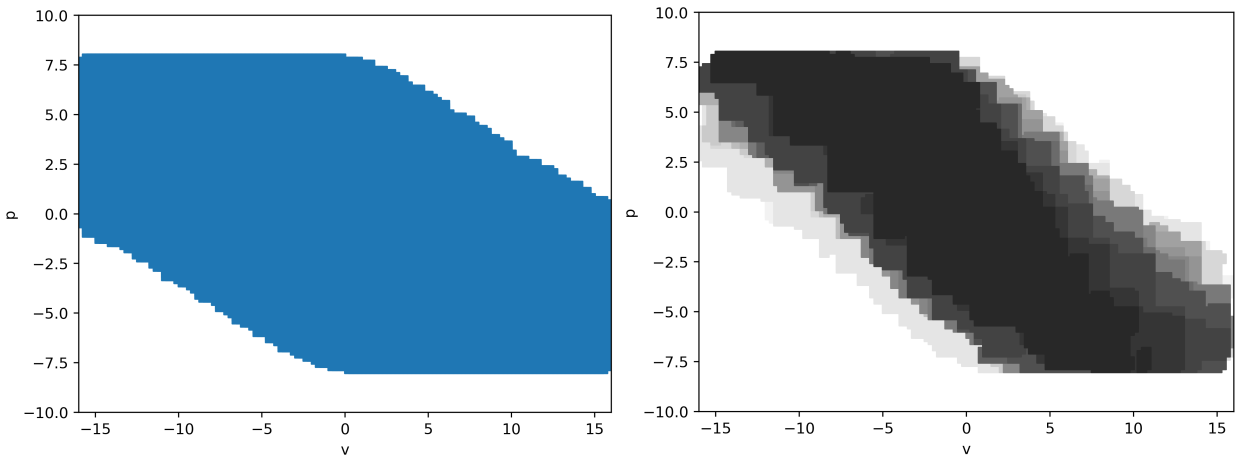


Figure 5.6: Double integrator controlled invariant set computed using the exhaustive grid traversal of two 16384 element grids (left). The invariant set generated from the random sample abstraction (right) grows with more samples. The different shades represent the invariant sets constructed after 400, 800, 1200, 1600, 2000 samples, where fewer samples represent darker contours.

Dubins Vehicle

Recall the Dubins vehicle example introduced in Section 2.2. An overapproximation of a forward reachable set was represented as a hyper-rectangle. This hyper-rectangle was computed by simple interval analysis techniques to compute output bounds for the $\sin(\cdot)$ and $\cos(\cdot)$ functions. Each dimension is quantized with 7 bits of precision, which corresponds to $128 = 2^7$ bins. The discrete state space consists of 2^21 or approximately 2 million values. The θ continuous state lies in a circular domain and the discretization is encoded using Gray code.

We evaluate the effectiveness of three methods to speed up the abstraction and synthesis steps:

1. **Random Overlapping Samples:** Figure 5.7 depicts an interface (F_x) being constructed with random boxes generated by uniformly sampling their widths and offsets along each dimension. The offsets and widths do not align with a predetermined and fixed grid and many rectangles also overlap. Figure 5.8 depicts how the reach game's winning region increases in size as the number of samples increases.

2. **Monolithic, Partially Decomposed, and Fully Decomposed Control Predecessors:** Figure 3.10 showed how the fully decomposed control predecessor is computed, but there is a spectrum between a monolithic system, a partially decomposed system, and fully decomposed system, where each decomposition has its own variant of `cpre`$(\cdot)$. Figure 5.9 shows the synthesis runtimes associated with different system decompositions, and depicts improved performance when one avoids preemptively constructing the monolithic interface.

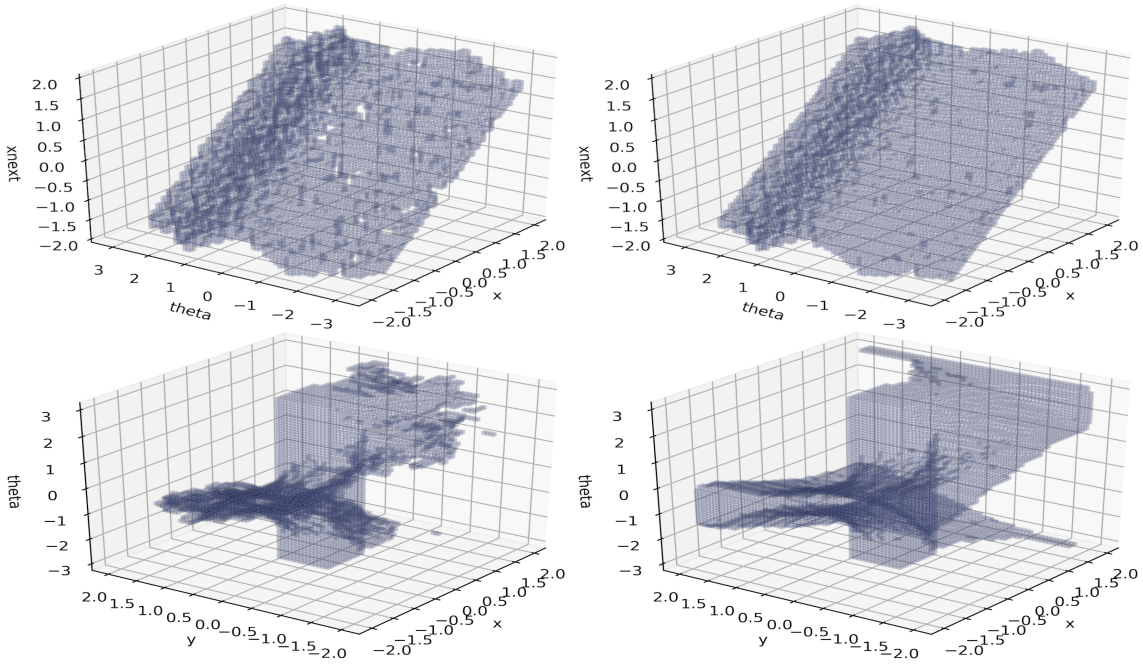


Figure 5.7: Discrete abstractions of interface F_x with 3000 (upper left) and 6000 (upper right) random samples, along with their corresponding solutions to the reach game (bottom). There are "holes" in the 3000 sample interface's domain, yet the winning region extends beyond the target set. With additional samples, the holes are covered and the winning region grows. The rectangular target region has $131k$ discrete states; the left set contains $258k$ states and the right $458k$ states. Many samples are rejected because the interface outputs exit the domain $\mathcal{D}(p_x) \equiv [-2, 2]$ over which the quantizer was well defined.

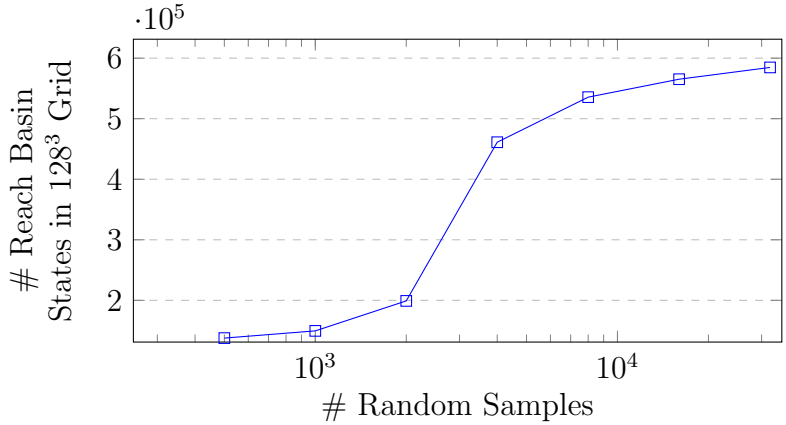
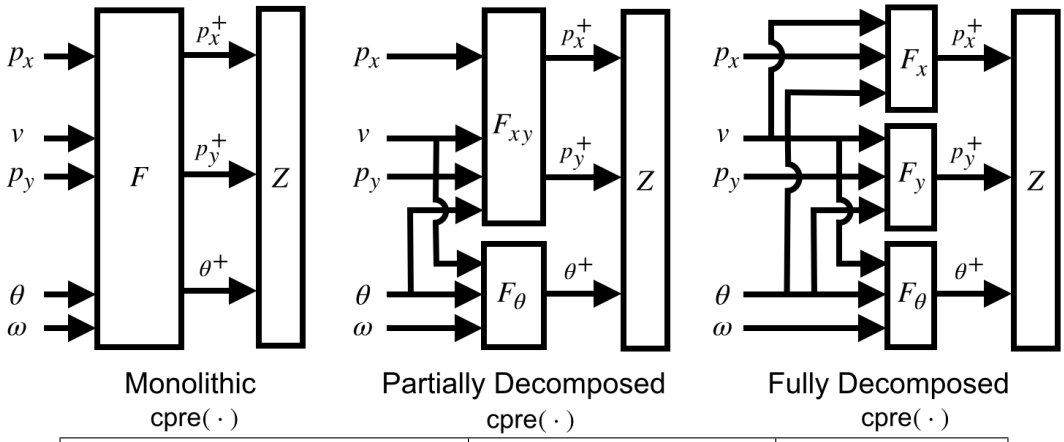


Figure 5.8: The number of states in the Dubins vehicle's computed reach basin grows with the number of random samples. The vertical axis is lower bounded by the number of states in the target $131k$ and upper bounded by $631k$, the number of states using an exhaustive traversal. Naive implementations of the exhaustive traversal would require 1.2×10^7 samples.



Decomposition	**Parallel Compose Runtime (s)**	**Reach Game Runtime (s)**
F (Monolithic)	0.56	103.09
$F_{y\theta}, F_x$ (Partially Decomp.)	0.02	28.31
$F_{x\theta}, F_y$ (Partially Decomp.)	0.01	28.71
F_{xy}, F_θ (Partially Decomp.)	0.06	10.61
F_x, F_y, F_θ (Fully Decomp.)	n/a	4.42

Figure 5.9: A monolithic `cpre`$(\cdot)$ incurs unnecessary pre-processing and synthesis runtime costs for the Dubins vehicle reach game. Each variant of `cpre`$(\cdot)$ above composes the interfaces F_x, F_y and F_θ in different permutations. For example, F_{xy} represents `comp`(F_x, F_y) and F represents `comp`(F_x, F_y, F_θ).

3. **Greedy Quantization:** The solution to the reach game requires 7127 BDD nodes to encode. This is reasonable for a desktop computer but may not be acceptable if one wants to store the resulting controller on an embedded, memory constrained device. Figure 5.10 shows the growth of the intermediate interfaces Z_i generated during the reach game. It grows both in size (number of states) and encoding complexity (number of BDD nodes) when a memory bound of 3000 BDD nodes is imposed. Figure 5.11 depicts the reach game solutions for various bounds on the BDD nodes.

Lunar Lander

We consider the lunar lander from OpenAI gym [12], a `python` simulation environment for reinforcement learning research. The lander is set up within a physics simulation engine and has six continuous state dimensions and two continuous input dimensions. After some minor

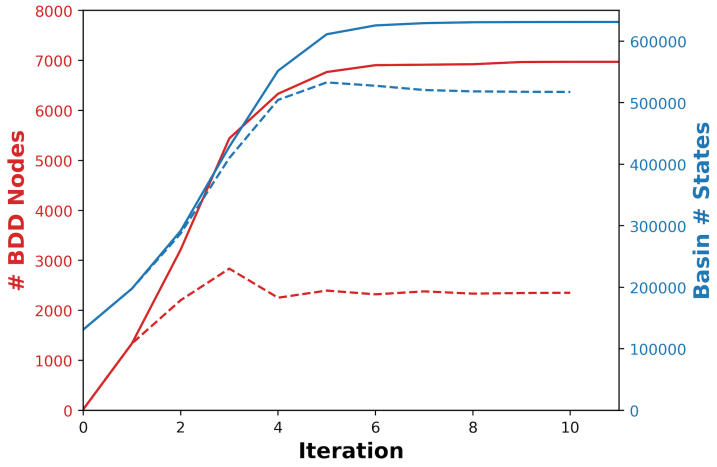


Figure 5.10: Number of BDD nodes (red) and number of states in reach basin (blue) with respect to the Dubins vehicle reach game iteration with a greedy quantization. The solid lines result from the unmodified game with no coarsening heuristic. The dashed lines result from greedy coarsening whenever the winning region exceeds 3000 BDD nodes.

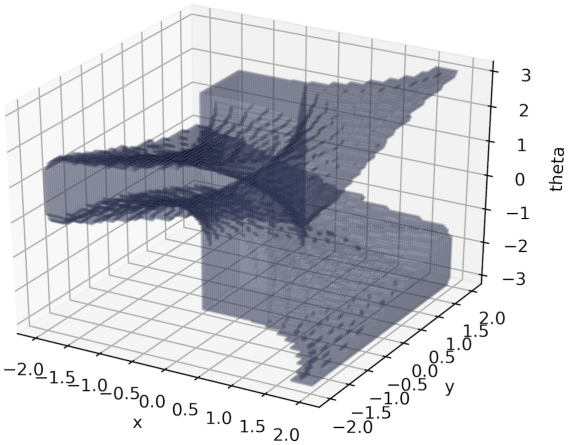
modifications [1], we identified the following discrete time dynamics

$$p_x^+ == p_x + .01031 v_x \tag{5.24}$$
$$p_y^+ == p_y + .0225 v_y \tag{5.25}$$
$$v_x^+ == v_x - .0539 u_1 \sin(\theta) + .0106 u_2 \cos(\theta) \tag{5.26}$$
$$v_y^+ == v_y + .0359 u_1 \cos(\theta) + .00707 u_2 \sin(\theta) - .0267 \tag{5.27}$$
$$\theta^+ == \theta + .05\omega \tag{5.28}$$
$$\omega^+ == \omega - .05598 u_2. \tag{5.29}$$

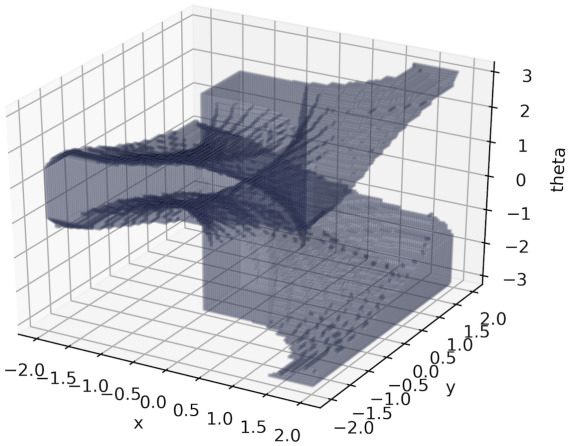
Control input $u_1 \in \{0\} \cup [.5, 1]$ represents a main thruster mounted on the bottom of the lander, and input $u_2 \in [-1, -.5] \cup \{0\} \cup [.5, 1]$ represents a pair of side thrusters. Only one side thruster can be activated at a time and both are aligned in such a way that they apply both a torque and a linear force when activated. Both thrusters can only apply impulses with magnitude greater than 0.5 when activated. This limits the system's ability to exert fine control over the system without resorting to high frequency chattering.

The continuous region of interest for our problem is $p_x \in [-1, 1]$, $p_y \in [0, 1.3]$, $v_x \in [-1, 1]$, $v_y \in [-1, 1]$, $\theta \in [-\frac{\pi}{5}, \frac{\pi}{5}]$ and $\omega \in [-.6, .6]$. The discretized state space consists of ∼137 billion states obtained from a $256 \times 256 \times 64 \times 64 \times 32 \times 16$ grid. Inputs are constrained to discrete sets $u_1 \in \{0, .66, .83, 1\}$ and $u_2 \in \{-.5, 0, .5\}$.

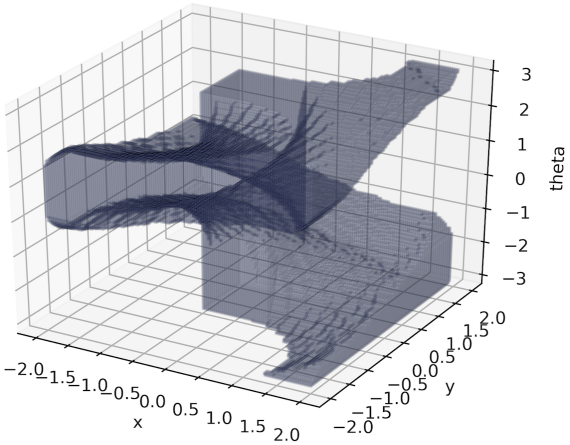
[1] We removed a small dispersion term that induced additive actuation noise. The geometry of the lander was changed to be a square instead of a trapezoid. The center of gravity and the point $(p_x, p_y) = (0, 0)$ were not the same for the original dynamics.



(a) Heuristic Limit: 3000 nodes. States in final basin: 565536. Final basin BDD nodes: 6247.



(b) Heuristic Limit: 5000 nodes. States in final basin: 608660. Final basin BDD nodes: 6433.



(c) Heuristic Limit: 7000 nodes. States in final basin: 631272. Final basin BDD nodes: 6889.

Figure 5.11: Dubins vehicle reachable basins computed with greedy quantization predecessor. Whenever the BDD representing the intermediate basin exceeded $N \in \{3000, 5000, 7000\}$ nodes, it was coarsened before applying `cpre`$(\cdot)$. Coarsening occurred along the dimension that reduced the basin size the least.

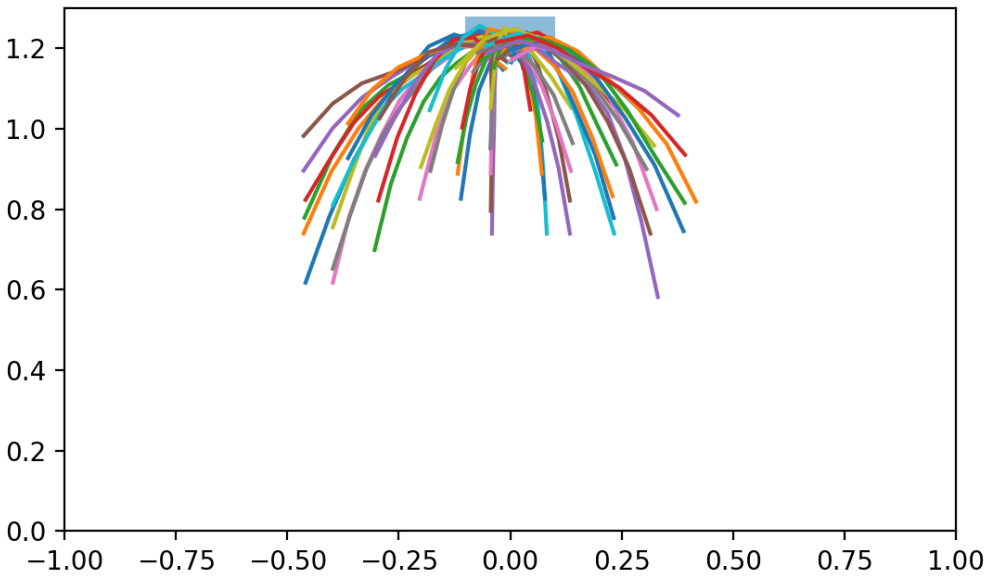


Figure 5.12: Projection of trajectories into the p_x and p_y dimentions with the target set's projection in blue.

While one could construct interfaces associated with each equation (5.24)-(5.29), the positional components p_x^+ and p_y^+ would require finer grids than those above to capture a changing state over the short time horizon. Instead of using the one-step dynamics above, we instead use a sampled system with a period of 9 time steps. We refer to the interfaces associated with the unrolled dynamics with the names shown in Table 5.2. The time to construct all six interfaces was 430.5 seconds. A reach objective with a target region T is specified by

$$p_x \in [-.1, .1] \wedge p_y \in [1.2, 1.3] \wedge \theta \in [-.15, .15] \wedge v_y \in [-8, .1]$$

which corresponds to 73 million states of the full dimensional space. We iterate a custom reach operator 20 iterations to construct a winning set with 344.6 million states. Trajectories reaching the target are depicted in Figure 5.12. The control synthesis runtime was 4194 seconds but a small portion of that includes applying the coarsening operation as detailed later. These abstraction and synthesis runtime numbers were achieved using all the techniques in this chapter (including those for the Dubins vehicle).

It also uses heterogeneous grids, where different interfaces have different can use quantizers of differing granularity for the same variable. While sampling over a longer time horizon lets us use a coarser grid, it comes at the cost of increasing the number of interface inputs. States θ and ω at time $t = 0$ can influence state p_x at future time steps through their effect on v_x. While p_x can depend on θ and ω over time, its sensitivity to those values is small over short horizons. Heterogeneous grids exploit this insight and allow small perturbations of θ and ω to be ignored. Table 5.2 shows how interface F_{p_x} allocates less bits to those variables

Interface Name	Output Variable	Input Variables
F_{p_x}	$p_x^+ \leftarrow 8$	$p_x \leftarrow 8$, $v_x \leftarrow 6$, $\theta \leftarrow 3$, $\omega \leftarrow 0$, u_1, u_2
F_{p_y}	$p_y^+ \leftarrow 8$	$p_y \leftarrow 8$, $v_y \leftarrow 6$, $\theta \leftarrow 3$, $\omega \leftarrow 0$, u_1, u_2
F_{v_x}	$v_x^+ \leftarrow 6$	$v_x \leftarrow 6$, $\theta \leftarrow 5$, $\omega \leftarrow 4$, u_1, u_2
F_{v_y}	$v_y^+ \leftarrow 6$	$v_y \leftarrow 6$, $\theta \leftarrow 5$, $\omega \leftarrow 4$, u_1, u_2
F_θ	$\theta^+ \leftarrow 5$	$\theta \leftarrow 5$, $\omega \leftarrow 4$, u_2
F_ω	$\omega^+ \leftarrow 4$	$\omega \leftarrow 4$, u_2

Table 5.2: Each row represents an interface of the unrolled lunar lander dynamics. The notation $p_x \leftarrow 8$ signifies that p_x is a continuous variable and the interface has allocated 8 bits to it. Different interfaces can have different views of the same variable, e.g. interface F_ω allocates 4 bits to ω while F_{p_x} allocates none. Inputs u_1 and u_2 are discrete so they do not adopt this notation.

than interfaces F_θ and F_ω, which are much more sensitive to the same perturbations over the same time horizon. Curiously, F_{p_x} assigns zero bits to ω. This signifies that it only assumes that $\omega \in [-.6, .6]$ but does not otherwise care what the specific value is due to the low sensitivity.

Arterial Traffic Network

An arterial traffic corridor is a high capacity local road that is actuated by signalized intersections.

The traffic network can be represented as directed graph with a set $\mathcal{E}$ of edges or road links and a set $\mathcal{V}$ of vertices or intersections. Every edge e is a directed arc from tail vertex $\tau(e) \in \{\mathcal{V} \cup \varnothing\}$ to head vertex $\eta(e) \in \mathcal{V}$. By convention, an edge with $\tau(e) = \varnothing$ serves as entry point to the network and edges that outflow the network are not explicitly modeled. Let

$$\mathcal{E}_v^{\text{in}} = \{e \in \mathcal{E} \mid \eta(e) = v\}, \text{ and} \tag{5.30}$$

$$\mathcal{E}_v^{\text{out}} = \{e \in \mathcal{E} \mid \tau(e) = v\} \tag{5.31}$$

denote the incoming and outgoing edges of vertex v , respectively. The set of local edges to edge e is given by

$$\mathcal{E}^{\text{local}}(e) = \mathcal{E}_{\tau(e)}^{\text{in}} \cup \mathcal{E}_{\tau(e)}^{\text{out}} \cup \mathcal{E}_{\eta(e)}^{\text{out}}. \tag{5.32}$$

Each edge $e \in \mathcal{E}$ has an associated link occupancy $x_e \in [0, \bar{x}_e]$ at time step $t \in z_{\geq 0}$, where the constant $\bar{x}_e$ denotes the maximum vehicular capacity of edge e . Then, the n-dimensional state space of the traffic model is

$$\mathcal{X} = \prod_{e \in \mathcal{E}} [0, \bar{x}_e] \subset \mathbb{R}^n, \tag{5.33}$$

where $n = |\mathcal{E}|$. An edge e is actuated if the traffic signal at vertex $\eta(e)$ permits outward flow from e to $\mathcal{E}^{\text{out}}_{\eta(e)}$. The controllable input u_e associated with edge e is given by

$$u_e = \begin{cases} 1 & \text{if } \eta(e) \text{ actuates } e \text{ at time step } t \\ 0 & \text{otherwise} \end{cases}. \tag{5.34}$$

The constant split ratio $\beta_{ek} \in [0, 1]$ denotes the fraction of vehicles flowing from edge e to edge $k \in \mathcal{E}^{\text{out}}_{\eta(e)}$. However, edge e can only send vehicles to k if k offers enough capacity. For this purpose, the constant supply ratio $\alpha_{ek} \in [0, 1]$ denotes the capacity fraction of k dedicated to e. Since the supply is split among actuated incoming edges, it follows that for all $k \in \mathcal{E}$ and time steps t

$$\sum_{e \in \mathcal{E}^{\text{in}}_{\tau(k)}} \alpha_{ek} u_e = 1. \tag{5.35}$$

In the following, we introduce the system dynamics based on the modified cell transmission model, which exhibits FIFO property. As we will see, the outflow and state update function of edge e depends only on the corresponding states of edges in $\mathcal{E}^{\text{out}}_{\eta(e)}$ and $\mathcal{E}^{\text{local}}(e)$, respectively. Thus, the traffic network exhibits a sparsity property, which we can exploit to compute abstractions more efficiently.

The flow out of edge $e \in \mathcal{E}$ is given by

$$f^{\text{out}}_e = u_e \min \left(x_e, c_e, \min_{k \in \mathcal{E}^{\text{out}}_{\eta(e)}} \left\{ \frac{\alpha_{ek}}{\beta_{ek}} (\bar{x}_k - x_k) \right\} \right). \tag{5.36}$$

The minimization in Equation (5.36) implies that the flow of exiting vehicles of edge e cannot exceed the current link occupancy, the constant saturation flow c_e, and the resulting supply offered by edges in $\mathcal{E}^{\text{out}}_{\eta(e)}$. The saturation flow c_e is a model parameter, which is determined by, e.g., number of lanes and speed limits. The inflow of edge e is denoted by

$$f^{\text{in}}_e = \sum_{k \in \mathcal{E}^{\text{in}}_{\tau(e)}} \beta_{ke} f^{\text{out}}_k. \tag{5.37}$$

Based on the principle of mass conservation, the discrete-time update equation of state x_e associated with edge e is

$$x^+_e = \min \left\{ \bar{x}_e, x_e - f^{\text{out}}_e + f^{\text{in}}_e + d_e \right\}, \tag{5.38}$$

where $d_e \in [0, \bar{d}_e]$ represents a time-dependent exogenous flow entering edge e , and the constant parameter $\bar{d}_e$ is an upper bound. It is clear based on Equations (5.36) to (5.38), that the state update equation of edge e depends only on the current states of edges in $\mathcal{E}^{\text{local}}(e)$. Therefore, we can reduce the computation time of the abstraction drastically by exploiting this sparse interconnection structure, as shown subsequently.

Arterial Traffic Corridor

We consider an easily extensible network to show the applicability of the decomposed abstraction approach to more than three dimensions. The n-dimensional traffic network is depicted in Figure 5.13 on the left side of the big plus sign.

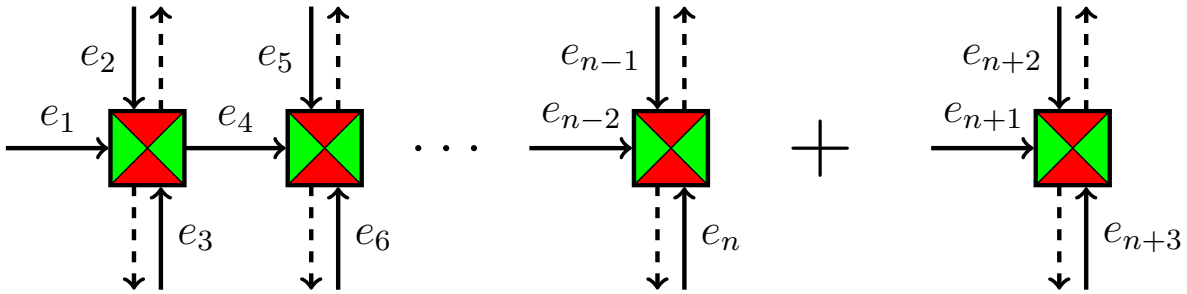


Figure 5.13: Addition of a new block, consisting of three edges and one vertex, to the existing n-dimensional traffic network. The dashed edges represent not explicitly modeled road links.

The dashed arrows represent road links that exit the network and are not explicitly modeled. Since we want to evaluate the performance of our approach with respect to the dimension of the state and input space, we consecutively add new blocks to the existing n-dimensional network, as shown in Figure 5.13. Each newly added block consists of three edges and one vertex, i.e., three road links and one signalized intersection. Thus, the dimension of the state space in Equation (5.33) is increased by 3 per block. The input space grid doubles, since we assume that every vertex actuates either the incoming arterial road edge or the two North-South links.

The used parameters of the n-dimensional vehicular traffic network in Figure 5.13 are as follows. The supply ratio α_{e_i,e_j} is equal 1 and the split ratio β_{e_i,e_j} is 0.8 if $i+3=j$ for $j \in \{4,7,\ldots,n-2\}$. Otherwise, both ratios are equal 0.5. The maximum vehicular capacity $\bar{x}_e$ is 10 for all edges. Every state dimension is quantized in 10 equally spaced intervals. Thus, the considered n-dimensional network has 10^n state space grid points. For all arterial road edges, the upper disturbance bound $\bar{d}_e$ is 0 and the saturation flow c_e is 6. For all North-South edges, the bound is 1 and the corresponding flow is 2.

Results

Since the vehicular traffic network in Figure 5.13 is a monotone system, we can use a straightforward modification of Equation (5.18) as an over-approximation function to account for exogenous flow entering the network. The time needed to compute the finite abstraction with respect to the dimension of the state space is illustrated in Figure 5.14. All computations are run on a single thread of an Intel Core i7-5500U (3.00GHz) with 8GB RAM.

Our proposed abstraction algorithm performs roughly linearly with respect to the state space dimension, although the total number of transitions grows exponentially. For instance,

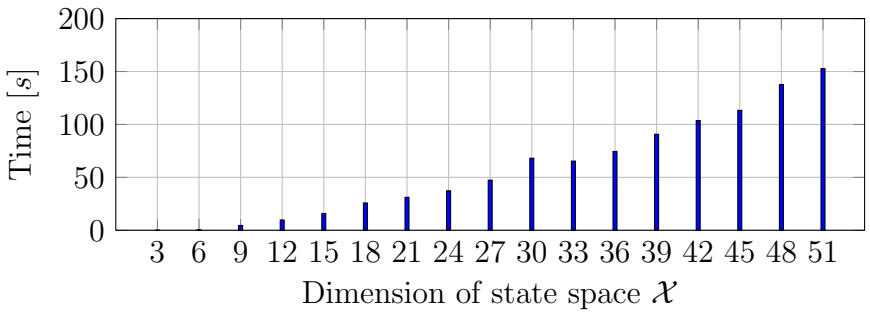


Figure 5.14: Computation time of the transition function BDD.

there are 3.3×10^{62} valid transitions in the 51-dimensional traffic network. The nonlinear effects are mostly due to the variable reordering heuristics of the `CUDD` BDD library [84].

The file size of the resulting transition function binary decision diagram (BDD) with respect to the dimension of the state space is shown in Figure 5.15. Due to the memory-

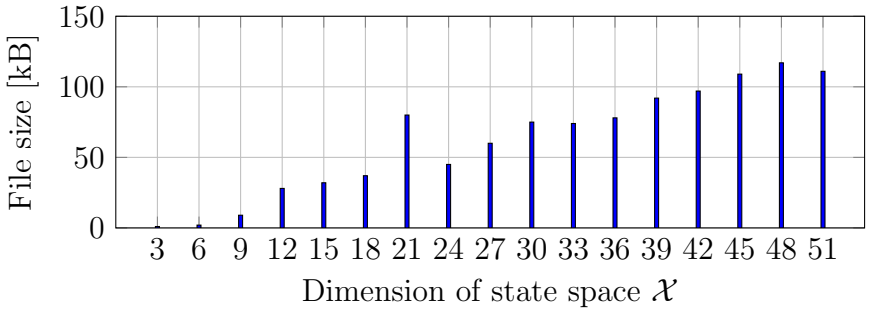


Figure 5.15: File size of the transition function BDD.

efficient data structure, the size of the BDD that encodes the transition function grows also roughly linearly.

Based on the discrete abstraction, we can use the safety fixed point iteration in Equation (3.27) to synthesize a memoryless state-feedback controller for the arterial traffic corridor in Figure 5.13. For instance, assume in the following that the occupancy of every edge has to be smaller than $\frac{4}{5}$ of the maximum vehicular capacity for all time steps, i.e., in our example less than 8. After representing this safety specification as a BDD, we obtain a controller for the 12-dimensional network after 46 iterations of the fixed point algorithm. The synthesis takes 2.5 minutes and 0.9MB are needed to store the obtained controller as BDD. This BDD represents all allowed inputs for each discrete state such that the formal specification is fulfilled. In this example, there exist more than 2.1×10^{11} allowed state-input combinations. Instead of using the memory-efficient BDD data structure, it is possible to store the controller as sparse matrix lookup table by explicitly enumerating the state and input space. Based on the given dimensions 44 bits are needed to store a single allowed state-input combination, resulting in 1.2TB of memory usage in contrast to 0.9MB for the BDD.

In contrast to the 12-dimensional network, the 15-dimensional case takes 4.3 hours until convergence. This exponential computation time increase has multiple reasons. In general, contrary to the coordinate-wise, conjunctive representation of the transition function, a controller cannot be represented in such a compressed form. Furthermore, the fixed point algorithm determines all allowed state-input combinations. Since the number of discrete control actions for every state grows exponentially with respect to the arterial corridor length, the controller synthesis problem becomes intractable.

Part II

Distributed Control Synthesis

Motivating Example: Distributed Traffic Control

Intelligent traffic management is an effective method to mitigate economic and environmental costs of transportation without upgrading the underlying infrastructure [69]. The traditional mechanism of traffic control in an urban setting is coordinating traffic lights. In freeway networks, traffic is controlled using ramp meters or dynamic speed limits. Control-theoretic aspects of traffic management have been widely studied in the literature [11, 20, 56, 92, 45, 36, 53].

Designing a traffic network controller requires careful tradeoffs among different objectives.

- **Safety:** Never experience severe congestion.
- **Liveness:** Freeway onramps and intersection queues must persistently be cleared.
- **Reactivity:** Whenever congestion appears, it will eventually dissipate.

The above specifications can be encoded in temporal logic. Formal control synthesis for vehicular traffic networks has been tackled by Coogan et al. [23] by constructing discrete abstractions and in by Sadraddini et al. [79] [80] by posing a receding horizon control problem as a mixed integer linear program.

Both approaches exhibit computational bottlenecks as the traffic network size grows. In the finite abstraction approach the number of states is exponential with respect to the number of traffic links. Similarly, larger networks induce a larger number of decision variables in the receding horizon approach. Because traffic networks can easily be expanded by adding additional roads, they are a prime candidate for benchmark problems in formal synthesis [21].

To combat the growth in computational complexity, both Kim et al. [49] and Sadraddini et al. [82] exploit locality properties of traffic networks. By decomposing into a collection of smaller networks, existing controller synthesis algorithms can be applied to problems with reduced state space, input space, and simpler specifications. In addition to computational complexity, there are numerous practical reasons for decomposition in the context of a freeway network interacting with an arterial corridor. They are often under the control of different government agencies and exhibit different control modalities. The California Department of Transportation (Caltrans) is a state agency that maintains jurisdiction over freeways and ramp metering. Traffic signals in local and arterial intersections are under the jurisdiction of cities.

While solving a controller synthesis problem for a smaller network is computationally simpler, naively stitching together the individual controllers does not guarantee that the global network will satisfy its desired specification. The actions of individual sub-network controllers can affect the dynamics of adjacent neighborhoods. Indeed, the *lack* of any coordination has resulted in congestion and suboptimal performance at neighborhood boundaries in the real world. Su et al. [85] studied the interaction between a real world signalized intersection and a freeway ramp meter. Figure 5.16 depicts a metered freeway onramp ramp and

an upstream signalized intersection in San Jose, California. The study states that there is a "strong dynamic interaction" and a "conflict of interest" between the two during peak hours when the onramp becomes a bottleneck. They also emphasize the "[necessity] to coordinate between [arterials and freeways]". By modifying the signal green times to take into account the onramp occupancy, the delay at the intersection was reduced.

A Distributed Approach to Formal Traffic Control Synthesis

This half of the thesis generalizes the insights behind Su et al.'s freeway-arterial coordination study [85] to more general systems networks and temporal logic specifications. We build the core mathematical machinery required for distributed formal controller synthesis. Consider two interacting networks $\mathcal{N}_1$ and $\mathcal{N}_2$ each with their own local specifications ϕ_1 and ϕ_2. The "global" specification is assumed to be $\phi_1 \wedge \phi_2$. While a controller can be synthesized for $\mathcal{N}_1$ such that it satisfies ϕ_1, the validity of that controller is predicated on the actions of the $\mathcal{N}_2$ controller and vice versa. Thus we run into a circular dependency resembling the problem of "the chicken or the egg".

To prevent controllers from inadvertently causing others to violate their specification, they need to be augmented with additional promises that regulate their interaction. These promises will come in the form of an assumption-guarantee contract. An individual network may *assume* that adjacent networks will restrict their behaviors to an a priori agreed upon set while simultaneously *guaranteeing* to those networks that its local behavior will be restricted. Under this assume-guarantee framework, controllers that were initially designed in isolation can be systematically recombined and provide certificates about the global system behavior.

Chapter 6 introduces assume-guarantee contracts and focuses on those contracts that are in parametric form. A parameter domain corresponds to a collection of different scenarios such as "severe rush hour traffic" or "diverted traffic from an accident". Parametric assume-guarantee reasoning rules are introduced to reason about this domain and also to derive guarantees on global behavior when components satisfying parametric contracts are interconnected. We also show how parametric assume-guarantee reasoning can capture the classical small-gain theorem for continuous systems.

Chapter 7 takes the core notions behind parametric assume-guarantee contracts and repurposes them as dynamic contracts for control synthesis. A static contract encodes a promise among agents that does not change with time. In practice this may lead to sub-optimal and conservative behaviors because the static contract cannot react to conditions at runtime. Dynamic contracts balance the tradeoff between performance and ensuring guarantees about runtime behavior.

A coordinator does not always need to be active, especially when a specification is not in immediate danger of being violated. In Chapter 8, we propose a method to identify when the coordinator needs to be active when multiple systems are unable to enforce a safety constraint without communicating with one another. In an example, this technique is used to identify when it is useful for two vehicles to coordinate with one another when both approach an intersection without a traffic signal.

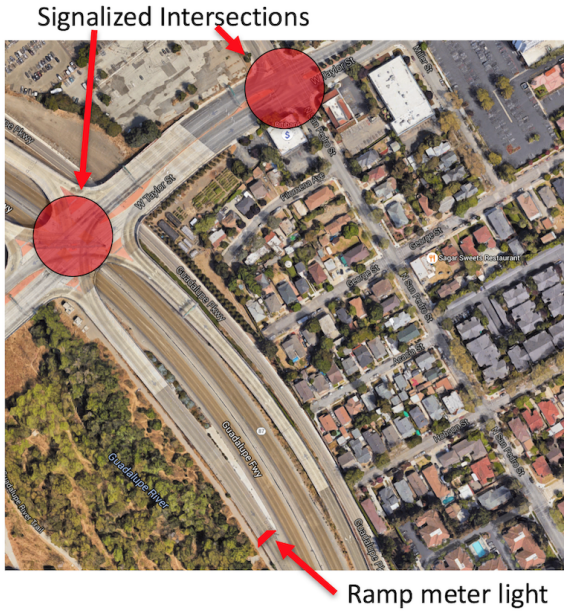


Figure 5.16: Freeway-arterial interaction at the intersection studied by Su et al. [85]. A long freeway ramp is actuated by a ramp meter that sets a rate that vehicles may enter the freeway. During periods of high congestion on the freeway, this rate decreases and the vehicle queue may spill back onto the signalized intersections upstream. Satellite Image Source: Google Maps.

Related Work

Recent approaches to tackling the scalability of control synthesis have focused on decomposing a system into smaller components, synthesizing a controller for each component locally, then recombining them using assume-guarantee reasoning. Each component can be annotated with assume-guarantee contracts that captures the component's behaviors in response to various environments. By carefully taking into account the interconnection topology, dynamics, and specification it is possible to ensure that the local controllers collectively are able to satisfy the original specification.

Assume-guarantee reasoning has been extensively used in compositional verification and design [41, 67]. Control synthesis with assume-guarantee contracts is a less mature field. For continuous spaces, contracts came in the form of a collection of co-designed invariant sets such as zonotopes by Nilsson et al. [64] or directed sets by Kim et al. [50]. Li et al. [54] and Jin et al. [47] have respectively mined assumptions and requirements as temporal logic formulas. There are still some open fundamental questions on on how to model and incorporate assumptions within the synthesis procedure [10, 17]. Assume-guarantee reasoning has also be applied in an extensive range of applications ranging from building automation [61] and path planning for robotics [30, 31].

Numerous recent advances in distributed control synthesis incorporate or generalize core concepts from robust control. Tarraf et al. [88] and Tabuada et al. [87] have both incor-

porated notions of robustness for systems with discrete input-output alphabets. Small-gain arguments were leveraged by Rungger and Zamani [77] to decompose the construction of approximate discrete abstractions of continuous systems and Dallal and Tabuada [26] to design customized decomposed abstractions for persistency specifications.

Chapter 6

Parametric Assume-Guarantee Reasoning

6.1 Assume-Guarantee Contracts

The contracts framework by Benveniste et al. [8] constitutes a formalism for (de)composition oriented design of cyber-physical systems. Contracts capture system requirements and behaviors and can be used to systematically reason about interconnections and implementations. Potential use cases include quickly identifying inconsistent requirements, enabling modularity and interchanability.

Assume-guarantee contracts are one specific formalism for contracts where assumptions characterize the collection of valid environments of a component, while guarantees specify the component's commitment to enforce a subset of behaviors whenever the assumption is satisfied. This chapter focuses on contracts for verification and deriving guarantees on system behavior once components are interconnected. Chapter 7 then uses the assume-guarantee framework to certify that the composition of distributed controllers satisfies a desired property.

In this chapter we view a system F as a collection of input-output behaviors $F \subseteq \mathcal{U}[\cdot] \times \mathcal{Y}[\cdot]$.

Definition 21 (Assume-Guarantee Contract for a Dynamical System)**.** *An assume-guarantee contract is a pair $\mathcal{C} = (\phi_a, \phi_g)$ consisting of an assumption $\phi_a \subseteq \mathcal{U}[\cdot] \times \mathcal{Y}[\cdot]$ and a guarantee $\phi_g \subseteq \mathcal{U}[\cdot] \times \mathcal{Y}[\cdot]$ that encodes the requirement that the logical implication $\phi_a \Rightarrow \phi_g$ holds.*

A system satisfies $\mathcal{C} = (\phi_a, \phi_g)$ if $F \cap \phi_a \subseteq \phi_g$ (where ϕ_a, ϕ_g are viewed as sets). Note that an assume-guarantee specification is automatically satisfied if the assumptions are violated. It can only be violated when the assumption is true and the guarantee is false.

Interfaces vs. Contracts

The theories of assume-guarantee contracts [8] and relational interfaces both are designed to be methods to reason about components and their interconnections. While similar in spirit, they contain some subtle philosophical differences. Interface variables are each associated with input and outputs, while no such distinction is made in the core theory on assume-guarantee contracts. While relational interfaces can be translated into an equivalent contract, the input-output annotations for variables are lost. While the transformation preserves refinement the same cannot be said about serial composition and conjunction as demonstrated by Nuzzo [66, 68]. A new assumption projection operator is proposed to preserve serial composition and compatibility checking. Finally, assumption violations or errors are handled differently. Sink interfaces were used to model requirements and raises errors whenever the inputted values caused blocking. In contrast, when contract assumptions are violated this simply releases the component from needing to impose a guarantee. The counterpart for shared refinement operation for relational interfaces is conjunction for contracts. Conjunction for contracts is always defined, whereas the shared refinement only exists when interfaces satisfy the shared refinability condition.

Contracts in this section are used to represent temporal logic properties instead of stateless input-output relations, which was the case for relational interfaces in the first half of this thesis.

Parametric Assume-Guarantee Contracts

Most assume-guarantee contracts make worst case assumptions about an environment's behavior at design time. A system's guarantee as a result is coarse in order to compensate for the uncertainty about which environment a system will experience once deployed.

A parametric input-output specification $\psi : \mathcal{P} \to 2^{\mathcal{U}[\cdot]\times\mathcal{Y}[\cdot]}$ is a collection of specifications indexed by elements in a parameter space $\mathcal{P}$. Parametric input specifications and output specifications are defined analogously.

In order to make guarantees more precise, we use parametric specifications to divide the assumption ϕ_a into smaller regions $\psi_a(p_a)$, where each p_a can be thought of as an "environmental scenario" parameterized over a set $\mathcal{P}_a$. Systems can provide a finer guarantee $\psi_g(p_g)$ in response to a smaller set of environment behaviors $\psi_a(p_a)$, as depicted for example in Figure 6.1. The relationship between the assumption and the guarantee is specified by a parameter map $\lambda : \mathcal{P}_a \to \mathcal{P}_g$.

Definition 22 (Parametric Assume-Guarantee Contracts)**.** *An assume-guarantee contract* (ϕ_a, ϕ_g) *is in parametric form if there exist parameter spaces* $\mathcal{P}_a$ *and* $\mathcal{P}_g$*, parametric specifi-*

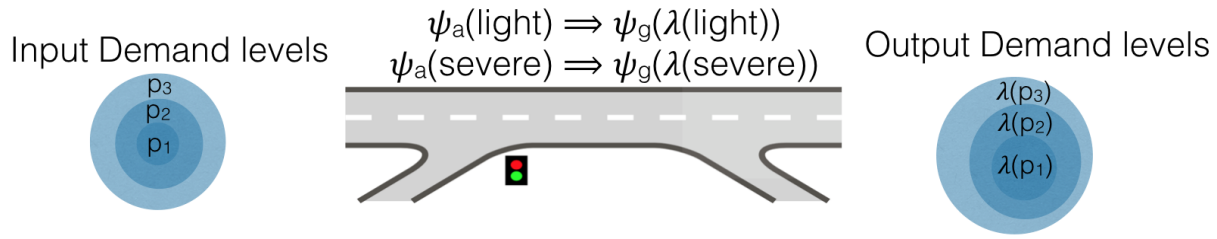


Figure 6.1: Parametric contracts can be used to encode different environmental scenarios such as the input demand induced on a road segment. The output demand is a function of the input demand.

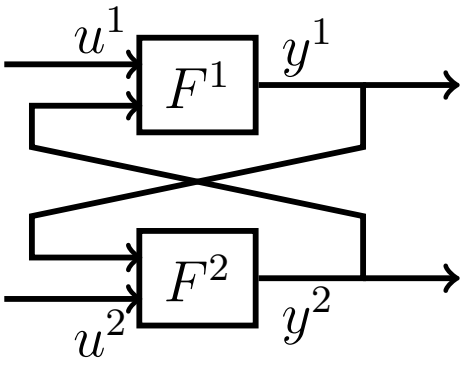


Figure 6.2: An interconnection with exogenous environment and feedback.

cations $\psi_a(\cdot), \psi_g(\cdot)$*, and a map* $\lambda : P_a \to P_g$ *such that*

$$\phi_a = \bigvee_{p \in \mathcal{P}_a} \psi_a(p) \tag{6.1}$$

$$\phi_g = \bigwedge_{p \in \mathcal{P}_a} \psi_a(p) \Rightarrow \psi_g(\lambda(p)) \tag{6.2}$$

A parametric assume-guarantee contract is itself a special instance of a regular assume-guarantee contract, but is generated via a collection of contracts associated with a parameter space. The parametric form resembles the shared refinement operation from relational interfaces.

6.2 A Small Gain Theorem For Parametric Assume-Guarantee Contracts

Consider the interconnection in Figure 6.2, which contains an exogenous environment and a feedback loop. Suppose for each system F^i where $i \in \{1, 2\}$ the input space $\mathcal{U}^i = \mathcal{U}^i_e \times \mathcal{U}^i_f$

is partitioned into an exogenous environment $\mathcal{U}_e^i$ component and a feedback component $\mathcal{U}_f^i$. In order for the interconnection to be valid the spaces must match $\mathcal{U}_f^1 = \mathcal{Y}^2$ and $\mathcal{U}_f^2 = \mathcal{Y}^1$. The assume-guarantee contract framework [8] in its full generality ignores the roles of ports as inputs and outputs. With Figure 6.2 in mind, we place the following restriction on assumptions and guarantees.

The assumption parameter space is partitioned into two components $\mathcal{P}_a^i = \mathcal{P}_{ae}^i \times \mathcal{P}_{af}^i$ and the parametric input assumptions $\psi_a : \mathcal{P}_{ae}^i \times \mathcal{P}_{af}^i \mapsto 2^{\mathcal{U}^i[\cdot]}$ is the conjunction of two components

$$\psi_a^i(p_{ae}^i, p_{af}^i) := \psi_{ae}^i(p_{ae}^i) \wedge \psi_{af}^i(p_{af}^i). \tag{6.3}$$

The parameter map $\lambda^i : \mathcal{P}_{ae}^i \times \mathcal{P}_{af}^i \mapsto \mathcal{P}_g^i$ is adjusted to account for this decomposition of the input space.

Main Results

Due to the feedback loop in Figure 6.2, a few additional assumptions are required to derive a new assume-guarantee contract for the interconnected system. First, the exogenous environment and internal feedback assumptions for at least one system need to be satisfied. Second, the guarantees from one system need to imply that the assumptions for the other system hold.

Theorem 6. *Consider the interconnection of two systems F^1, F^2 depicted in Figure 6.2. We assume the following.*

1. *Both systems satisfy their parametric assume guarantee contracts. That is for $i \in \{1, 2\}$, $F^i \cap \phi_a^i \subseteq \phi_g^i$ where ϕ_a^i and ϕ_g^i are defined as*

$$\phi_a^i = \bigvee_{(p_{ae}^i, p_{af}^i) \in \mathcal{P}_a^i} \left(\psi_{ae}^i(p_{ae}^i) \wedge \psi_{af}^i(p_{af}^i)\right) \tag{6.4}$$

$$\phi_g^i = \bigwedge_{(p_{ae}^i, p_{af}^i) \in \mathcal{P}_a^i} \left(\psi_{ae}^i(p_{ae}^i) \wedge \psi_{af}^i(p_{af}^i) \Rightarrow \psi_g^i(\lambda^i(p_{ae}^i, p_{af}^i))\right). \tag{6.5}$$

2. *The guarantee parameter spaces are subsets of the feedback components of the assumption parameter spaces, i.e., $\mathcal{P}_g^2 \subseteq \mathcal{P}_{af}^1$ and $\mathcal{P}_g^1 \subseteq \mathcal{P}_{af}^2$. Moreover, the guarantee $\psi_g^2(\cdot)$ from one system implies the feedback assumption $\psi_{af}^1(\cdot)$ and vice versa:*

$$\forall p \in \mathcal{P}_g^1 \quad \psi_g^2(p) \Rightarrow \psi_{af}^1(p) \tag{6.6}$$

$$\forall p \in \mathcal{P}_g^2 \quad \psi_g^1(p) \Rightarrow \psi_{af}^2(p). \tag{6.7}$$

This condition is trivially satisfied if $\psi_g^2(\cdot) = \psi_{af}^1(\cdot)$ and vice versa.

3. *There exist environment parameters* $p^1_{ae} \in \mathcal{P}^1_{ae}$ *and* $p^2_{ae} \in \mathcal{P}^2_{ae}$ *such that* $\psi^1_{ae}(p^1_{ae})$ *and* $\psi^2_{ae}(p^2_{ae})$ *are satisfied.*

4. *For either* $i = 1, 2$ *there exists a feedback parameter* $p^i_{af}[0] \in \mathcal{P}^i_{af}$ *such that* $\psi^i_{af}(p^i_{af}[0])$ *is true.*

Without loss of generality let $i = 1$*, define new feedback parameter maps* $\hat{\lambda}^1(\cdot) = \lambda^1(p^1_{ae}, \cdot)$ *and* $\hat{\lambda}^2(\cdot) = \lambda^1(p^2_{ae}, \cdot)$ *associated with exogenous environment assumptions* p^1_{ae}, p^2_{ae}*, and define guarantee parameter iterations*

$$p^1_g[k+1] = (\hat{\lambda}^1 \circ \hat{\lambda}^2)(p^1_g[k]) \tag{6.8}$$

$$p^2_g[k+1] = (\hat{\lambda}^2 \circ \hat{\lambda}^1)(p^2_g[k]) \tag{6.9}$$

with initializations $p^1_g[0] = \hat{\lambda}^1(p^1_{af}[0])$*,* $p^2_g[0] = \hat{\lambda}^2(p^1_g[0])$*. Then the guarantee simplifies to*

$$\bigwedge_{k=0}^{\infty} \psi^1_g\left(p^1_g[k]\right)\right) \wedge \bigwedge_{k=0}^{\infty} \psi^2_g\left(p^2_g[k]\right). \tag{6.10}$$

For the case when $i = 2$ *then a similar guarantee can be obtained by switching the indexes in (6.8), (6.9), and (6.10).*

Proof. Without loss of generality let $i = 1$. The existence of satisfying p^1_{ae}, p^2_{ae} and $p^1_{af}[0]$ ensure that we can bootstrap an infinite sequence of implications from (6.5), (6.6) and (6.7). The parameters in this implication are generated from the sequences (6.8) and (6.9).

$$\begin{aligned}
&\psi^1_{ae}(p^1_{ae}) \wedge \psi^2_{ae}(p^2_{ae}) \wedge \psi^1_{af}(p^1_{af}[0]) \\
&\wedge \left(\psi^1_{ae}(p^1_{ae}) \wedge \psi^1_{af}(p^1_{af}[0]) \Rightarrow \psi^1_g(p^1_g[0])\right) \\
&\wedge \left(\psi^1_g(p^1_g[0]) \Rightarrow \psi^2_a(p^1_g[0])\right) \\
&\wedge \left(\psi^2_{ae}(p^2_{ae}) \wedge \psi^2_{af}(p^1_g[0]) \Rightarrow \psi^2_g(p^2_g[0])\right) \\
&\wedge \left(\psi^2_g(p^2_g[0]) \Rightarrow \psi^1_a(p^2_g[0])\right) \\
&\wedge \left(\psi^1_{ae}(p^1_{ae}) \wedge \psi^1_{af}(p^2_g[0]) \Rightarrow \psi^1_g(p^1_g[1])\right) \\
&\wedge \ldots
\end{aligned}$$

This infinite conjunction sequence contains within it (6.10) as a subsequence. □

A Small Gain Theorem for Hausdorff Continuous Parametric Specifications

One of the weaknesses of Theorem 6 is that the guarantee (6.10) may be difficult to reason about since it is an infinite conjunction of statements. The guarantee (6.10) can be simplified dramatically by investigating the contraction properties of the parameter iterations (6.8) and

(6.9). To achieve this simplification we assume that the parameter sets $\mathcal{P}_g^i$ for $i = 1, 2$ are equipped with distance metrics $d_{\mathcal{P}}^i : \mathcal{P}_g^i \times \mathcal{P}_g^i \mapsto \mathbb{R}_{\geq 0}$ and the input and output spaces $\mathcal{U}[\cdot] \times \mathcal{Y}[\cdot]$ are equipped with a distance metric. We opt for the Hausdorff pseudo-metric to measure the difference between specifications. Given a metric $d : \mathcal{X}[\cdot] \times \mathcal{X}[\cdot] \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ between signals, the Hausdorff distance $d_H(\phi_a, \phi_b)$ between $\phi_a \subseteq 2^{\mathcal{X}[\cdot]}$ and $\phi_b \subseteq 2^{\mathcal{X}[\cdot]}$ is

$$d_H(\phi_a, \phi_b) = \max \left(\sup_{a[\cdot] \in \phi_a} \inf_{b[\cdot] \in \phi_b} d(a[\cdot], b[\cdot]), \sup_{b[\cdot] \in \phi_b} \inf_{a[\cdot] \in \phi_a} d(a[\cdot], b[\cdot]) \right) \tag{6.11}$$

If $d_H(\phi_a, \phi_b) < \epsilon$, then for each signal that satisfies ϕ_a there exists a signal that satisfies ϕ_b that is at most ϵ away and vice versa. The Hausdorff distnace can assume infinite values and is a pseudo-metric because $d_H(\phi_a, \phi_b) = 0$ implies $\mathrm{cl}\phi_a \equiv \mathrm{cl}\phi_b$ rather than $\phi_a \equiv \phi_b$. A parametric specification with a metric-equipped parameter space is Hausdorff continuous if it satisfies the standard $\epsilon - \delta$ definition. That is, any arbitrarily small bound on the Hausdorff distance between specifications can be enforced for a sufficiently small parameter difference.

Theorem 7. *(Small Gain Theorem for Parametric Assume-Guarantee Contracts) Let $\mathcal{P}_g^1$ and $\mathcal{P}_g^2$ be metric spaces and $\psi_g^1(\cdot), \psi_g^2(\cdot)$ be specifications on a metric space. If in addition to the assumptions of Theorem 6 the following are also true:*

1. *Sequences generated by the iterations (6.8), (6.9) have nonempty $\omega-$limit sets W^1, W^2 respectively.*

2. *The specifications ψ_g^1, ψ_g^2 vary continuously with parameters everywhere in $\mathcal{P}_g^1, \mathcal{P}_g^2$, where the Hausdorff distance d_H is used as a metric between specifications. In other words for both $i = 1, 2$ for all $\epsilon^i > 0$ and $p \in \mathcal{P}_g^i$ there exists a $\delta^i > 0$ such that*

$$d_{\mathcal{P}}^i(p, \hat{p}^i) < \delta^i \Rightarrow d_H(\psi_g^i(p), \psi_g^i(\hat{p}^i)) < \epsilon^i$$

then the guarantee (6.10) is over-approximated by:

$$\bigwedge_{p^1 \in W^1} \mathrm{cl}\psi_g^1(p^1) \wedge \bigwedge_{p^2 \in W^2} \mathrm{cl}\psi_g^2(p^2) \tag{6.12}$$

where $\mathrm{cl}\psi$ is the closure of the specification set ψ.

Proof. Without loss of generality, we seek to prove that the ψ_g^1 component of formula (6.10) implies $\mathrm{cl}\psi_g^1(p^1)$ for a p^1 in the ω-limit set W^1. Suppose $\epsilon > 0$. By Hausdorff continuity of ψ_g^1, there exists a δ such that $|p - p^1| < \delta$ implies $d_H(\psi_g^1(p), \psi_g^1(p^1)) < \epsilon$. Because $p^1 \in W^1$, there is a subsequence of (6.8) that converges to p^1 and for arbitrary δ. It follows that (6.10) consists of an infinite sequence of intersections that converge to $\psi_g^1(p^1)$ with arbitrary precision. Because of Hausdorff distance of zero implies that the closure of the two sets are equivalent, this infinite intersection then implies that $\mathrm{cl}\psi_g^1(p^1)$ holds. Similar arguments can be made for any $p^1 \in W^1$ and for ψ_g^2. □

If the iterations (6.8) and (6.9) are contractions to a single point, then we can declare a new parametric assume-guarantee contract for the interconnected system in Figure 6.2 that is expressed between the exogenous inputs and the outputs.

Corollary 2. *If in addition to the assumptions of Theorem 7, the iterations (6.8) and (6.9) globally converge to fixed points for any initial parameters $p^1_{af}[0], p^2_{af}[0]$ then the interconnected system of Figure 6.2 satisfies the parametric assume-guarantee contract associated with*

$$\begin{aligned}
\mathcal{U} &:= \mathcal{U}^1_e \times \mathcal{U}^2_e \\
\mathcal{Y} &:= \mathcal{Y}^1 \times \mathcal{Y}^2 \\
\mathcal{P}_a &:= \mathcal{P}^1_{ae} \times \mathcal{P}^2_{ae} \\
\mathcal{P}_g &:= \mathcal{P}^1_g \times \mathcal{P}^2_g \\
\psi_a(p^1_{ae}, p^2_{ae}) &:= \psi^1_{ae}(p^1_{ae}) \wedge \psi^2_{ae}(p^2_{ae}) \\
\psi_g(p^1_g, p^2_g) &:= \mathrm{cl}\psi^1_g(\lambda^1(p^1_{ae}, p^2_{ae})) \wedge \mathrm{cl}\psi^2_g(\lambda^2(p^1_{ae}, p^2_{ae})))
\end{aligned}$$

and $\lambda^1 : \mathcal{P}^1_{ae} \times \mathcal{P}^2_{ae} \mapsto \mathcal{P}^1_g$ and $\lambda^2 : \mathcal{P}^1_{ae} \times \mathcal{P}^2_{ae} \mapsto \mathcal{P}^1_g$ are the respective limit points of the iterations (6.8) and (6.9) as a function of exogenous environment assumptions p^1_{ae} and p^2_{ae}.

Ensuring that Guarantees are Satisfiable

One technical issue with applying Theorem 7 to richer sets of behaviors is determining whether the guarantees (6.10) or (6.12) are nonempty sets. It is advantageous to design parametric specifications to ensure that satisfiability is maintained.

We link parameters to set containment through the notion of monotone specifications. Jin et al. [47] have previous used them in the context of requirement mining. Given a partially ordered parameter space $\mathcal{P}^1_g$ equipped with an ordering relation $\leq_{\mathcal{P}^1_g}$, the parametric output specification $\psi^1_g : \mathcal{P}^1_g \mapsto 2^{\mathcal{Y}[\cdot]}$ is monotone if $a \leq_{\mathcal{P}^1_g} b$ implies $\psi^1_g(a) \subseteq \psi^1_g(b)$.

Proposition 6 uses these notions of monotonicity and set containment to give a sufficient condition for the guarantees to be nonempty.

Proposition 6. *Suppose that*

1. *For both $i = 1, 2$ and all nonempty subsets $\mathcal{L}$ of parameter space $\mathcal{P}^i_g$, there exists a lower bound $p \in \mathcal{P}^i_g$ such that $p \leq_{\mathcal{P}^i_g} q$ for all $q \in \mathcal{L}$.*

2. *Parametric output guarantees $\psi^1_g(\cdot)$ and $\psi^2_g(\cdot)$ are monotone specifications and for all parameters $p^1 \in \mathcal{P}^1_g, p^2 \in \mathcal{P}^2_g$ the guarantees $\psi^1_g(p^1), \psi^2_g(p^2)$ are nonempty sets.*

Then the guarantees (6.10) and (6.12) are satisfiable/nonempty.

Proof. Parameters that appear within the sequences (6.8) and (6.9) have a lower bound from the first condition and second conditions. Let these lower bounds be denoted as l^1

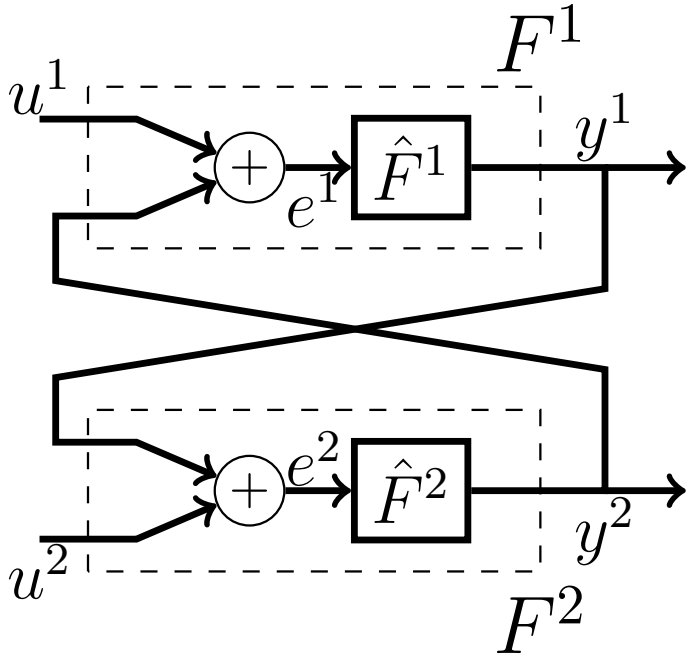


Figure 6.3: System interconnection for the classical small gain theorem. This interconnection is related to Figure 6.2 via composite systems F^1, F^2 which incorporate the addition blocks.

and l^2 respectively. The sets $\psi_g^1(l^1)$ and $\psi_g^2(l^2)$ are nonempty and $\psi_g^1(l^1) \subseteq \psi_g^1(p_g^1[k])$ and $\psi_g^2(l^2) \subseteq \psi_g^2(p_g^2[k])$ for all $k \in \mathbb{Z}_{\geq 0}$. Therefore, because $\psi_g^1(l^1) \subseteq \cap_{k\in\mathbb{Z}_{\geq 0}} \psi_g^1(p_g^1[k])$ and $\psi_g^2(l^2) \subseteq \cap_{k\in\mathbb{Z}_{\geq 0}} \psi_g^2(p_g^2[k])$, the guarantees (6.10) and (6.12) correspond to nonempty sets and are hence satisfiable. □

Classical Small Gain Theorem as a Special Case

Theorem 7 and Proposition 6 recover the traditional small-gain theorem [28]. Given some norm, let $\mathcal{L}$ be the associated space of norm bounded signals. A signal $x[\cdot]$ has an associated T-truncated norm $|x[\cdot]|_T = |\mathbb{I}_T x[\cdot]|$. The signal $x[\cdot]$ is pointwise multiplied with $\mathbb{I}_T$ the indicator function on the time interval $[0, T]$ before the signal norm is taken. The $\mathcal{L}$-extended space $\mathcal{L}_e$ is defined as $\{x[\cdot] : \forall T > 0,\ |x[\cdot]|_T < \infty\}$ and it is clear that $\mathcal{L}$ is a strict subset of $\mathcal{L}_e$.

Corollary 3. *(Classical Small Gain Theorem [28]) Let systems $\hat{F}^1, \hat{F}^2$ be input-output maps $\hat{F}^i : \mathcal{L}_e \mapsto \mathcal{L}_e$ and interconnected as in Figure 6.3. Let $e^1[\cdot], e^2[\cdot] \in \mathcal{L}_e$ and $u^1[\cdot], u^2[\cdot]$ be defined such that*

$$u^1[\cdot] = e^1[\cdot] - y^2[\cdot] \tag{6.13}$$
$$u^2[\cdot] = e^2[\cdot] - y^1[\cdot]. \tag{6.14}$$

Suppose there are four constants $\gamma^1, \gamma^2, \beta^1, \beta^2 \geq 0$ *such that*

$$|y^1[\cdot]|_T \leq \gamma^1 |e^1[\cdot]|_T + \beta^1 \tag{6.15}$$
$$|y^2[\cdot]|_T \leq \gamma^2 |e^2[\cdot]|_T + \beta^2 \tag{6.16}$$

for all T*. If* $\gamma^1\gamma^2 < 1$*, then for all* T*:*

$$|y^1[\cdot]|_T \leq \frac{1}{1-\gamma^1\gamma^2}\left(\gamma^1|u^1[\cdot]|_T + \gamma^1\gamma^2|u^2[\cdot]|_T + \gamma^1\beta^2 + \beta^1\right) \tag{6.17}$$
$$|y^2[\cdot]|_T \leq \frac{1}{1-\gamma^1\gamma^2}\left(\gamma^2|u^2[\cdot]|_T + \gamma^1\gamma^2|u^1[\cdot]|_T + \gamma^2\beta^1 + \beta^2\right). \tag{6.18}$$

Proof. The interconnection defined by (6.13) and (6.14) is depicted in Figure 6.3 where the dashed boxes correspond to F^1, F^2 in Figure 6.2 used in Theorem 7. Via the triangle inequality, the bounds (6.15) and (6.16) are replaced with

$$|y^1[\cdot]|_T \leq \gamma^1 |u^1[\cdot]|_T + \gamma^1 |y^2[\cdot]|_T + \beta^1 \tag{6.19}$$
$$|y^2[\cdot]|_T \leq \gamma^2 |u^2[\cdot]|_T + \gamma^2 |y^1[\cdot]|_T + \beta^2. \tag{6.20}$$

The assumption parameters associated with the exogenous inputs u^1, u^2 are bounds on their truncated norms. The feedback assumptions pertain to norm bounds on y^1, y^2. The parameter spaces are $\mathcal{P}^1_{ae}, \mathcal{P}^1_{af}, \mathcal{P}^1_g = \mathbb{R}_{\geq 0} \cup \{\infty\}$. For system F^1, define the exogenous assumption, feedback assumption, and guarantee as

$$\psi^1_{ae}(p) = (|u^1[\cdot]|_T \leq p) \tag{6.21}$$
$$\psi^1_{af}(r) = (|y^2[\cdot]|_T \leq r) \tag{6.22}$$
$$\psi^1_g(r) = (|y^1[\cdot]|_T \leq r) \tag{6.23}$$

with the parameter iteration map $\lambda^1(p, r) = \gamma^1 p + \gamma^1 r + \beta^1$. With the above definitions, the bounds (6.19) can be replaced with a parametric assume-guarantee contract. Analogous definitions for F^2 lead to a similar reformulation of (6.20). The first condition of Theorem 6 is therefore satisfied. The second condition is satisfied because both the guarantees and feedback assumptions are of the same form. The third and fourth conditions of Theorem 6 are satisfied because the existence of $e^1[\cdot], e^2[\cdot] \in \mathcal{L}_e$ implies that their T-truncated norm is finite for some T. Via (6.15) and (6.16), $y^1[\cdot], y^2[\cdot]$ also have finite T-truncated norm for an identical T. Via the triangle inequality, $u^1[\cdot], u^2[\cdot]$ must have finite T-truncated norm and satisfy (6.21).

For fixed norm bounds on $u^1[\cdot], u^2[\cdot]$, the feedback iteration functions become $\hat{\lambda}^1(r) := \gamma^1|u^1[\cdot]|_T + \gamma^1 r + \beta^1$ and $\hat{\lambda}^2(r) := \gamma^2|u^2[\cdot]|_T + \gamma^2 r + \beta^2$. When $\gamma^1\gamma^2 < 1$ the parameter iterations converge to a pair of fixed points, which are given by the right hand sides of (6.17) and (6.18). Theorem 7 certifies that these bounds are in fact enforced. We know these guarantees are satisfiable via Proposition 6 because any subset of $\mathcal{P}^1_g, \mathcal{P}^2_g = \mathbb{R}_{\geq 0} \cup \{\infty\}$ has a lower bound within $\mathbb{R}_{\geq 0} \cup \{\infty\}$ and the guarantees ψ^1_g, ψ^2_g are non-empty for all parameters. □

6.3 Hausdorff Continuity of Parametric Linear Temporal Logic

The results from the previous section place relatively mild conditions on guarantee specifications $\psi_g(\cdot)$ to provide a small gain result. These were satisfied when the parameteric specification corresponded to sublevel sets of a norm on signals. In this section, we consider a parametric variant of linear temporal logic (LTL) that satisfies the requirements of Theorem 7. Regular LTL is defined in Section 10.1 contained in the Appendix.

In our problem formulation, LTL formulas $\phi \subseteq \mathcal{U}^\omega \times \mathcal{Y}^\omega$ can be thought of as sets of infinite length input-output sequences. We consider a variant of LTL where input predicates are subsets of $\mathcal{U}$ of the form $f(x) \sim p$ where $\sim \in \{\leq, \geq\}$ and $f : \mathcal{U} \mapsto \mathbb{R}$ is a real-valued function. Output predicates are defined analogously. We call this variant parametric LTL. Definition 23 provides a syntax for these formulas.

Definition 23. *Parametric linear temporal logic formulas are constructed with the syntax below*

$$\phi = \top \mid f(\cdot) \sim p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi_1 \mathbf{U} \phi_2.$$

with parametric predicates $f(\cdot) \sim p$ where $p \in \mathbb{R}$ and $\sim \in \{\leq, \geq\}$.

The simplest parametric LTL formula is a parametric predicate, which takes the form of sublevel or superlevel set of a function. The parameter p corresponds to the level. Unfortunately, even for continuous functions $f(\cdot)$, the Hausdorff distance between sublevel sets does not vary continuously for all $p \in \mathbb{R}$. Consider the example given in Figure 6.4. Due to the presence of a spurious local minimum, perturbing p from $p = -1$ to $p = -1 - \epsilon$ for any $\epsilon > 0$ causes the point at zero to vanish from the sublevel set. The Hausdorff distance between the two sublevel sets is lower bounded in this example by $\sqrt{2}$ for all sufficiently small neighborhoods around $p = -1$.

To alleviate this issue of disconnected sublevel sets appearing with spurious local minima, we consider a fragment of LTL where the predicates are compact convex sets.

Definition 24. *LTL formulas with convex parametric predicates are constructed with the following syntax.*

$$\phi = \top \mid f(\cdot) \leq p \mid g(\cdot) \geq q \mid \phi_1 \vee \phi_2 \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi_1 \mathbf{U} \phi_2.$$

where for each predicate $f(\cdot) \leq p$ associated with a convex $f(\cdot)$ we restrict p to the domain $[\min_x f(x), \infty]$ and similarly $g(\cdot)$ is concave with $q \in [-\infty, \max_x g(x)]$.

We make the mild technical assumption that $f(\cdot)$ is uniformly continuous; that is, for all $\epsilon > 0$ there exists a δ where $d(x, y) < \delta$ implies $|f(x) - f(y)| < \epsilon$ for all appropriate x, y.

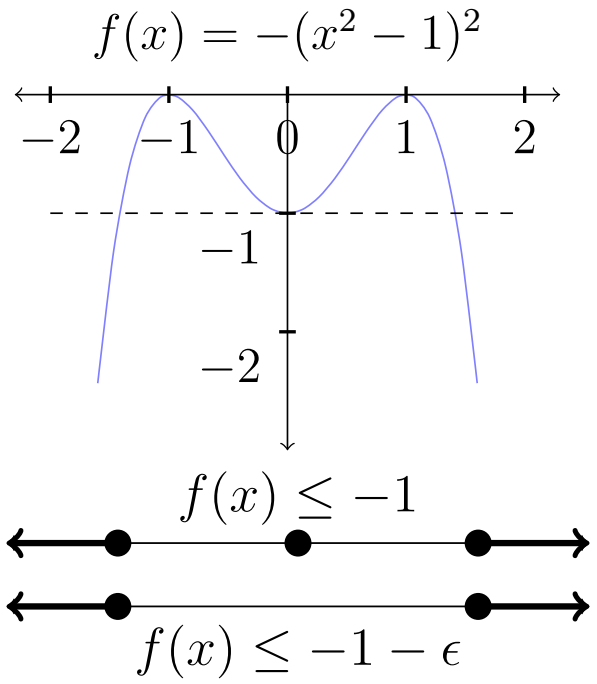


Figure 6.4: Parametric sublevel sets of non-convex continuous functions are not necessarily Hausdorff continuous.

Proposition 7. *The sublevel set of uniformly continuous convex predicates $f(\cdot) \leq p$ varies continuously with parameter $p \in [\min_x f(x), \infty]$ when the distance between predicates is given by the Hausdorff metric.*

Proof. Consider two predicates $\psi(p) = f(x) \leq p$ and $\psi(p') = f(x) \leq p'$. Without loss of generality, we assume that $p < p'$. Suppose $\epsilon > 0$. Let $m^+ \geq 0$ be defined as

$$m^+ = \left(\sup_{x \in \mathcal{B}_\epsilon(\psi(p))} f(x) \right) - p.$$

m^+ is the least upper bound on how much $f(\cdot)$ may increase by bloating the set $\psi(p)$ by ϵ. Due to uniform continuity of $f(\cdot)$, m^+ is finite. Because $f(\cdot)$ is convex its sublevel sets cannot consist of many disjoint regions and $\{x : f(x) \leq p + am^+\} \subseteq \mathcal{B}_\epsilon(\psi(p))$ for all $0 < a < 1$. If $|p - p'| < am^+$ then $\psi(p') \subseteq \mathcal{B}_\epsilon(\psi(p))$. An identical argument can be made when $p' < p$. Thus, if the parameters $|p - p'| < am^+$ then the Hausdorff distance of the sublevel sets are bounded above by ϵ. □

Note that the syntax in Definition 24 makes the curious choice of permitting disjunctions $\vee$ and not conjunctions $\wedge$. This choice was made due to the following property of the

Hausdorff distance

$$d_H(A \cup B, C \cup D) \leq \max(d_H(A, C), d_H(B, D)) \tag{6.24}$$

which upper bounds the distance between sets after a union. No analogous property exists for set intersections because they may be empty and the Hausdorff distance is ill defined. The potential loss of convexity under unions and disjunctions is not an issue because convexity of predicates in Definition 24 simply serves as a sufficient condition for predicates to be Hausdorff continuous and is not necessary.

Theorem 8. *Let $\psi(\cdot)$ be a parametric specification constructed with the convex predicate LTL grammar from Definition 24. Define $N \in \mathbb{Z}_{\geq 0}$ to be the number of times a predicate appears in $\psi(\cdot)$ and parameter space $\mathcal{P} = [\min_x f_1(x), \infty] \times \ldots \times [\min_x f_N(x), \infty]$. Specifications constructed with the grammar $\psi(\cdot)$ are Hausdorff continuous where signals distances are measured with a supremum metric, $d(x[\cdot], y[\cdot]) = \sup_{k \in \mathbb{Z}_{\geq 0}} d(x[k], y[k])$*

Proof. Suppose $\epsilon > 0$. Let m_i be defined as it appears at the end of the proof of Proposition 7 for predicate $f_i(p_i) \leq p_i$. Let $m = \min_{i \in \{1,\ldots,N\}} m_i$. Suppose that $x[\cdot] \models \psi(p)$ but $x[\cdot] \not\models \psi(p')$ and $\max_i(|p_i - p_i'|) < m$. Each predicate p_i in formula $\psi(p)$ has an associated infinite Boolean sequence where the k-th value is $\top$ if and only if $x[k] \models p_i$. For some time k, there must be at least one predicate that is different; for it to be otherwise would contradict the assertion that $x[\cdot] \not\models \psi(p')$. Given such a time step k, Proposition 7 and (6.24) guarantee that for any time step k when the difference arises, $x[k]$ must be less than a distance ϵ away from a point $y[k]$ that satisfies the same set of predicates for p'. Thus, $\psi(p) \subseteq \mathcal{B}_\epsilon(\psi(p'))$ where the $\epsilon-$expansion of $\psi(p')$ is with respect to the supremum metric. A similar argument can be made for the case when $x[\cdot] \models \psi(p')$ and $x[\cdot] \not\models \psi(p)$. □

Theorem 8 augments Theorem 7 by providing a concrete instantiation of a class of Hausdorff continuous specifications with temporal logic operators.

6.4 Certification of Parametric Contracts

To apply the results from previous sections we need to show that each system satisfies a parametric assume-guarantee contract. We pose a falsification problem that seeks to construct a violation of the contract. Consider a system F with a state space $\mathcal{X}$ and initial state set $\mathcal{X}_0$. The notation $y[\cdot] \in F(x[0], u[\cdot])$ signifies that output $y[\cdot]$ satisfies the dynamics F permitted by $x[0]$ and $u[\cdot]$. Let (ϕ_a, ϕ_g) be a parametric contract obtained from parametric specifications ψ_a, ψ_g and parameter map $\lambda : \mathcal{P}_a \to \mathcal{P}_g$.

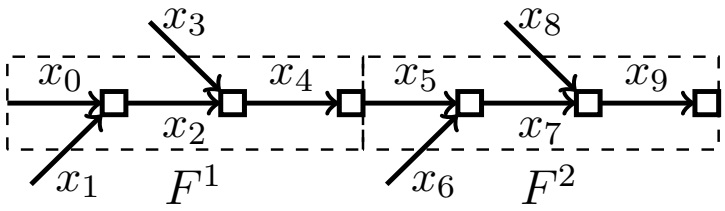


Figure 6.5: An example network with two on-ramps x_1, x_3. Dashed arrows are exogenous network links.

Problem 1. *If there exist p, $x[0]$, and $u[\cdot]$ that satisfy constraints (6.26), (6.27) and (6.28) then F does not satisfy the parametric assume-guarantee contract (ϕ_a, ϕ_g).*

$$\begin{aligned} \textit{find} \quad & p \in \mathcal{P}_a, x[0] \in \mathcal{X}_0, u[\cdot] && (6.25)\\ \textit{subject to} \quad & u[\cdot] \models \psi_a(p) && (6.26)\\ & y[\cdot] \not\models \psi_g(\lambda(p)) && (6.27)\\ & y[\cdot] \in F(x[0], u[\cdot]). && (6.28) \end{aligned}$$

The proper falsification engine to solve Problem 1 is implementation specific and depends on both the system dynamics and specification representation.

For black-box systems and systems exhibiting complex, hybrid, and non-linear dynamics, simulation-based falsification is the most practical method to certify that an assume-guarantee contract is satisfied. Most existing simulation-based falsification algorithms are sound but typically not complete. While the failure to falsify a contract does not imply that the contract holds, it is evidence suggesting that the contract holds. Simulation-based falsification tools are built into toolboxes `S-TaLiRo`[4] and `Breach`[29] for metric and signal temporal logic.

If the falsification algorithm is complete and no violating p, $x[0]$, and $u[\cdot]$ exist, then F satisfies (ϕ_a, ϕ_g). The examples in the next section use a component of the `BluSTL` toolbox [75] to translate bounded time temporal logic specifications (6.26) and (6.27) into mixed integer constraints for the optimization toolbox `YALMIP` [57].

6.5 Freeway Interconnection Example

This section applies Theorems 6 and 7 to a freeway traffic example. Consider the two freeway segments depicted in Fig. 6.5, where the left segment has a main stretch of three links x_0, x_2, x_4 and two on-ramps x_1, x_3. The right segment has identical dynamics. We use the cell transmission model (CTM) a macroscopic fluid-like model of freeway dynamics developed by Daganzo [24]. Individual vehicles are not a component of this model. Each discrete time instant represents a five minute interval.

Freeway Dynamics

We first describe the dynamics for F^1, which are identical to the dynamics of F^2 besides a variable renaming, and subsequently describe how the interconnected networks resemble the small-gain interconnections (e.g. Figure 6.2) from previous sections.

Freeway segment F^1's state space $\mathcal{X}^1 \subset {\mathbb{R}_{\geq 0}}^5$ represents the average occupancy over the five minute period in each of the five links. We overload notation and refer to links and their occupancy values using the same variable. The upper bound on occupancy is encoded with a vector $x^{\max} = [40, 20, 40, 20, 40]$. The state update equations arise from conservation of mass:

$$\begin{aligned}
x_0[k+1] &= x_0[k] - f_0^{\text{out}}[k] + f_0^{\text{in}}[k] \\
x_1[k+1] &= x_1[k] - f_1^{\text{out}}[k] + f_1^{\text{in}}[k] \\
x_2[k+1] &= x_2[k] - f_2^{\text{out}}[k] + f_0^{\text{out}}[k] + f_1^{\text{out}}[k] \\
x_3[k+1] &= x_3[k] - f_3^{\text{out}}[k] + f_3^{\text{in}}[k] \\
x_4[k+1] &= x_4[k] - f_4^{\text{out}}[k] + f_2^{\text{out}}[k] + f_3^{\text{out}}[k]
\end{aligned}$$

where $f_i^{\text{out}}[k]$ and $f_i^{\text{out}}[k]$ respectively represent the flows exiting and entering link x_i at time k.

The flows into and out of a link are determined by *demand* and *supply*. A link's demand is the rate at which it would like to send vehicles to downstream links. The demand $d_i(x_i[k])$ that link x_i exhibits is a non-decreasing function

$$d_i(x_i[k]) = \min(c_i, x_i[k]) \tag{6.29}$$

where c_i is a saturation rate.The primary links have saturation rates $c_0 = c_2 = c_4 = 10$ and on-ramps have saturation rates $c_1 = c_3 = 5$. All links also exhibit a supply function

$$s(x_i[k]) = x_i^{\max} - x_i[k], \tag{6.30}$$

which is the rate of incoming vehicles that it can accept from upstream. A link's supply is partitioned among upstream links, with links x_2, x_4 allocating 80% of their supply to an upstream highway link and 20% to on-ramps. Link x_2's supply and demand functions are depicted in Figure 6.6. Congestion occurs when demand exceeds supply and the left term in the minimization is active. The flows out of links $0, 1, 2, 3$ are the minimum between the supply available to them and their demand:

$$\begin{aligned}
f_0^{\text{out}}[k] &= \min\left(.8(40 - x_2[k]), 10, x_0[k]\right) \\
f_1^{\text{out}}[k] &= \min\left(.2(40 - x_2[k]), 5, x_1[k]\right) \\
f_2^{\text{out}}[k] &= \min\left(.8(40 - x_4[k]), 10, x_2[k]\right) \\
f_3^{\text{out}}[k] &= \min\left(.2(40 - x_4[k]), 5, x_3[k]\right)
\end{aligned}$$

Interconnection between Networks

Figure 6.7 summarizes the input and output variables for each network. Network F^1 has a vector of demands as its exogenous input $u^1_{ae} = (d^{\text{exog}}, d^{\text{on}}_1, d^{\text{on}}_3)$ and the feedback input $u^1_{af} = (s_5)$ is the supply from downstream. Similarly, F^2 has exogenous input $u^2_{ae} = (s^{\text{exog}}, d^{\text{on}}_6, d^{\text{on}}_8)$ and feedback input $u^2_{ae} = (d_4)$. The outputs can be identified in a similar manner.

With the notions of demand and supply in mind, we can now consider how both networks are affected by their interconnection and by exogenous environments. The flow f^{out}_4 between F^1 and F^2 is determined by d_4 and s_5

$$f^{\text{out}}_4[k] = \min\left(.8(40 - x_5[k]), 10, x_4[k]\right).$$

Both systems experience an exogenous environment via the on ramp demands. The upstream system F^1 also experiences a demand d^{exog} for link x_0 and the downstream network F^2 experiences an exogenous supply for link x_9.

Link x_0 allocates 80% of its supply to the exogenous environment. The flow into x_0 is therefore

$$f^{\text{in}}_0[k] = \min(.8(40 - x_0[k]), d^{\text{exog}}).$$

The onramps x_i with $i \in \{1, 3, 6, 8\}$ allocate all supply to the environment so

$$f^{\text{in}}_i[k] = \min((20 - x_i[k]), d^{\text{exog}}_i).$$

Similarly, link x_9's outflow is governed by an exogenous environment so

$$f^{\text{out}}_9[k] = \min(.8s^{\text{exog}}, 10, x_9[k]).$$

Certifying Intermittent Congestion

Congestion is shown to be intermittent after the two segments are interconnected. Intermittency is encoded via "always" and "eventually" temporal operators augmented with intervals

$$\mathbf{G}_{[0,3]}\phi := \phi \wedge \mathbf{X}\phi \wedge \mathbf{XX}\phi \wedge \mathbf{XXX}\phi \tag{6.31}$$

$$\mathbf{F}_{[0,2]}\phi := \phi \vee \mathbf{X}\phi \vee \mathbf{XX}\phi. \tag{6.32}$$

For both systems, all onramp demands are limited to always be less than 3. That is,

$$\mathbf{G}(d^{\text{on}}_i \leq 3) \tag{6.33}$$

for all $i \in \{1, 3, 6, 8\}$. All links are assumed to have an initial occupancy less than 5, i.e., $\mathcal{X}_0 = \prod_{i=0,\ldots,4}[0, 5] \subset \mathcal{X}$.

From Figure 6.7, it's clear that the upstream network F^1 is subjected to an exogenous mainline demand d^{exog} and the supply availability from downstream network F^2. A static

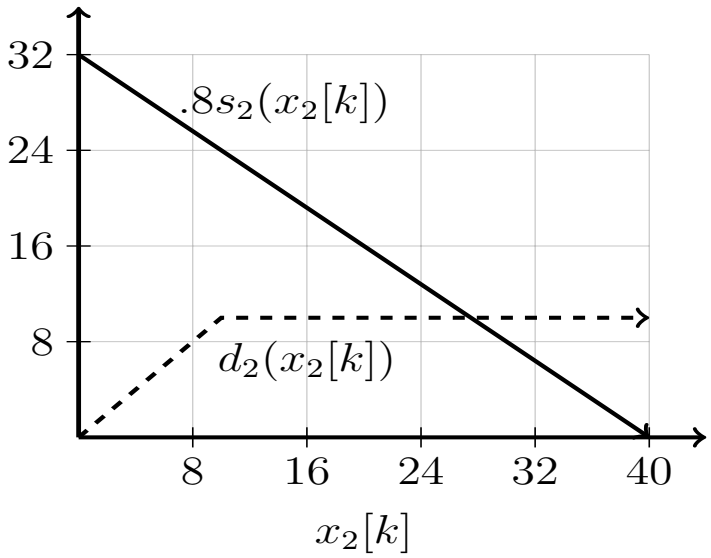


Figure 6.6: Supply (solid) that link x_2 provides to link x_0 and Demand (dashed) that link x_2 creates for link x_4

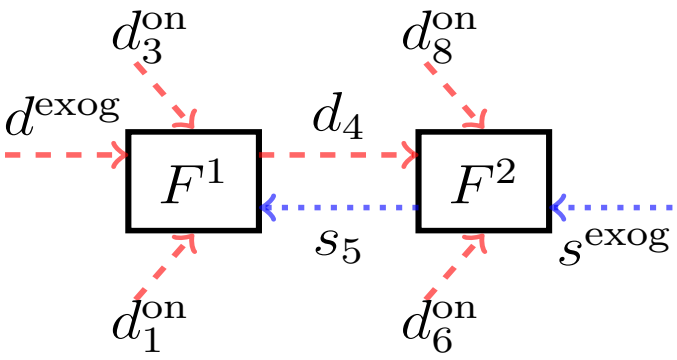


Figure 6.7: Although vehicular flow in Figure 6.5 is from left to right, the right network also affects the left network. Demand's influence (dashed lines) is directed forward while supply's influence (dotted lines) is directed backward.

exogenous environment contract is imposed by assuming that the main line demand satisfies the assumption with no free parameters

$$\mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d^{\text{exog}} \leq 15). \tag{6.34}$$

How does the supply from F^2 affect the demand outputted by F^1? Via monotonicity of the network dynamics, a greater supply availability means that F^1 can expel vehicles quicker and will be able to lower the demand it outputs. This relationship is encoded in the parametric assume-guarantee contract below:

$$\phi_a^1 := \bigvee_{s\geq 0} \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(s_5(x_5) \geq 10 - s) \tag{6.35}$$

$$\phi_g^1 := \bigwedge_{s\geq 0} \Big(\mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(s_5(x_5) \geq 10 - s) \tag{6.36}$$

$$\Rightarrow \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d_4(x_4) \leq \lambda_1(s))\Big) \tag{6.37}$$

$$\lambda_1(s) := .9s + 4. \tag{6.38}$$

The falsification procedure encoded in Problem 1 failed to violate the assume-guarantee contract for any parameter $s \geq 0$ and hence F^1 satisfies the parametric contract (ϕ_a^1, ϕ_g^1).

Similarly the downstream network F^2 is subjected to the demand from x_4 and exogenous supply. The exogenous supply has a fixed assumption

$$\mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(s^{\text{exog}} \geq 5). \tag{6.39}$$

It influences F_1 by outputting supply from x_5 and the contract is

$$\phi_a^2 := \bigvee_{d\geq 0} \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d_4(x_4) \leq d) \tag{6.40}$$

$$\phi_g^2 := \bigwedge_{d\geq 0} \Big(\mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d_4(x_4) \leq d) \tag{6.41}$$

$$\Rightarrow \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(s_5(x_5) \geq 10 - \lambda_2(d))\Big) \tag{6.42}$$

$$\lambda_2(d) := .2d. \tag{6.43}$$

Again, Problem 1 failed to violate the contract and F^2 therefore satisfies the contract (ϕ_a^2, ϕ_g^2).

Each of the conditions for Theorem 6 have been proven to hold in this section.

1. The parametric contracts are satisfied for each network.
2. Guarantees from one network imply the feedback assumptions of the other network because they are of the same form. In other words, pairs (6.40), (6.37) and (6.35), (6.42) are identical parametric specifications.

3. The exogenous assumptions are satisfied via (6.33), (6.34), and (6.39).

4. Let $i = 2$. For a large enough $d \geq 0$, the feedback assumption $\psi^2_{af}(d) = \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d_4(x_4) \leq d)$ is satisfied because d_4 has a maximum value of 10.

The composition of the parameter mapping functions λ_1, λ_2 is a contraction and hence converges in the limit to a fixed point $(d, s) = (4.878, .975)$. Thus, via Theorem 7 the following statement must also hold:

$$\mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(d_4(x_4) \leq 4.878) \wedge \mathbf{G}_{[0,3]}\mathbf{F}_{[0,2]}(s_5(x_5) \geq 9.025).$$

Chapter 7

Dynamic Contracts for Distributed Temporal Logic Control Synthesis

In the context of formal control synthesis, contracts have mainly come in the form of controlled invariant sets [64, 83, 49, 82]. An individual subsystem treats the unknown coupling from neighboring subsystems as disturbances and computes a control strategy that is robust to that uncertainty. The invariant sets bound that uncertainty and satisfaction of a contract is enforced as constraints in each sub-network's MPC problem. These invariant sets are typically computed offline and ignore real-time conditions, giving rise to unnecessary conservatism.

We propose a method to dynamically change the contracts to attain a better optimum based on real-time conditions. Each individual sub-network must first be "mined" for contracts over a range of demand severity levels and different local scenarios. The mined contracts are merged into a finite state machine that serves as a high-level coordinator. Each graph node corresponds with the contract constraints imposed on each individual sub-network. Unsafe regions of the graph are sets of scenarios where sub-networks' promises to each other are inconsistent. Edges in the coordinator graph are permitted only when a smooth hand-off between contracts is ensured and recursive feasibility is maintained. Existence of some edges can be guaranteed offline, while additional edges may occasionally be added at runtime. Our protocol accommodates a rich class of temporal specifications that include safety, reachability, and recurrence properties.

Just as Chapter 6 showcased how parametric contracts permits one to have tighter guarantees for verification, this chapter shows how dynamic assume-guarantee contracts can provide a richer class of guarantees for control synthesis.

7.1 Preliminaries

A discrete time interval $I = [a, b)$ is a contiguous subset of $\mathbb{N}$ where $a, b \in \mathbb{N} \cup \{\infty\}$, $a \leq b$, and $b \notin [a, b)$. A closed discrete interval is given by $[a, b] = [a, b) \cup [b]$. Let $x[a, b)$ and $x[a, b]$

represent a slice of the signal $x[0, \infty)$ along the intervals $[a, b)$ and $[a, b]$ respectively.

Network Dynamics

We adopt the model from Sadraddini et al. [82]. A traffic network $\mathcal{N}$ can be represented as a graph with a set of links $\mathcal{L}$ representing roads and nodes $\mathcal{S}$ representing intersections. Each link l has an associated vehicle occupancy at time $t \in \mathbb{N}$ denoted by $x^l[t] \in [0, c^l]$ with a maximum capacity $c^l \geq 0$. The entire network state space is $\mathcal{X} = \prod_{l \in \mathcal{L}} [0, c^l]$.

Link l's set of upstream links is $\mathcal{L}^l_{\text{up}}$ and its set of downstream links is $\mathcal{L}^l_{\text{down}}$. Vehicles always flow from an upstream link to a downstream link. The maximum flow out of link l is $q^l \in [0, c^l]$. Vehicle flow is controlled via traffic lights and ramp meters. Green and red lights at intersections are represented via a control set $\mathcal{U}^l \in \{0, q^l\}$, with $u^l = 0$ indicating that no vehicles may exit link l (red light). If l is actuated by a ramp meter, then $\mathcal{U}^l \in [0, q^l]$ is a continuous variable. If a link is uncontrolled, we have $\mathcal{U}^l = \{q^l\}$. The control input space is denoted by $\mathcal{U} = \prod_{l \in \mathcal{L}} \mathcal{U}^l$. A non-ordered pair of links (l, k) is *antagonistic* if $u^l > 0 \Rightarrow u^k = 0$. The set of all antagonistic pairs is denoted by $\mathcal{A} \subset \mathcal{L} \times \mathcal{L}$.

Flow allocation is determined by functions specifying turning ratios $\beta : \mathcal{L} \times \mathcal{L} \mapsto [0, 1]$ and supply ratios $\alpha : \mathcal{L} \times \mathcal{L} \mapsto [0, 1]$. The turning ratio $\beta(l, m)$ represents the proportion of vehicles exiting l and entering m, is nonzero only for $m \in \mathcal{L}^l_{\text{down}}$ and the ratio sum cannot exceed one, i.e. $\sum_{m \in \mathcal{L}^l_{\text{down}}} \beta(l, m) \leq 1$. The supply ratio $\alpha(k, l)$ represents the proportion of free space in l allocated to upstream link k, and is nonzero only for $k \in \mathcal{L}^l_{\text{up}}$.

A link l's output y^l represents the total vehicles it would like to send and is given by:

$$y^l[t] = \min(x^l[t], q^l, u^l[t]), \tag{7.1}$$

where the first two terms in the minimization represent the number of vehicles that want to exit link l and the third term controls the output demand. The flow exiting link l is

$$f^l[t] = \min\left(y^l[t], \frac{\alpha(k, l)}{\beta(k, l)}(c^k - x^k[t])\right), \tag{7.2}$$

where the second term limits flow due to lack of supply downstream. The state update equation is driven by conservation of mass:

$$\begin{aligned} x^l[t+1] = \quad & \min\big(c^l, x^l[t] + \textstyle\sum_{m \in \mathcal{L}^l_{\text{up}}} \beta(m, l) f^m[t] \\ & - f^l[t] + d^l[t]\big), \end{aligned} \tag{7.3}$$

where $d^l[t]$ is the exogenous demand entering link l at time t. We compactly represent the dynamics above with

$$x[t+1] = F(x[t], u[t], d[t]) \tag{7.4}$$

$$y[t] = g(x[t], u[t]). \tag{7.5}$$

The congestion free region of a network is defined as the set $\psi \subseteq \mathcal{X} \times \mathcal{U}$ where for all $l \in \mathcal{L}$ the second term is not a unique minimizer in eq. (7.2) and c^l is not a unique minimizer of eq. (7.3). The freeflow condition is a local condition; that is, link l's membership in the freeflow region is determined only by adjacent links. Given the joint space $\mathcal{X} \times \mathcal{U}$ we define a partial order $\preceq_{\mathcal{X}\times\mathcal{U}}$. Two points $(x, u), (x', u')$ in this space satisfy $(x, u) \preceq_{\mathcal{X}\times\mathcal{U}} (x', u')$ if and only if $x \leq x'$ and $u \leq u'$ coordinate-wise. Likewise, the disturbance space $\mathcal{D}$ is equipped with a partial order $\preceq_{\mathcal{D}}$.

Definition 25. *A subset $\mathcal{K} \subseteq \mathcal{P}$ is a lower set if for all $q, r \in \mathcal{P}$, $r \in \mathcal{K}$ and $q \preceq_{\mathcal{P}} r$ imply $q \in \mathcal{K}$.*

Definition 26. *Monotone Dynamics The dynamics $F(x, u, d)$ is monotone with respect to the partial orders $\preceq_{\mathcal{X}\times\mathcal{U}}$ and $\preceq_{\mathcal{D}}$ if and only if*

$$(x, u) \preceq_{\mathcal{X}\times\mathcal{U}} (x', u') \wedge d \preceq_{\mathcal{D}} d' \Rightarrow F(x, u, d) \preceq_{\mathcal{X}} F(x', u', d'). \tag{7.6}$$

The congestion free region ψ is a lower set of $\mathcal{X} \times \mathcal{U}$ and exhibits monotone system dynamics [82].

Metric Temporal Logic

To encode desirable discrete time behaviors of the traffic network, we use metric temporal logic (MTL) as defined in Section 10.1 in the Appendix. These specifications will be encoded as constraints for a model predictive control (MPC) solver. We introduce the notion of a specification's horizon because the MPC solvers can only reason over a finite time horizon. A specification's horizon $h(\phi)$ is the minimal length of a suffix required to determine satisfaction of ϕ. For example, $h(\mathbf{G}_{[0,3)}\psi) = 3$ if ψ is a predicate. The satisfaction of ϕ by signal $s[\cdot]$ at time t is decided by $s[t, t + h(\phi))$; satisfaction at time t is independent of $s[t']$ for $t' \notin [t, t + h(\phi))$.

Lower Specifications

Lower specifications from Kim et al [50] are an instance of a lower set over a signal domain equipped with a partial order. If $s[k] \preceq_{\mathcal{X}} r[k]$ for all k in an appropriate time interval, then $s[\cdot] \preceq_{\mathcal{X}[\cdot]} r[\cdot]$.

Definition 27. *Lower Specifications A specification $\phi \subseteq \mathcal{X}[\cdot]$ is a lower specification if for all signals $s[\cdot], r[\cdot] \in \mathcal{X}[\cdot]$, conditions $s[\cdot] \models \phi$ and $r[\cdot] \preceq_{\mathcal{X}[\cdot]} s[\cdot]$ imply $r[\cdot] \models \phi$.*

All of the specifications in this chapter are lower specifications. Specifications that adhere to the following grammar are guaranteed to be lower specifications

Definition 28 (Lower Specification Grammar [50]). *Any specification ϕ^l constructed from the following grammar is a lower specification.*

$$\phi^l = \top \mid \mu^l \mid \neg\phi_u \mid \phi_1^l \wedge \phi_2^l \mid \phi_1^l \mathbf{U}_{[a,b)} \phi_2^l \tag{7.7}$$
$$\phi^u = \top \mid \mu^u \mid \neg\phi_l \mid \phi_1^u \wedge \phi_2^u \mid \phi_1^u \mathbf{U}_{[a,b)} \phi_2^u \tag{7.8}$$

where μ^l (respectively μ^u) is an atomic predicate that corresponds to a lower (respectively upper) set.

7.2 Problem Statement and Approach

Problem Formulation

We partition a traffic network into N smaller sub-networks $\mathcal{N}^i, i = 1, \cdots, N$. A guideline for partitioning traffic networks while taking into account formal specifications was introduced by Sadraddini et al. [82]. We consider traffic specifications of the following form.

$$\phi := \mathbf{G}_{[0,\infty)} \bigwedge_{i=1}^{N} \left(\psi^i \wedge \mathbf{F}_{[0,\infty)} \phi^i \right), \tag{7.9}$$

where ψ^i is the congestion-free region of network $\mathcal{N}^i$, and ϕ^i is a bounded-time MTL formula describing internal requirements of network $\mathcal{N}^i$.

Problem 2 (Control Synthesis). *Given a network $\mathcal{N}$, an initial condition $x[0]$ and a specification ϕ in the form* (7.9)*, find a control strategy such that ϕ is satisfied.*

An optimality criterion is added in Section 7.3.

Motivating Example

Consider a network $\mathcal{N}$ depicted in fig. 7.1 which consists of sub-networks $\mathcal{N}^i, i = 1, \cdots, 4$. Sub-network $\mathcal{N}^4$ represents a high capacity freeway while the others are urban areas. All sub-networks are interconnected via on(off)-ramps or urban roads. At each urban intersection, 50% of the vehicles proceed straight, 20% turn left, and 30% turns into an un-modeled external environment (e.g., parking). We have $q^l = 15, c^l = 40$ for urban roads, $q^l = 60, c^l = 25$ for freeways and $q^l = 30, c^l = 15$ for ramps. For all entry links to the network, let $d^l[t] \in [0, \frac{1}{4}q^l], t \in \mathbb{N}$ [1].

We aim for multiple control objectives. First, the network must remain in congestion-free region ψ at all times. Second, at each urban intersection traffic lights signalizing vertical and horizontal flows become simultaneously red infinitely often, allowing pedestrians to pass

[1] The full, detailed, parameter valuations of the network, code and simulation results of this paper are publicly available at `https://github.com/ericskim/cdc17dynamiccontracts`.

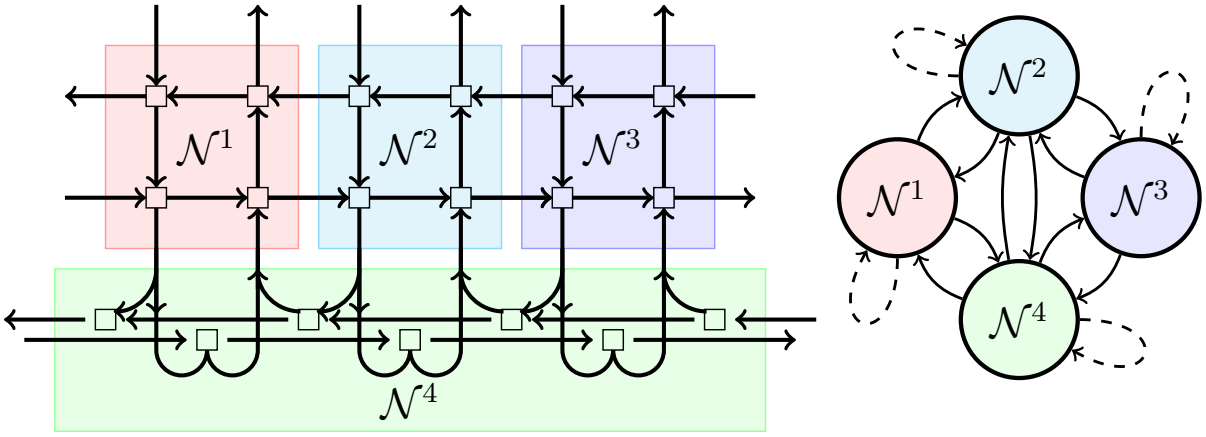


Figure 7.1: Example network $\mathcal{N}$ partitioned into four sub-networks $\mathcal{N}^i, i = 1, \cdots, 4$. Contract obligations from section 7.3 are depicted on the right. Solid arrows indicate an obligation from one network to an adjacent sub-network to limit incoming vehicular flow. Dashed arrows represent recursive feasibility obligations from a sub-network to its future self.

through the intersection in any direction and making the congestion-free specification harder to accomplish.

$$\phi = \mathbf{G}_{[0,\infty)}\Big(\bigwedge_{(l,k)\in\mathcal{A}} \psi \wedge \mathbf{F}_{[0,\infty)}(u_l = 0 \wedge u_k = 0)\Big). \tag{7.10}$$

This specification is decomposable into the form in (7.9).

7.3 Contract-Based Model Predictive Control

We adopt a distributed model predictive control (MPC) approach to solve the controller synthesis problem with temporal logic constraints, as solving the MPC problem for the entire network $\mathcal{N}$ in real time is computationally intractable. A distributed controller synthesis approach is enabled by having each sub-network *concurrently* compute an optimal control trajectory independently. Each sub-network is unaware of adjacent network states during this computation, yet they interact via sending and receiving vehicles to and from adjacent networks. Coordination is enforced by introducing assume-guarantee contracts between networks into each sub-network's MPC problem, ensuring that a sub-network does not inadvertently violate an adjacent sub-network's assumptions and that the distributed control synthesis procedure is sound.

Optimization Objective and Constraints

The MPC algorithm executes at each time step t, and aims to minimize the total delay over a horizon T:

$$J = \sum_{\tau=t}^{T+t-1} \sum_{l \in \mathcal{L}} \left(x^l[\tau] - f^l[\tau] \right) \tag{7.11}$$

by computing a control sequence $u[t, t+T)$ given the current state $x[t]$. Above, $x^l[\tau] - f^l[\tau]$ is the number of vehicles that are forced to remain in l at time τ. Cost (7.11) is a monotone function with respect to state if the evolution is restricted to the congestion-free region [82].

Both the network dynamics and MTL specification ϕ^i can be encoded as mixed integer constraints. The piecewise affine dynamics are encoded using the scheme by Mayne and Raković [59] where integer variables are used to detect membership within a set of state space polytopes. Boolean variables that capture predicate satisfaction at different moments in time [91, 74] allow one to encode satisfaction of bounded specification ϕ^i with a finite number of mixed integer constraints, given a sufficiently long MPC horizon.

Assumption 1. *Each sub-network $\mathcal{N}^i$ has a common MPC horizon T that is greater than the longest specification horizon. That is, $T > \max_i h(\phi^i)$.*

Specification ϕ requires that ϕ^i be satisfied infinitely often. Unlike the optimization procedure which executes with a receding horizon, the constraint ϕ^i can be enforced periodically at times kT where $k \in \mathbb{N}$. Periodic enforcement of ϕ^i for every kT translates to sets of constraints over disjoint intervals $[kT, kT + h(\phi^i))$ each associated with a $k \in \mathbb{N}$. A MPC iteration executing at kT will plan a trajectory over $[k, (k+1)T]$, $k \in \mathbb{N}$ that satisfies ϕ^i. A subsequent MPC iteration executing within that time interval $t \in [kT+1, (k+1)T)$ takes into account a state trajectory over $[kT, t]$ and input trajectories over $[kT, t)$ and can maintain satisfaction of ϕ^i at time kT simply by either i) Executing the input trajectory proposed by the iteration at time kT ii) Computing at time t an input trajectory with lower cost that still satisfies ϕ^i at time kT. While the first option is sufficient to enforce the specification, computing a lower cost trajectory at time t permits the MPC controller to react to incoming vehicles from adjacent sub-networks.

Interconnections and Contracts

The distributed MPC scheme contains a circular dependency because each sub-network is unaware of neighboring networks' planned actions. Each sub-network needs to promise neighboring sub-networks that they will satisfy each other's assumptions, but the feasibility of such a promise depends on the actions of one's neighbors. Assume-guarantee contracts are MPC constraints that break this dependency.

Definition 29 (Assume-Guarantee Contract (for distributed traffic control))**.** *Network $\mathcal{N}^i$'s assume-guarantee contract $\mathcal{C}^i$ along time interval $[kT, (k+1)T]$ consists of*

- *Assumption $\phi_a^i(x_*^i[kT], d_*^i[kT, (k+1)T))$ on the incoming demand and vehicles initially in the network:*

$$\bigwedge_{l\in\mathcal{L}^i}\big(x^l[kT]\le x^l_*[kT]\big) \tag{7.12}$$

$$\wedge\bigwedge_{t=kT}^{(k+1)T-1}\Big(\bigwedge_{l\in\mathcal{L}^i_{in}}\big(d^l[t]\le d^l_*[t]\big)\Big) \tag{7.13}$$

- *Guarantee* $\phi^i_g(x^i_*[kT+1,(k+1)T],y^i_*[kT+1,(k+1)T])$ *on the terminal state and output trajectory:*

$$\bigwedge_{l\in\mathcal{L}^i}\big(x^l[(k+1)T]\le x^l_*[(k+1)T]\big) \tag{7.14}$$

$$\wedge\bigwedge_{t=kT}^{(k+1)T}\Big(\bigwedge_{l\in\mathcal{L}^i_{out}}\big(y^l[t]\le y^l_*[t]\big)\Big) \tag{7.15}$$

The contract $\mathcal{C}^i$ is characterized by a set of parameters: the initial state $x_*[kT]$, external demand $d_*[kT,(k+1)T)$, terminal state $x_*[(k+1)T]$, and output trajectory $y_*[kT,(k+1)T])$ over output links. Sub-network $\mathcal{N}^i$'s assumption component states conditions over local and incoming links $\mathcal{L}^i$ and $\mathcal{L}^i_{\text{in}}$. The output guarantee is viewed as a signal $y_*[kT+1,(k+1)T])$ that upper bounds output trajectories of $y[kT+1,(k+1)T])$ on links in $\mathcal{L}^i_{\text{out}}$.

Definition 30 (Contract Satisfaction)**.** *A sub-network satisfies an assume-guarantee contract at time kT if for all $x[kT]$ and $d[kT,(k+1)T)$ satisfying eq.* (7.12) *and eq.* (7.13) *respectively, a control sequence $u[kT,(k+1)T)$ exists such that $\mathcal{N}^i$ remains in the local freeflow region ψ^i, and both the guarantee ϕ^i_g and MTL requirement ϕ^i are satisfied at kT.*

Contract satisfaction for *all* assumption satisfying scenarios $x[kT]$ and $d[kT,(k+1)T)$ is difficult to certify for general non-linear dynamics. However, within the congestion free region ψ the dynamics exhibit a monotonicity property, where a partial ordering with respect to state trajectories is preserved, and the initial state $x_*[kT]$ and demand $d^i_*[kT,(k+1)T)$ jointly yield the most adversarial environment. An environment that satisfies ϕ^i_a cannot violate the guarantee if $x_*[kT]$ and $d^i_*[kT,(k+1)T)$ do not violate the assumption [50]. Thus, synthesizing a satisfying control sequence $u^i_*[kT,(k+1)T)$ for environmental scenario $x_*[kT]$ and $d_*[kT,(k+1)T)$ such that the system satisfies the guarantees and remains congestion free also ensures that the control sequence will be satisfactory under more benign scenarios [81]. A set of contracts is consistent if all sub-network assumptions are implied by the guarantees.

Definition 31 (Assume-Guarantee Parameter Consistency)**.** *A set of contracts $\mathcal{C}^1,\dots,\mathcal{C}^4$ are consistent if if for all $\mathcal{N}^i$, input links $l\in\mathcal{L}^i_{in}$ and times $t\in[kT,(k+1)T)$*

$$\sum_{k\in\mathcal{L}^l_{up}}\beta(l,k)y^k_*[t]\le d^l_*[t]. \tag{7.16}$$

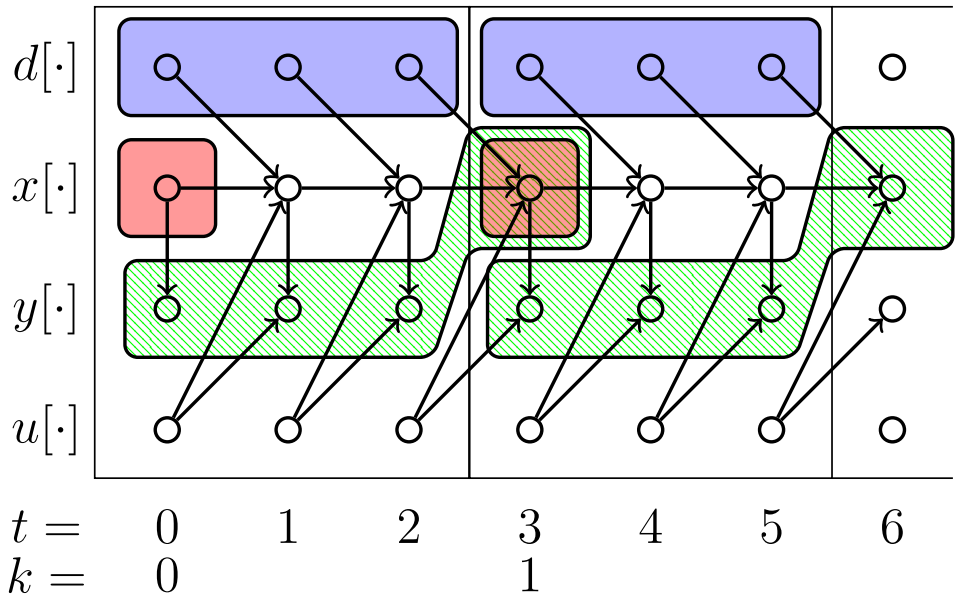


Figure 7.2: Three rows show different signals of the MPC algorithm over two MPC horizons where $T = 3$. Arrow indicate the flow of dependencies amongst signals over time. The shaded regions correspond to a network's assumptions about its initial state (red) and incoming demand (blue). The green patterned region is the guarantee over an output trajectory (recall that output y is a function of state and input) and a terminal state. At time $t = 3$, the assume-guarantee contract is reset, and the overlap at $x[3]$ between the initial state assumption and the state trajectory guaratee ensures that recursive feasibility is maintained.

Recursive Feasibility with Fixed Contracts

Recursive feasibility can be viewed as a network making a promise to its future self that all future constraints will remain feasible. Recursive feasibility has two components, corresponding to the specification ϕ_g^i constraint and the contract constraint eq. (7.15). Feasibility of the ϕ_g^i constraint at the kT-th time step has already been established via the initial state condition eq. (7.12).

Let contract $\mathcal{C}^i$ is be periodically every T steps with fixed parameters. Consider two different MPC executions at times kT and $(k+1)T$. The guarantee constraint ϕ_g^i along the interval $[(k+1)T, (k+2)T)$ is feasible when eq. (7.12) is satisfied at time $(k+1)T$. The MPC algorithm executing at time kT imposes the terminal state guarantee $x[(k+1)T] \leq x_*[(k+1)T]$, which implies that the initial state assumption at $(k+1)T$ with identical contract $\mathcal{C}^i$ is satisfied if:

$$\bigwedge_{l \in \mathcal{L}^i} x_*^l[(k+1)T] \leq x_*^l[(k+1)T]. \tag{7.17}$$

If $\mathcal{C}^i$ satisfies eq. (7.17) then it is said to be a recursively feasible contract. The final MPC problem for each sub-network requires enforcing the following constraints.

Problem 3 (Distributed MPC)**.** *Under the assumption that input demand satisfies eq.* (7.13)*, each sub-network* $\mathcal{N}^i$ *computes a local control sequence* $u[kT,(k+1)T)$*:*

$$\begin{aligned} \underset{u[kT,(k+1)T)}{argmin} \quad & \sum_{t=kT}^{(k+1)T} \sum_{l\in\mathcal{L}} \left(x^l[t] - f^l[t]\right) \\ s.t. \quad & (x[kT,(k+1)T], u[kT,(k+1)T)) \models \phi^i \\ & \textit{Guarantee eq. (7.15) to adjacent networks} \\ & \textit{Terminal State eq. (7.14)} \\ & \textit{Dynamics constraint eq. (7.4)} \end{aligned}$$

Assumption ϕ_a^i *is encoded in the constraint eq.* (7.4)*.*

Proposition 8 (Infinite Horizon Spec. Satisfaction)**.** *If each network* $\mathcal{N}^i$ *satisfies its assume-guarantee contract, each assume-guarantee contract* $\mathcal{C}^i$ *is recursively feasible, and the global initial state* $x[0]$ *satisfies each initial state eq.* (7.12) *assumption, then the distributed MPC algorithm satisfies the global specification eq.* (7.9)*.*

7.4 Dynamic Contracts

Definition 29 introduced contracts that are uniquely parametrized by $x_*[kT]$, $d_*[kT,(k+1)T)$, $x_*[(k+1)T]$, and $y_*[kT,(k+1)T)$, which do not change over many MPC horizons. Fixed contract parameters may lead to conservative guarantees if the network experiences less demand than expected and $d^l[kT,(k+1)T) \ll d_*^l[kT,(k+1)T)$ elementwise in eq. (7.13) because conservative assumptions prevent aggressive responses to benign real-time conditions. Likewise when $x_*^l[\tau]$ in eq. (7.15) is small, the guarantee is too optimistic and perhaps infeasible if a network experiences a sudden influx of vehicles.

Dynamic contracts allow parameters to switch at runtime. This enables the network to react to real-time conditions and increase the space of feasible contracts.

In general, a sub-network $\mathcal{N}^i$ can satisfy a collection of m^i assume-guarantee contracts,

$$\mathcal{P}^i = \{\mathcal{C}^i(p_1^i), \ldots, \mathcal{C}^i(p_{m_i}^i)\}. \tag{7.18}$$

each associated with different parameters (attributes)

$$p^i[kT,(k+1)T] = \begin{pmatrix} x_*^i[kT], \\ d_*^i[kT,(k+1)T), \\ x_*^i[(k+1)T], \\ y_*^i[kT,(k+1)T), \\ u_*^i[kT,(k+1)T), \\ J_*^i \end{pmatrix}$$

where p^i is used for notational compactness in eq. (7.18). Section 7.5 provides a method to generate such a collection. Optimal control sequence $u_*^i[kT,(k+1)T)$ and an induced delay J_*^i are also computed and stored during the contract generation process.

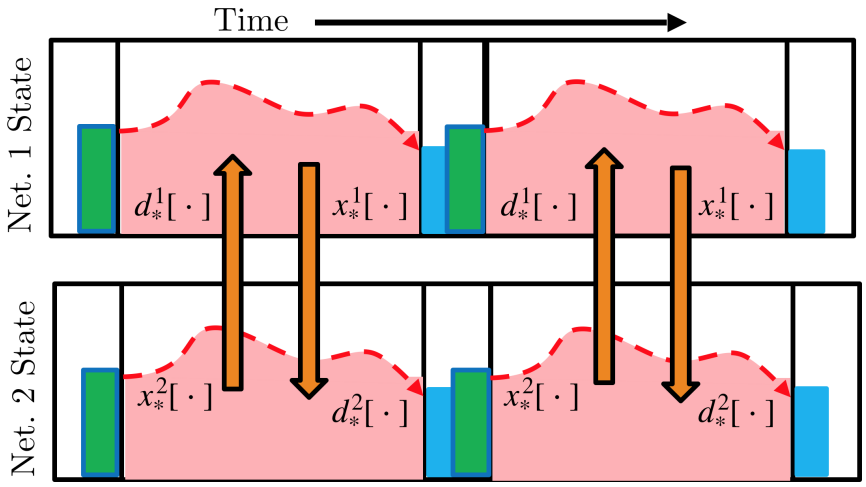


Figure 7.3: Visualization of contract parameter roles over multiple intervals of length T. Recursive feasibility is encoded as an intra-network contract depicted as a initial state assumption $x_*[k]$ in green and terminal guarantee $x_*[(k+1)T]$ in blue. The inter-network contract is depicted as the maximum incoming flow assumption $d_*[kT,(k+1)T)$ in yellow and the maximum outgoing flow guarantee $x_*[kT,(k+1)T)$ in red.

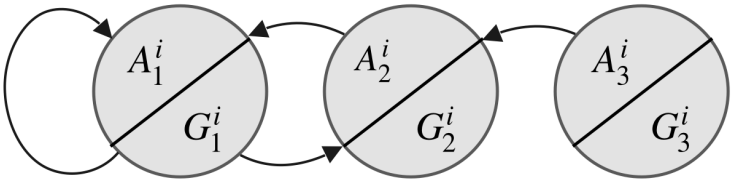


Figure 7.4: Visualization of a single local coordinator associated with an individual network $\mathcal{N}^i$. Each node represents a specific collection of assume-guarantee parameters. Transitions across different parameters encode that a contract handoff is feasible and eq. (7.19) is satisfied. Infinite horizon recursive feasibility corresponds to a cycle.

Designing a Contract Coordinator

A contract coordinator's role is to ensure that parameters across multiple networks are compatible and that parameter switches over time do not inadvertently cause specification violations.

Preserving formal guarantees restricts how contract parameters may change at runtime. First, contracts parameters must always be consistent in the sense of Definition 31. Second, the notion of recursive feasibility needs to be modified to accommodate a changing set of requirements.

Concretely, a contract coordinator is a transition system with state space $\mathcal{P} = \prod_{i=1}^N \mathcal{P}_i = \prod_{i=1}^N \{\mathcal{C}^i(p_1^i), \ldots, \mathcal{C}^i(p_{m_i}^i)\}$ designed to ensure that these two properties are satisfied. Every coordinator transition corresponds to a potential change in contract parameters for each

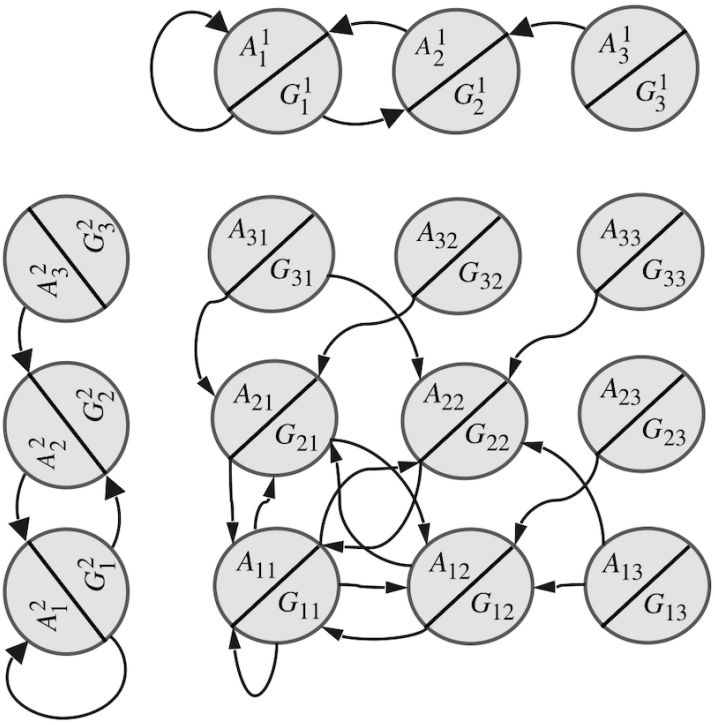


Figure 7.5: Visualization of contract coordinator constructed from networks $\mathcal{N}^1$ and $\mathcal{N}^2$. The finite state machine's state space is the product of the individual coordinator parameter spaces and the transitions are consistent with the individual coordinator transitions.

network and may execute every T time steps. After a transition, each sub-network is notified of the contract it must satisfy.

Consistent Contract Parameters

A coordinator state $p \in \mathcal{P}$ is a tuple $(p^1_{k_1}, \ldots, p^N_{k_N})$ where each network $\mathcal{N}^i$ picks a single contract $\mathcal{C}(p^i_{k_i})$ it would like to satisfy. Not all elements of this space satisfy the contract consistency requirement in definition 31. For instance in fig. 7.1 satisfaction of $\mathcal{N}^1$'s assumption is determined by $\mathcal{N}^2$ and $\mathcal{N}^4$'s guarantees. If contract parameters are such that $\mathcal{N}^2, \mathcal{N}^4$'s guarantees do not jointly imply assumption $\mathcal{N}^1$'s assumption, then these contract parameters are inconsistent.

A subset $\mathcal{P}_v \subseteq \mathcal{P}$ of the parameter space that corresponds to all consistent network-wide parameters. Set $\mathcal{P}_v$ is enumerated via a depth first traversal over a tree with depth N and branching factors m_i. The traversal aggressively prunes branches of the contract space as soon as a contract inconsistency parameter is identified.

Contract Transitions and Recursive Feasibility

A contract transition is valid when each sub-network can promise its future self the ability to satisfy the new contract via a transition from the old contract. Given two consistent contract parameters $p, \hat{p} \in \mathcal{P}_v$ where p is for use over $[kT, (k+1)T]$ and $\hat{p}$ is for use over $[(k+1)T, (k+2)T]$, a switch from p to $\hat{p}$ is *valid* if the following element-wise inequality holds:

$$\bigwedge_{l \in \mathcal{L}} x_*^l[(k+1)T] \leq \hat{x}_*^l[(k+1)T]. \tag{7.19}$$

The left side is the terminal state guarantee from p and the right is the initial state assumption of $\hat{p}$. Recursive feasibility as in section 7.3 is a special case when $p = \hat{p}$.

We define the contract parameters *Viable Graph* (VG) as a directed graph $(\mathcal{P}_v, \mathcal{E}_v)$, where $\mathcal{P}_v$ is the set of nodes, and $\mathcal{E}_v \subseteq \mathcal{P}_v \times \mathcal{P}_v$ is the set of edges such that $\forall (p, \hat{p}) \in \mathcal{E}_v$, the switch from p to $\hat{p}$ is valid. We denote $\mathcal{E}_v^p = \{(p, \hat{p}) | (p, \hat{p}) \in \mathcal{E}_v\}$. A node p is a dead-end if $\mathcal{E}_v^p = \varnothing$. If no dead-end is reached, then there always exists a consistent contract with feasible transition options to other contracts, which by construction implies the following statement.

Proposition 9. *Given a infinite-time contract parameter sequence $p_0, p_1, \cdots$, where p_k is used for control synthesis in the time interval $[kT, (k+1)T]$, the specification* (7.9) *is satisfied if $(p_k, p_{k+1}) \in \mathcal{E}_v$ and $p_k \in \mathcal{P}_v, \forall k \in \mathbb{N}$.*

Optimal Contract Coordination

The recursive feasibility property and contract consistency require that no dead-end node in VG is reached. By recursively removing the dead-end nodes and the edges leading to them, we obtain a fixed point which characterizes the *viable kernel graph* (VKG) $(\mathcal{P}_{v,\kappa}, \mathcal{T}_{v,\kappa})$, where $\mathcal{P}_{v,\kappa} \subset \mathcal{P}_v$ and $\mathcal{T}_{v,\kappa} \subseteq \mathcal{P}_{v,\kappa} \times \mathcal{P}_{v,\kappa}$ with the extra property that $\forall p \in \mathcal{P}_{v,\kappa}, \mathcal{E}_{v,\kappa}^p \neq \varnothing$. Once a parameter contract of a node in VKG is chosen, there always exist a feasible handover of the contract to another node in VKG, establishing infinite-time recursive feasibility and consistency.

Each contract p^i corresponds to a cost J_*^i for network $\mathcal{N}^i$, which is the delay induced if control sequence $u_*^i[kT, (k+1)T)$ is applied starting from $x_*^i[kT]$ under the demand assumptions $d_*^i[kT, (k+1)T)$. It follows from monotonicity properties that J_*^i is a upper-bound for possible costs in real-time implementation. Given a contract parameter $p = (p_{k_1}^1, \cdots, p_{k_N}^N)$, the sum of associated contract costs is

$$c(p) := \sum_{i=1}^{N} J_*^i(p_{k_i}). \tag{7.20}$$

Now we determine which contract parameter from VKG nodes to choose at each time $kT, k \in \mathbb{N}$. In order to aim for optimality, we choose the contract parameter for which the infinite-horizon cost $c_\infty(p) = \sum_{k=0}^{\infty} \alpha^k c(p_k)$ is minimal, where $p = p_0$, and $\alpha \in (0, 1)$ is a discount

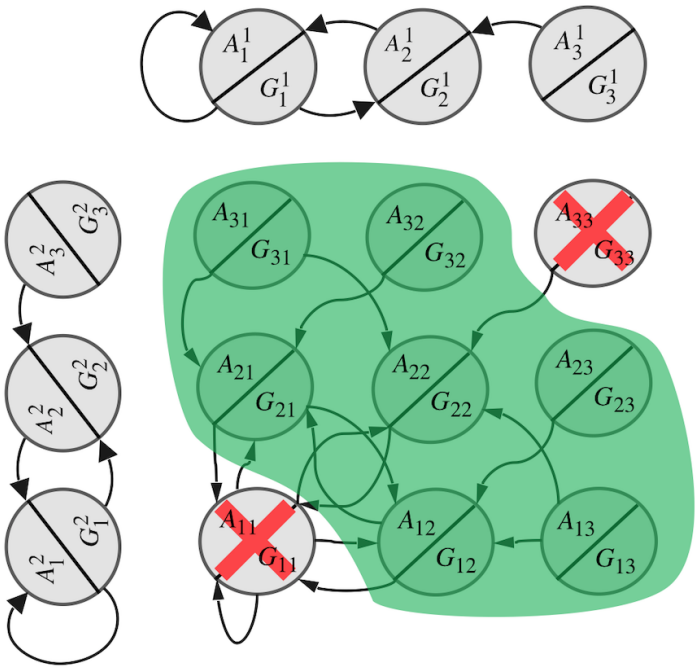


Figure 7.6: Visualization of a viability kernel (green) induced by inconsistent contract parameters (red crosses). From any collection of contract parameters in the viability kernel, it is possible to prevent the inconsistent contract parameters at any time in the future.

factor to make the cost properly defined. Denote the optimum cost to go from p by c^*_∞ which follows from Bellman's equation [9]:

$$c^*_\infty(p) = c(p) + \alpha \min_{p' \in \mathcal{E}^p_{v,\kappa}} (J^*_\infty(p')) \tag{7.21}$$

In order to find $c^*_\infty(p), p \in \mathcal{P}_{v,\kappa}$, (7.21) is cast as a linear program. Finally, at time kT we choose the optimal contract parameter $p^* \in \mathcal{P}_{x[KT]}$ with minimum $c^*_\infty(p)$.

7.5 Contract Mining

A delicate tradeoff exists between conservative assumptions which can accommodate an influx of vehicles and aggressive guarantees which quickly dissipate vehicles in the network. We present a heuristic to generate a set of assume-guarantee pairs for each sub-network.

Given a fixed assumption and $\mathcal{N}^i$, a miner is a bounded horizon optimization algorithm that computes a control trajectory that induces minimal guarantees. Guarantee parameters $x^i_*[(k+1)T], y^i_*[kT,(k+1)T)$ are minimal if contract satisfaction as in Definition 30 is infeasible for any smaller guarantee pair such that $\hat{x}^i_*[kT] \leq x^i_*[kT]$ and $\hat{y}^i_*[kT,(k+1)T) \leq y^i_*[kT,(k+1)T)$ with element-wise inequality. The miner's optimization objective can be any monotone function of $\mathcal{X} \times \mathcal{U}$; we opt to minimize a combination of the l_1 and l_∞ norms.

After mining, a guarantee is propagated into an assumption for adjacent networks via Equation (7.16) with the equality is replaced with an inequality, and the mining continues.

Algorithm 3 provides pseudocode which generates `MaxIter` guarantees for every sub-network. The contract sets are initially empty. Infeasibility of the mining algorithm triggers a multiplicative decrease in the initial conditions and disturbance by a factor $\gamma \in (0, 1)$ until the contract is satisfied. Propagation can be visualized over the bottom of Figure 7.1 where inter-network promises (solid edges) and a network's recursive feasibility constraint (dashed edges) are both updated. Both contract parameters and the control sequence are saved.

Algorithm 3 Guarantee Mining Algorithm

```
Set N = Number of Sub-networks
for all i = 1, ..., MaxIter do
    for all i ∈ {1, ..., N} do
        while True do
            Feas, x[·], u[·], y[·]= mine(N^i,x[0], d[·])
            if Feas then
                Break
            x[0] := γx[0], d[·] := γd[·]
        Add to N^i's contract set
        Propagate Guarantee
```

7.6 Example

Figure 7.1's network is used to evaluate the efficacy of the dynamic contracts system. Algorithm 3 is used to generate a set of 25 contract parameters for each sub-network. There are $|\mathcal{P}_v| = 1363$ consistent contract parameters, of which 664 were members of the viability kernel $\mathcal{P}_{v,\kappa}$. All optimization problems were posed and solved using `Gurobi`'s mixed integer linear program solver [39]. We used the method in Section 7.4 to change contract parameters as a feedback of system state every T time steps. We simulated the network for 30 time steps (5 rounds of contract transitions). The satisfaction of the specification was implicitly implied by the fact that the MPC optimization problem was feasible at all times. Sample results are illustrated in Figure 7.7.

We also compared the optimality of our results with other control methods. Table 7.1 shows the accumulated delay for different network conditions and control architectures. The first column shows the subset of networks that experience a fully adversarial demand, e.g. the $(1, 3)$ column means that $\mathcal{N}^1, \mathcal{N}^3$ experience the maximum number of incoming vehicles and $\mathcal{N}^2, \mathcal{N}^4$ experience no exogenous demand. As expected, the cumulative delay decreases when the network load decreases. The fixed controller executes a control sequence without state feedback in the interval $(kT, (k + 1)T)$, but permits contract switches every T steps. The MPC controller with fixed contracts achieves similar objective values, but is not strictly better than the fixed control with dynamic contracts, suggesting that contract constraints

Table 7.1: Accumulated Delays for Different Control Methods

Networks Experiencing Full Demand	**Dynamic Contracts Fixed Control**	**Fixed Contracts MPC**	**Dynamic Contracts MPC**
All	784	753	747
(1,4)	354	381	346
(2,4)	248	244	226
(1,2,4)	513	513	495
(1,2,3)	670	646	635
(1,2)	405	398	394
(1,3)	498	507	460
(1)	238	249	229
(2)	154	122	104
(4)	115	86	85

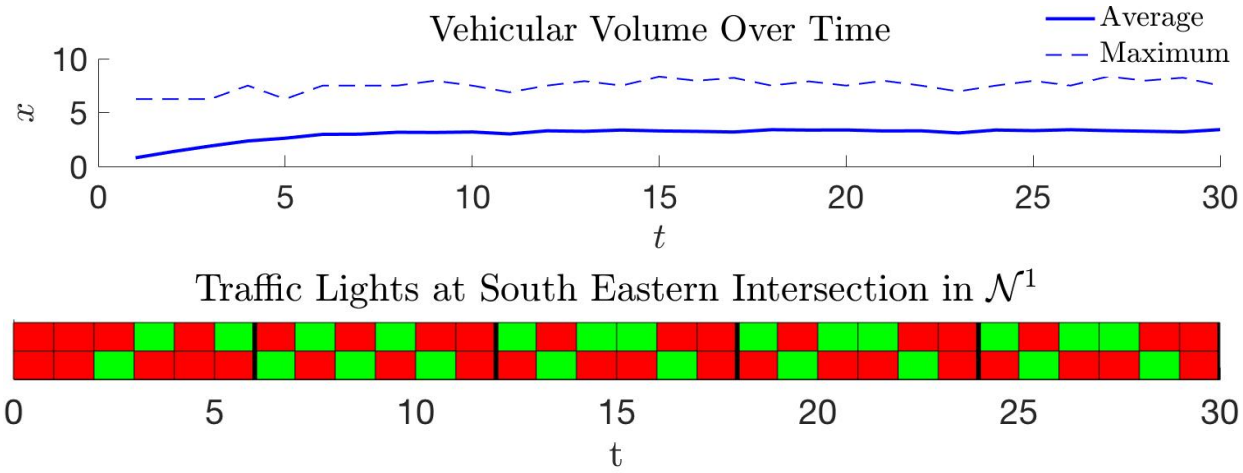


Figure 7.7: Simulation Results: (Top) Aggregated state of the network over time. (Bottom) The traffic lights history for the links in South-Eastern Intersection of $\mathcal{N}^1$. The lower color is for the light corresponding to the link in North-South direction and the upper one stands for the link in East-West direction. The pedestrian liveness requirement (both lights simultaneously getting red) is satisfied in each round of contract transitions (shown by thick vertical block lines).

are a major impediment for achieving a lower delay. The dynamic contracts with MPC column outperforms both other control strategies. The performance gain is greater when the exogenous demand has an asymmetric profile, when dynamic contracts assign higher priority to sub-networks experiencing higher demand. This suggests the usefulness of dynamic contracts. On the other hand, when the exogenous demands are uniformly entering the network in different directions, the performance benefit gained from dynamic contracts is negligible, which suggests that a fixed contract is appropriate for traffic networks with static demand profiles.

Chapter 8

Identifying Communication Requirements

Interaction amongst agents can come in various forms such as coupled dynamics, coupling constraints, or a joint optimization objective. A common facet of multi-agent systems is the use of a distributed control architecture, where each agent has authority over different sets of actuators, and an accompanying communication network for agents to coordinate their actions or observe others. Communication and collective decision making facilitate complex interactions amongst agents and enable them to reliably achieve collective behaviors that would otherwise be difficult to accomplish without some coordination protocol.

In this chapter, we consider the problem of satisfying a safety objective with a controller that is distributed over multiple agents. We say that these agents are coordinating within a given time step if they communicate and collectively agree upon actions to execute. As a motivating example, consider two fully autonomous vehicles equipped with vehicle-to-vehicle (V2V) communication and tasked with avoiding a collision. At one extreme are scenarios where no communication is necessary due to a sufficiently large distance between the vehicles, while at the other extreme are near miss scenarios where collisions are only avoided through precise timing, actuation, or luck. Preemptive cooperation enabled by V2V communication is designed to help the vehicles avoid these danger scenarios and for vehicles to negotiate collision-free trajectories.

How can one distinguish between these extremes and determine when multi-system coordination is and is not necessary to maintain a safety objective? We present a method that takes a closed loop control system and a safety requirement, then identifies a subset of the state space that is robustly safe against temporary communication losses. This subset naturally shrinks with time as the duration of the communication loss increases. At its core, our method iterates an appropriate operator which propagates a coordination-free region and resembles fixed point algorithms in the literature on symbolic system verification. This operator is defined such that it incorporates information about the system dynamics and the controller architecture. These results are first used to consider a scenario when multiple agents want to cooperate, but can only do so after some delay. We then develop a self-

triggered coordination scheme where agents can preemptively schedule when they would like to communicate, while still maintaining safety guarantees.

This chapter tackles a new problem that has not, to the best of our knowledge, been addressed within the control theory literature. Compared to other work, we do not assume a decomposition of the state space[25, 18] nor is the objective assumed to be decomposable [18]. Instead we only consider a decomposition of the control input space and can thus accommodate instances when there are complex coupling dynamics that need to be reasoned about collectively. This work leverages compositional tools and techniques developed for formal controller synthesis. These may involve constructing abstractions compositionally [77], decomposing the controller synthesis procedure [49][61], or decomposing the controller itself [82]. Assume-guarantee reasoning has also been used for compositional synthesis with multiple agents by abstracting out internal information that is irrelevant to reason about system interactions [67]. Chapter 7 on dynamic contracts provided *sufficient* conditions for multiple agents to jointly enforce a specification with communication limited to the contract parameter space. It also assumed that the specification was decomposable. This work on the other hand seeks to identify a set of states where it is *necessary* for agents to cooperate; computing this set exactly is computationally difficult, so we leverage the theoretical machinery of Chapter 4 to compute approximations. Our self-triggering communication scheme may be compared to similar schemes in the self-triggered control literature [40], where often the objective is to minimize the energy expended by sensors and actuators subjected to a stability constraint [13][37]. Our work instead seeks to minimize the communication overhead incurred as multiple agents negotiate safe actions.

8.1 Formulation

Notation

Given two sets $\mathcal{A}$ and $\mathcal{B}$, let $|\mathcal{A}|$, $2^{\mathcal{A}}$, and $\mathcal{A} \times \mathcal{B}$ respectively represent $\mathcal{A}$'s cardinality, $\mathcal{A}$'s power set (set of all subsets), and the Cartesian product between $\mathcal{A}$ and $\mathcal{B}$. Let $\mathbb{R}$, z represent the real and integer numbers respectively, while $\mathbb{R}_{\geq 0}$ and $\mathbb{Z}_{\geq 0} = \mathbb{N}$ are their non-negative counterparts. With an appropriate universal set Ω, $\mathcal{A}$'s complement $\mathcal{A}^C$ is defined as $\Omega \setminus \mathcal{A}$. Given a Cartesian product of M sets $\prod_{i=1}^{M} \mathcal{A}_i$ and a subset $L \subseteq \prod_{i=1}^{M} \mathcal{A}_i$, the projection operation $\pi_{\mathcal{A}_j} : \prod_{i=1}^{M} \mathcal{A}_i \to \mathcal{A}_j$ retains the coordinates associated with $\mathcal{A}_j$ and is defined as:

$$\pi_{\mathcal{A}_j}(L) = \{a_j \in \mathcal{A}_j : \exists (a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_M) \text{ such that } (a_1, \ldots, a_M) \in L\}. \tag{8.1}$$

Signals and Systems

An interval $[a, b]$ where $a, b \in z$ includes both end points. Let $[a, b) = [a, b-1]$ and $[a] = [a, a]$. Given a space $\mathcal{P}$, the space of trajectories evolving in $\mathcal{P}$ is $\mathcal{P}[\cdot]$. A trajectory $p[\cdot]$ over time interval $\mathcal{I}$ is a map $p[\cdot] : \mathcal{I} \to \mathcal{P}$. Let $\mathcal{X}$ and $\mathcal{U}$ represent a system's state and input spaces respectively. Sets $\mathcal{X}[\cdot]$ and $\mathcal{U}[\cdot]$ are referred to as state and input trajectory sets. This chapter

deals with systems where the input space $\mathcal{U}$ consists of N components so that $\mathcal{U} = \prod_{i=1}^{N} \mathcal{U}_i$ [1]. Each of these N components is thought of as an individual agent. The system's discrete-time dynamics are given by a relation $f \subseteq \mathcal{X} \times \mathcal{U} \times \mathcal{X}$, which can also be viewed as a set-valued function $f : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$. Let $\mathcal{U}(x) = \{u \in \mathcal{U} : f(x, u) \neq \varnothing\}$ denote the set of non-blocking control inputs at x.

A memoryless controller for system f is a relation $C \subseteq \mathcal{X} \times \mathcal{U}$. The set of states $\mathcal{B} = \{x \in \mathcal{X} : (x, u) \notin C \text{ for all } u \in \mathcal{U}\}$ is the set of blocking states under controller C. A controller may also be viewed as a function $C : \mathcal{X} \to 2^{\mathcal{U}}$ that maps states to sets of admissible inputs (states with no corresponding control input map to an empty set). A controller C and system f can be interconnected into a closed loop system denoted as $f \circ C : \mathcal{X} \to 2^{\mathcal{X}}$ [2]. The next state $x[k+1]$ satisfies $x[k+1] \in f \circ C(x[k]))$ if and only if there exists a $u[k] \in C(x[k])$ such that $x[k+1] \in f(x[k], u[k])$. All sequences $x[\cdot]$ that satisfy the aforementioned condition and $x[0] \in \mathcal{L}$ are said to be generated by the closed loop system $f \circ C$ with initial state set $\mathcal{L} \subseteq \mathcal{X}$.

Control for Safety

Safety is a common requirement for cyber-physical systems. We encapsulate this notion of safety as a region of the state space $\mathcal{S} \subseteq \mathcal{X}$ that should never be exited. For a vehicle, set $\mathcal{S}$ could represent a collision-free zone and a speed limit, while for a medical device $\mathcal{S}$ could represent safe blood sugar levels.

Definition 32. *Let $\mathcal{S} \subseteq \mathcal{X}$ be a set of safe states. A control policy $C : \mathcal{X} \to 2^{\mathcal{U}}$ and initial set $\mathcal{L} \subseteq \mathcal{S}$ is said to satisfy safety constraint $\mathcal{S}$ if all trajectories generated by a closed loop system $f \circ C$ with any initial state $x[0] \in \mathcal{L}$ never exit $\mathcal{S}$.*

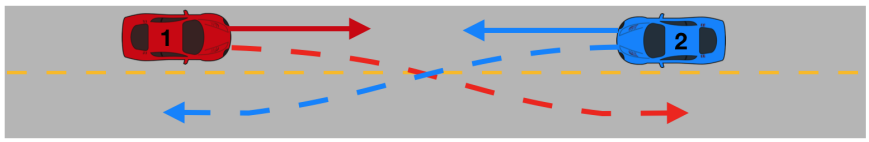
At each state x, there is a set of admissible control inputs $C(x) \subseteq \mathcal{U}$. A controller is deterministic if $|C(x)| = 1$ only permits one action for all $x \in \mathcal{X}$. Although determinism simplifies analysis of a closed loop system, deterministic controllers may be too restrictive if the system needs to satisfy additional requirements on top of safety. For instance if two vehicles want to avoid a collision, then a safe controller can simply enforce that both vehicles have zero velocity but this prevents vehicles from reaching a desired location.

Loss of Safety Guarantees with a Distributed Controller

More permissive controllers can act as supervisors that restrict control actions only enough to ensure safety. They are useful because they can be combined with other controllers that seek to achieve other objectives such as reaching a region. When a distributed controller is deployed on multiple systems without an underlying communication scheme, the non-determinism contained in permissive controllers can lead to safety violations.

[1] Some $\mathcal{U}_i$ may be multi-dimensional so N is not necessarily the dimension of $\mathcal{U}$.

[2] This notation was inspired by $\circ$'s usage as a function composition operator. However, it is not a composition in the strictest sense where $f(g(x)) = (f \circ g)(x)$.



	Right Vehicle Change	Right Vehicle Stay
Left Vehicle Change	Collision	No Collision
Left Vehicle Stay	No Collision	Collision

Figure 8.1: Motivating Collision Avoidance Example

If $\mathcal{U} = \prod_{i=1}^{N} \mathcal{U}_i$ is decomposed into N inputs that are each under control from a different agent, then each must concurrently select a single input u_i such that

$$(u_1, \ldots, u_N) \in C(x). \tag{8.2}$$

It is this step where multiple agents concurrently select an input that leads to coordination hazards. Whenever $|C(x)| > 1$ then assuring that (8.2) holds is not always possible without explicit coordination and communication with other agents.

Example 5 (Illustrative Example)**.** *Consider a scenario depicted in Figure 8.1 where two vehicles are facing one another and a collision is imminent. Both vehicles can choose between staying in their lane or switching to the other lane and a collision is avoided only when one vehicle switches. Clearly it is possible for a collision to be avoided as long as the two vehicles are able to communicate and negotiate which one changes lanes. On the other hand suppose that these vehicles are not equipped with V2V communications. If a collision does occur it is not possible to assign fault to solely one vehicle because from both vehicles' points of view its action was safe as long as the other vehicle responded with the appropriate action. Instead one can only attribute the fault to both agents' failure to negotiate.*

To formalize the notion of coordination, we first define a minimal independent controller $\texttt{IND}_C$ associated with C. The set of possible controller actions at x is $\texttt{IND}_C(x)$ and depicted in Figure 8.2.

$$\texttt{IND}_C(x) := \prod_{i=1}^{N} \pi_{\mathcal{U}_i} C(x). \tag{8.3}$$

The projection $\pi_{\mathcal{U}_i} C(x)$ of this controller onto each agent i's individual component $\mathcal{U}_i$ yields the set of all control inputs permitted at state x without any information about how other agents behave. Any input $u_i \notin \pi_{\mathcal{U}_i} C(x)$ indicates that agent i is either reckless or malicious. If all agents pick a $u_i \in \pi_{\mathcal{U}_i} C(x)$ then they have all reasonably attempted to satisfy the safety condition by selecting a point $(u_1, \ldots, u_N) \in \texttt{IND}_C(x)$, but the joint

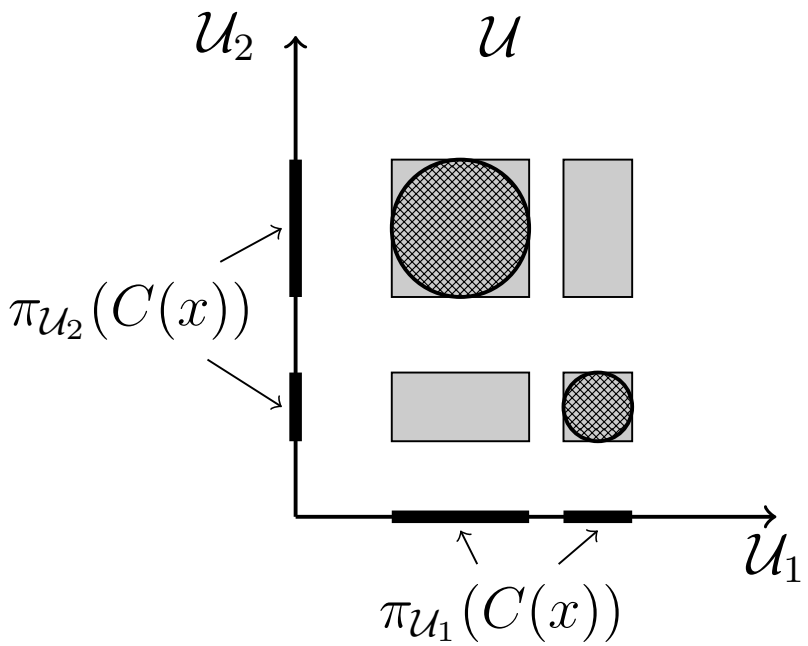


Figure 8.2: For some fixed $x \in \mathcal{X}$, the original safe control set $C(x)$ (union of circular patterned region) is projected onto the axes and yields $\pi_{\mathcal{U}_1}(C(x))$ and $\pi_{\mathcal{U}_2}(C(x))$ (thick lines on axes). Combining the projections gives the coordination-free counterpart $\texttt{IND}_C(x)$ (darker regions) defined in Section 8.2.

condition $(u_1, \ldots, u_N) \in C(x)$ is not necessarily satisfied because $C(x) \subseteq \texttt{IND}_C(x)$. The independent controller $\texttt{IND}_C$ may also be viewed as the set of possible control actions that are reasonable in the undesirable situation where each agent believes itself to be the leader and relies on the other agents to be followers that respond to the leader's choice. The set $\texttt{IND}_C(x) \subseteq \mathcal{U}$ is the minimal independent set that contains $C(x)$.

Throughout the rest of this chapter, we analyze properties of the new closed loop system $f \circ \texttt{IND}_C$, which is derived from $f \circ C$ but exhibits additional behaviors due to the absence of coordination.

Note that the set of trajectories that are exhibited under $f \circ C$ is a subset of those exhibited under $f \circ \texttt{IND}_C$. Thus, even though the original system $f \circ C$ may be safe, $f \circ \texttt{IND}_C$ may exhibit unsafe trajectories.

Problem 4. *Given a set of dynamics f, a distributed controller $\texttt{IND}_C$, a safe region $\mathcal{S}$, and coordination-free interval $I = [a, b)$ identify a subset of the state space $\mathcal{L}$ such that all behaviors of $f \circ \texttt{IND}_C$ with initial state $x[a] \in \mathcal{L}$ remain in $\mathcal{S}$ within the interval I.*

Remarks on Coordination with Mesh Networks

V2V technology also enables the creation of ad hoc vehicular mesh networks which enables applications in cooperative cruise control, vehicular platoons, and congestion mitigation. Suppose each agent is represented by a vertex in an undirected graph and two agents with a V2V have their corresponding vertices connected by an edge. Such a graph can be grouped into equivalence classes corresponding to its connected components. We assume that agents in the same class can communicate instantly even if they are separated by more than one edge.

Assumption 2. *Each agent in an equivalence class can coordinate with all other agents in that class within each time step k.*

In practice, Assumption 2 is a requirement that the time scale over which messages is passed in the network are effectively instantaneous relative to the time scale of the physical dynamics. The independence definition of Equation (8.3) was stated under the assumption that each $\mathcal{U}_i$ corresponded to one agent and that no agents cooperate. If agent cooperation occurs over a mesh network with P connected components, then the independence condition corresponds to the connected components of the graph. For each of $l = 1, \ldots, P$ equivalence classes, let $\hat{\mathcal{U}}_l$ be the Cartesian product of the coordinates $\mathcal{U}_i$ that belong to that class.

$$\texttt{IND}_C(x) := \prod_{l=1}^{P} \pi_{\hat{\mathcal{U}}_l} C(x). \tag{8.4}$$

This formulation allows for a platoon to be treated as a single agent instead of a collection of vehicles. For notational simplicity, we simply assume that the decomposition into equivalence classes is given and use Equation (8.3) throughout the rest of this chapter.

8.2 Coordination-Free Operator

Given some controller $C \subseteq \mathcal{X} \times \mathcal{U}$, we use the associated minimally restrictive independent controller from Equation (8.3) as a formal characterization of all the possible actions with a distributed implementation of C in the absence of coordination.

The set of predecessor states which enforce membership within a region $Z \subseteq \mathcal{X}$ without coordination is computed with the operator

$$\texttt{IPRE}(Z) = \{x : x \in \pi_{\mathcal{X}}(\texttt{IND}_C)\} \cap \{x : \varnothing \neq f(x,u) \subseteq Z \text{ for all } u \in \texttt{IND}_C(x)\}. \tag{8.5}$$

The first set ensures that there is always a valid input because $\pi_{\mathcal{X}}(\texttt{IND}_C)$ is a state domain over which the controller produces admissible inputs. The second set takes into account the system dynamics and ensures that all states are in Z. A state in $\texttt{IPRE}(Z)$ is robust in the sense that all future possible next states $f(x,u)$ are contained in Z *despite* uncertainty about which $u \in \texttt{IND}_C(x)$ is chosen.

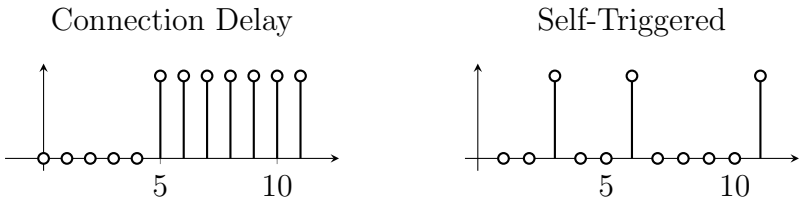


Figure 8.3: Two scenarios with intermittent connections. A high value signifies an established connection.

Operator $\texttt{SIPRE}_{\mathcal{S}}$ below identifies states that can stay in Z and remain safely in $\mathcal{S}$ without coordination

$$\texttt{SIPRE}_{\mathcal{S}}(Z) = Z \cap \texttt{IPRE}(Z) \cap \mathcal{S}. \tag{8.6}$$

By iterating this operator k times, we can identify a region of the state space that remains in $\mathcal{S}$ for k time steps despite communication losses. Both operators are simple modifications on standard controllable predecessor operators [86].

Remarks about Algorithmic Implementation

Set intersection, union, negation, and projection are the main operations that are required to compute Equation (8.5) and Equation (8.6) exactly. In a continuous domain, support for these algebraic operations may only be possible to encode for a specific set of system dynamics and constraints (consider for instance linear system dynamics and constraints given as unions of polyhedra). However in the scenario where state and inputs spaces are finite, binary decision diagrams (BDDs) are an efficient data structure that supports all of the aforementioned operations [14, 15]. Instead of imposing constraints on the system dynamics and safety region, we opt for the finite case by using a grid to approximate a continuous domain. Moreover, there exists a rich theoretical literature of abstraction methods [86] [76] and accompanying software tools such as [78] which construct approximately similar finite systems such that Assumption 3 is satisfied, even if the state and input spaces of system f are dense, continuous subsets of Euclidean space.

Assumption 3. *Both $\mathcal{X}$ and $\mathcal{U}$ are finite sets.*

8.3 Applications

We consider two applications. One is to characterize latency requirements for a wireless communication system and the other is a design for a self-triggered coordination scheme.

Maximum Allowed Connection Delay

Our first application involves N agents that seek to establish a wireless communication channel subject to a maximum connection delay $D \in \mathbb{N}$. Once a connection is established, it is assumed to be maintained as in the left of Figure 8.3 where $D = 5$. If all agents attempt to initiate a connection starting at time k, then they are able to jointly choose a control input starting at time $k + D$.

Definition 33. *A system in state $x[k]$ at time k is robustly safe to connection initialization delays of length D if $x[k, \infty) \in \mathcal{S}$ for all trajectories $x[k, \infty)$ generated by the time varying closed loop system*

$$x[k+1] \in f \circ \mathtt{IND}_C(x[k]) \text{ if } k \in [k, k+D) \tag{8.7}$$
$$x[k+1] \in f \circ C(x[k]) \text{ if } k \in [k+D, \infty) \tag{8.8}$$

where we adopt the convention $[k, k + D) = \varnothing$ if $D = 0$.

The approach to generating the set of states that are robust to connection initialization delays of length D is as follows. We first identify an invariance set $\mathcal{K}$ where the system $f \circ C$ remains in $\mathcal{S}$ along an infinite horizon $[k + D, \infty)$ once $x[k + D] \in \mathcal{K}$. Invariance set $\mathcal{K}$ is distinct from safe set $\mathcal{S}$ because a state $x[k] \in \mathcal{S} \setminus \mathcal{K}$ satisfies the safety condition at time k but is not guaranteed to do so along an infinite horizon. With set $\mathcal{K}$, we then iterate $\mathtt{SIPRE}_{\mathcal{S}}(\mathcal{K})$ D times to identify the states that are guaranteed to reach $\mathcal{K}$ at time $k + D$ without exiting $\mathcal{S}$ within $[k, k + D)$.

To identify $\mathcal{K}$, we define operators that are analogous to `IPRE` and `SIPRE`, except that $\mathtt{IND}_C$ is replaced with C

$$\mathtt{PRE}(Z) = \{x : x \in \pi_{\mathcal{X}}(C)\} \cap \{x : \varnothing \neq f(x, u) \subseteq Z \text{ for all } u \in C(x)\} \tag{8.9}$$
$$\mathtt{SPRE}_{\mathcal{S}}(Z) = Z \cap \mathtt{PRE}(Z) \cap \mathcal{S} \tag{8.10}$$

Lemma 1. *Let $\mathcal{K} := \lim_{i\to\infty} \mathtt{SPRE}^i_{\mathcal{S}}(\mathcal{X})$. Then all trajectories $x[k + D, \infty)$ such that $x[k + D] \in \mathcal{K}$ will never intersect the unsafe set $\mathcal{S}^C$.*

Proof. The Tarski fixed point theorem [89] ensures that the limit on the right hand side exists and is unique if $\mathcal{X}$ is a finite set and $\mathtt{SPRE}_{\mathcal{S}}$ is a monotone operator. Assumption 3 ensures that $\mathcal{X}$ is finite, and monotonicity of $\mathtt{SPRE}_{\mathcal{S}}$ with respect to the set containment ordering can easily be verified. Note that $\mathcal{S} = \mathtt{SPRE}^1_{\mathcal{S}}(\mathcal{X})$. Membership of state $x[k]$ in set $\mathtt{SPRE}\mathcal{S}^{i+1}(\mathcal{X})$ ensures that both $x[k], x[k+1] \in \mathcal{K}$. By induction, given $x[k+D] \in \mathtt{SPRE}^i_{\mathcal{S}}(\mathcal{X})$ and $i > 0$, trajectories from system $f \circ C$ will remain in $\mathcal{S}$ along the interval $[k+D, k+D+i)$. Because the limit set exists, $\lim_{i\to\infty} \mathtt{SPRE}^i_{\mathcal{S}}(\mathcal{X})$ is the set of points that are safe along the interval $[k + D, \infty)$. □

Building on the previous lemma, iterating `SIPRE` D times yields a region where all trajectories of length D are safe without coordination. The closed loop system under $\mathtt{IND}_C$ must never exit $\mathcal{S}$ within the interval $[k, k + D)$, and also must terminate at $x[k + D] \in \mathcal{K}$ so that the system under C can ensure safety along the infinite horizon $[k + D, \infty)$.

Proposition 10. *Let* $\mathcal{K} := \lim_{i\to\infty} \mathit{SPRE}^i_{\mathcal{S}}(\mathcal{K})$. *Then* $\mathit{SIPRE}^k_{\mathcal{S}}(\mathcal{K})$ *is the set of states that are safe under* $\mathtt{IND}_C$ *for* $k-1$ *time steps.*

Proof. Suppose $x[0] \in \mathtt{SIPRE}^k_{\mathcal{K}}(\mathcal{K})$. The set of possible states for $x[1]$ under controller $\mathtt{IND}_C$ is uniquely defined as $\mathtt{SIPRE}^{k-1}_{\mathcal{K}}(\mathcal{K})$ and is non-empty. By induction, a sequence $x[\cdot] = x[0]\dots x[k]$ generated by closed loop system $f \circ \mathtt{IND}_C$ must satisfy $x[j] \in \mathtt{SIPRE}^{k-j}_{\mathcal{K}}(\mathcal{K})$ for all $j \in [0,k]$. By definition $\mathtt{SIPRE}^0_{\mathcal{K}}(\mathcal{K}) = \mathcal{K}$. □

Self-triggered coordination

It is also possible to design a scheduler for triggering communication amongst agents. Each agent maintains a countdown for the latest time communications can be initiated. As the system executes, this time is updated to provide a constantly changing upper bound on the latest time the agents need to communicate. For clarity, we assume that the connection initialization delay as in the previous section is $D = 0$.

The fixed point computation in Proposition 10 yields a sequence of disjoint sets. Define $T : [0, F] \to 2^{\mathcal{X}}$ such that

$$T(k) = \begin{cases} \mathtt{SIPRE}^k_{\mathcal{S}}(\mathcal{K}) \setminus \mathtt{SIPRE}^{k+1}_{\mathcal{S}}(\mathcal{K}) & \text{if } k < F \\ \mathtt{SIPRE}^k_{\mathcal{S}}(\mathcal{K}) & \text{if } k = F \end{cases} \tag{8.11}$$

where $F \in \mathbb{N}$ is the first value where the sequence reaches a fixed point

$$F = \text{argmin}_{i\in\mathbb{N}_{\geq 0}} \mathtt{SIPRE}^{i+1}_{\mathcal{S}}(\mathcal{K}) = \mathtt{SIPRE}^i_{\mathcal{S}}(\mathcal{K}). \tag{8.12}$$

A modified inverse function $\hat{T}^{-1} : \mathcal{X} \to [0, F]$ is given by:

$$\hat{T}^{-1}(x) = \{i \in [1, F] : x \in T(i)\}. \tag{8.13}$$

Because the collection $T(1), \dots, T(F)$ consists of disjoint sets, $\hat{T}^{-1}(x)$ is well defined (i.e. a singleton set) for each $x \in \mathcal{K}$. Because each agent has access to $\hat{T}$ and the state x, they can independently determine the unique value for i such that $x \in T(i)$. A countdown with initial value i is then initialized for each agent. When that value reaches $i = 0$ then the agents coordinate by selecting an action and also initialize a new countdown timer. This framework exhibits reduced communication overhead compared to a centralized architecture, while also preserving the guarantees that are otherwise impossible with a fully decentralized and coordination free controller architecture.

The self-triggered system is defined by augmenting the original system with a countdown that resets after coordination has been triggered.

Definition 34. *The system with a self-triggered communication architecture satisfies the following dynamics.*

$$x[k+1] = \begin{cases} f \circ \texttt{IND}_C(x[k]) & \text{if } i[k] > 0 \\ f \circ C(x[k]) & \text{if } i[k] = 0 \end{cases} \tag{8.14}$$

$$i[k+1] = \begin{cases} i[k] - 1 & \text{if } i[k] > 0 \\ \hat{T}^{-1}(x[k+1]) & \text{if } i[k] = 0 \end{cases} \tag{8.15}$$

Note that when $i[k] = 0$, the counter is reset to $\hat{T}^{-1}(x[k+1])$ after the state transition from Equation (8.14) occurs.

Proposition 11. *If $x[k] \in \mathcal{K}$, then all trajectories $x[k, \infty)$ under the self-triggered communication system from Definition 34 will remain inside $\mathcal{S}$.*

8.4 Examples

In each of our examples, we use a modified version of the SCOTS symbolic controller synthesis toolbox [78], which takes a continuous control system and creates a finite state machine that serves as an abstract representation over which a controller is synthesized. In addition to modifications to compute Equation (8.4) and Equation (8.6), we exploit internal system dependencies to reduce the computation time of the abstraction [38]. Creating the discrete abstraction depends on parameters such as the grid size and granularity. Consider a set $\mathcal{P} = \prod_{i=1}^{N} \mathcal{P}_i$ and a discretization parameter $\eta \in \mathbb{R}^N_{>0}$. Its corresponding discretization grid is $[\mathcal{P}]_\eta := \prod_{i=1}^{N} [\mathcal{P}_i]_{\eta_i}$ where $[\mathcal{P}_i]_{\eta_i} := \{a \in \mathcal{P}_i : a = k\eta_i \text{ with } k \in \mathbb{Z}\}$ is a grid over a single dimension. A full introduction to the underlying theory appears in [86] and is beyond the scope of this paper.

Invariance in a Circle

Two agents each have control over different axes and both need to remain within a circular region.

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \end{aligned} \tag{8.16}$$

Let $\mathcal{X} = \mathcal{U} = [-1, 1] \times [-1, 1]$. Although the dynamics are independent, the safety region is a circle with a radius 0.8 so $\mathcal{S} = \{(x_1, x_2) : x_1^2 + x_2^2 \leq .64\}$ so both agents must coordinate with one another to avoid exiting $\mathcal{S}$ near the boundary. It is clear that the system can always enforce safety within $\mathcal{S}$ simply by picking a control input $(u_1, u_2) := -(x_1, x_2)$.

A discretization of the system dynamics is constructed with a sampling period of $t = .01$. The state space grid $[\mathcal{X}]_\eta$ is constructed with $\eta = [.01, .01]$ and input space grid is $[\mathcal{U}]_\epsilon$ with $\epsilon = [.05, .05]$. Figure 8.4 depicts all safe control inputs at $(x_1, x_2) = (-.62, .5)$ which is near

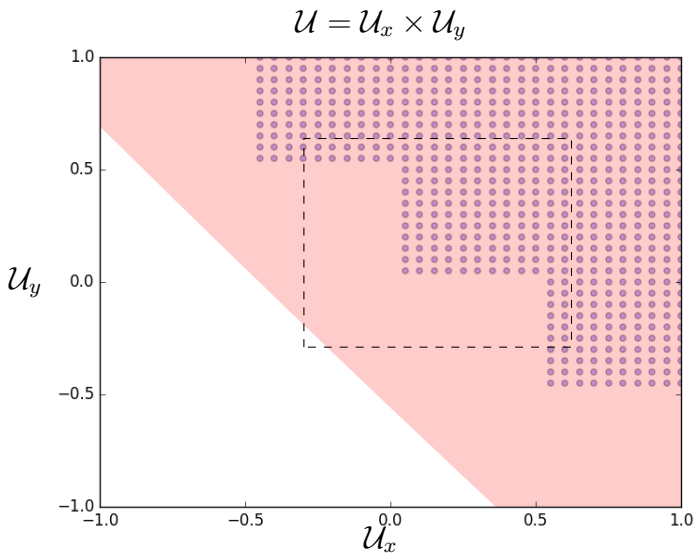


Figure 8.4: Individual dots represent the synthesized safe control set from SCOTS under $C(x)$ at point $x = (x_1, x_2) = (-.62, -.5)$. Without discretization, the true safe action space would be the shaded region in red. The dashed box shows the possible coordination-free actions $\texttt{IND}_C(x)$, which is not contained in the safe action space. Importantly, the synthesized safe inputs are a subset of the true set. Note that $||x||_2 \approx .796$, which is near the boundary of $\mathcal{S}$.

the boundary of $\mathcal{S}$. The staircase shape of the boundary between the safe and unsafe inputs is due to the discretization of the dynamics. Inputs towards the upper right move the state to the interior of $\mathcal{S}$, while safe inputs at the lower left hug the boundary between $\mathcal{S}$ and $\mathcal{S}^C$. If both systems jointly pick low values for u_1 and u_2 then a violation occurs, however both agents can pick $u_1, u_2 = -1$ if the other agent concedes and chooses a higher value.

Figure 8.5 depicts the propagation at various time steps of the coordination-free region via the `SIPRE` operator in Section 8.2. Figure 8.5 shows that a system beginning at the origin can experience an uncoordinated collision is possible after 29 discrete time steps which under sampling period $t = .29$ corresponds to an interval of length .29 in continuous time. However for the continuous system the worst case time step is roughly twice as much $.8/\sqrt{2} \approx .565$, which is the case when $u_1, u_2 \in \{-1, 1\}$ and maintain constant values over time. This is mainly due to the discretization errors that arise when abstracting the continuous system to a discrete one. Note that the discretization error does not jeopardize the safety guarantee. Rather, the discrete case underestimates how much time is available for agents to avoid communication, thus providing a more conservative guarantee.

Intersection Collision Avoidance

Consider two vehicles that are approaching an intersection with no stop sign or a traffic signal. They are controlled independently but each are equipped with V2V radios and may

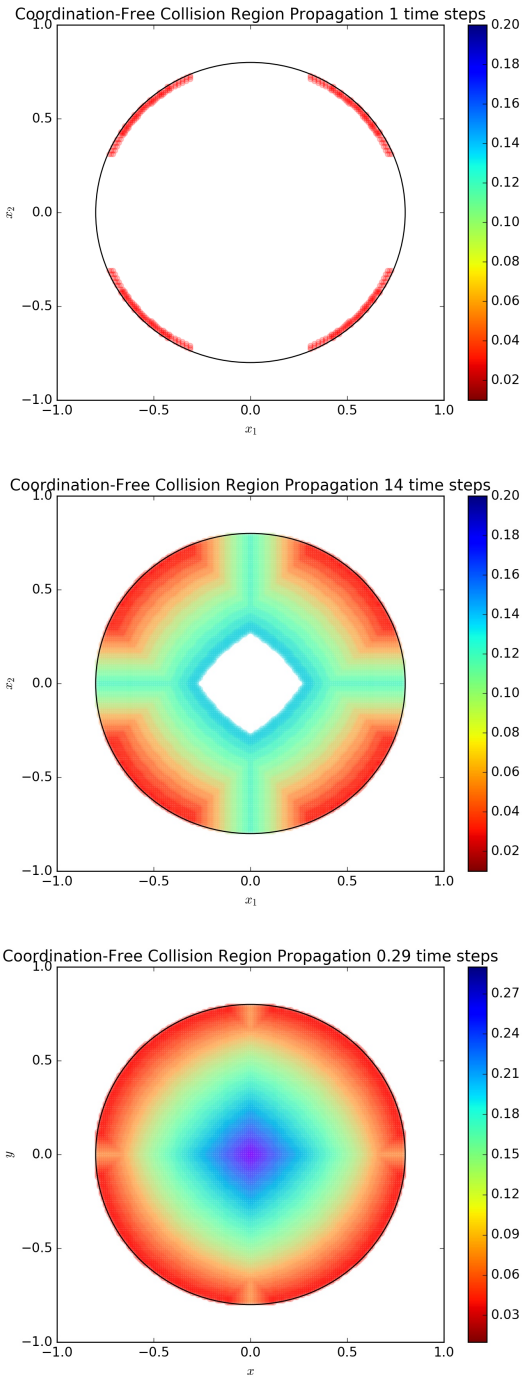


Figure 8.5: Multiple snapshots at $i = 1, 14, 29$ as the region $\mathcal{K} \setminus \texttt{SPRE}^i_{\mathcal{S}}(\mathcal{K})$ grows. One can alternatively visualize $\texttt{SPRE}^i_{\mathcal{S}}(\mathcal{K})$ as a shrinking interior white region as the length of the communication-free interval grows. Red regions represent areas where the system will imminently exit $\mathcal{S}$ unless the two agents coordinate their actions, while blue regions in the interior are only unsafe if the agents do not coordinate for a prolonged period. A fixed point was reached at $i = 29$.

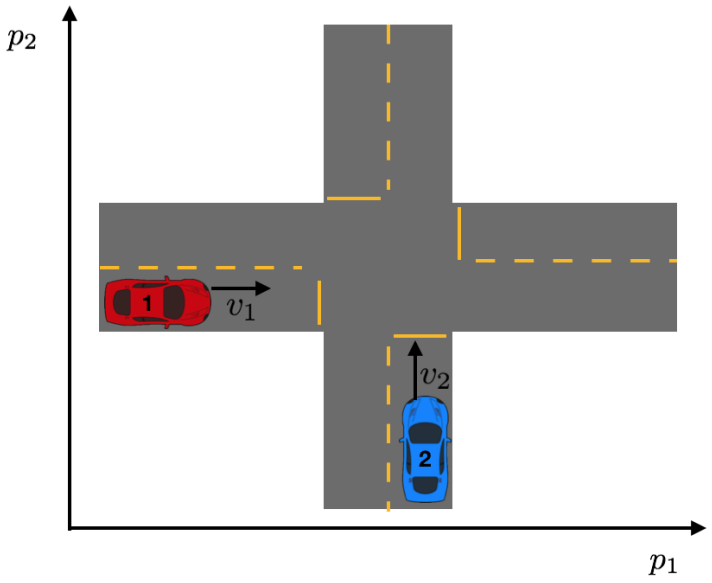


Figure 8.6: Intersection Collision Avoidance

communicate with one another. They also are equipped with enough sensors to identify the position and velocity of all vehicles near the intersection. We consider a simple set of system dynamics given by

$$\dot{p}_i = v_i \tag{8.17}$$
$$\dot{v}_i = u_i - Kv^2 \tag{8.18}$$

with some constant $K = .2$. A higher value for k signifies higher air drag. Let $\mathcal{P}_1, \mathcal{P}_2 = [-10, 10]$ and $\mathcal{V}_1, \mathcal{V}_2 = [0, 3]$. The state space is $\mathcal{X} := \prod_{i=1}^{2} (\mathcal{P}_i \times \mathcal{V}_i)$ and $\mathcal{U} := \prod_{i=1}^{2} [-1, 1]$. The invariant region is the region where at least one vehicle is outside the intersection and no collision has occurred and is succinctly encoded as the set

$$\mathcal{S} := \{x : (|p_1| \geq 2) \vee (|p_2| \geq 2)\}. \tag{8.19}$$

We use the SCOTS toolbox to synthesize a supervisory controller C and compute its corresponding invariance region $\mathcal{K}$ with the procedure in Section 8.3. The system dynamics discretization used a sampling period of $t = .2$, state space grid $[\mathcal{X}]_\eta$ parameter $\eta = [.1, .1, .1, .1]$ and input space grid $[\mathcal{U}]_\epsilon$ parameter $\epsilon = [.1, .1]$.

After synthesizing controller C, its decomposed counterpart $\texttt{IND}_C$ is analyzed. Within $\mathcal{K}^C$ even a centralized controller is unable to guarantee that a collision will *not* occur. This unsafe region is to be avoided and communication is necessary to avoid it. Section 8.4 depicts the 3D projection of $\mathcal{K}^C$ and the evolution of the unsafe region $(\texttt{SIPRE}_{\mathcal{S}}^{D}(\mathcal{K}))^C$ with no communication.

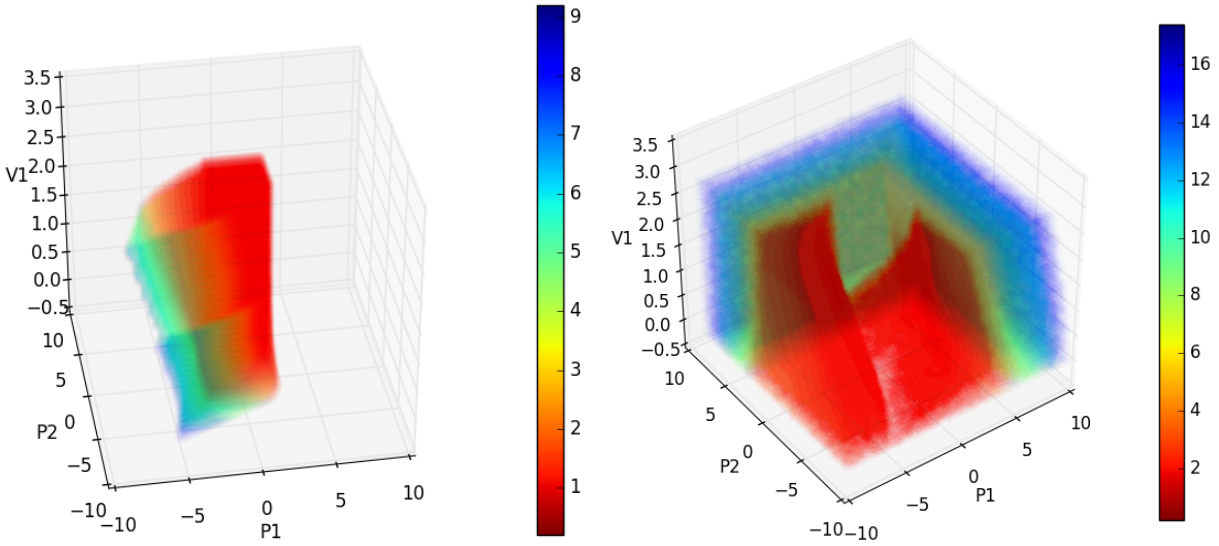


Figure 8.7: (Left) Three dimensional projection of the four dimensional unsafe region $\mathcal{K}^C$ for centralized controller C with $v_2 = 2.8$ held constant. Color scale shows the earliest potential collision time. (Right) Figure shows the unsafe action region $(\texttt{SIPRE}^D_{\mathcal{S}}(\mathcal{K}))^C$ for the system $f \circ \texttt{IND}_C(x)$ expand as communication delay D increases.

Self-Triggered Coordination in a 2D Gridworld

Let there be $N = 2$ agents navigating a 2D grid. Both agents have identical dynamics to Equation (8.16) as shown below with superscripts $i = 1, 2$ as indexes for each agent.

$$\begin{aligned} \dot{x}_1^i &= u_1^i \\ \dot{x}_2^i &= u_2^i \end{aligned} \tag{8.20}$$

The sets $\mathcal{X}^i = [-.2, .2] \times [-.2, .2]$ and $\mathcal{U}^i = [-1, 1] \times [-1, 1]$ for both $i = 1, 2$. A collision has occurred between both agents in the region

$$\mathcal{S}^C = \{(x^1, x^2) \in \mathcal{X}^1 \times \mathcal{X}^2 : \max(|x_1^1 - x_1^2|, |x_2^1 - x_2^2|) < 0.1\}. \tag{8.21}$$

SCOTS is again used to synthesize a centralized controller for the system. The discrete abstraction was constructed with sampling period $\tau = .01$, state space grid $[\mathcal{X}]_\eta$ with parameter $\eta = [.01, .01, .01, .01]$, and input space grid $[\mathcal{U}]_\epsilon$ with parameter $\epsilon = [.2, .2, .2, .2]$. Figure 8.8 shows the trajectory of the system with the self-triggering implementation and how $\hat{T}^{-1}(x[k])$ as defined in Equation (8.13) varies with respect to time.

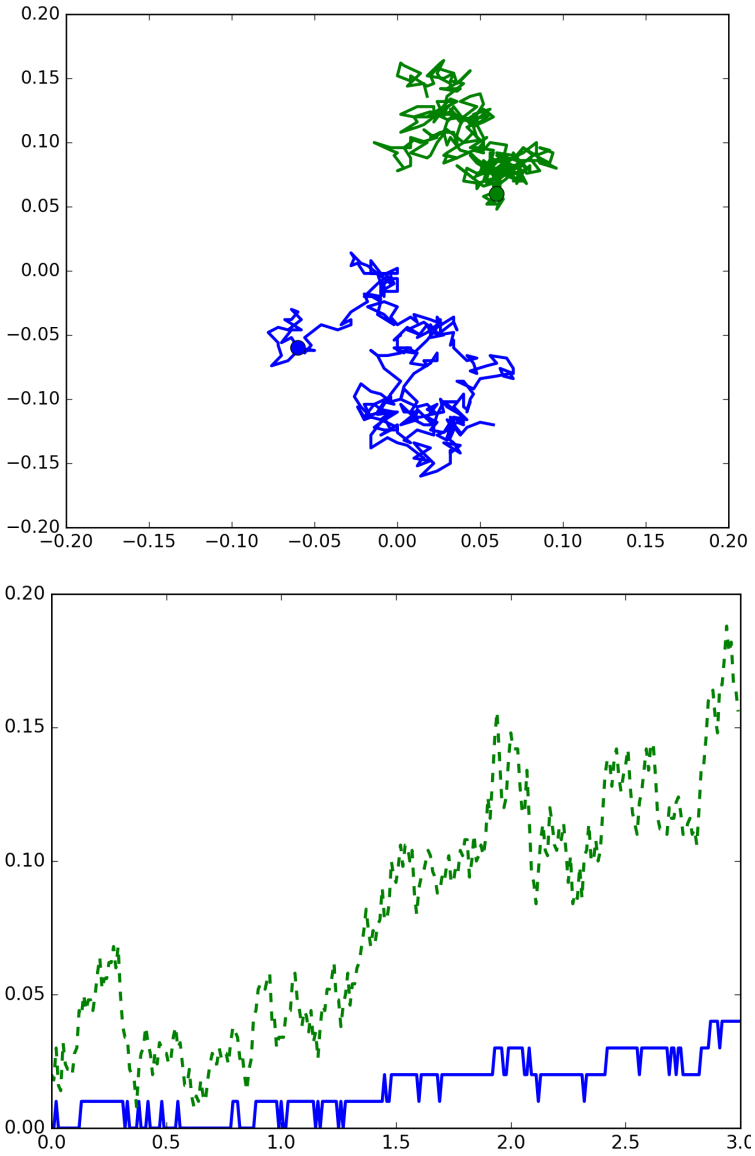


Figure 8.8: (Top) Trajectories of both systems (Bottom) The solid line is the value for $\hat{T}^{-1}(x[k])$ which underapproximates the actual time to when a collision is inevitable $\mathcal{K}^C$. Because Equation (8.16) is fully actuated, the safe set $\mathcal{K}$ and the invariance region $\mathcal{S}$ are identical.

Chapter 9

Conclusion and Future Directions

This thesis has provided a variety of methods to tackle key algorithmic issues in controller synthesis. One of its central goals was to foster a symbiotic relationship between theoretical and algorithmic breakthroughs by easing the transfer of knowledge between the two. It accomplished this by mapping continuous and discrete algorithms for robust controller synthesis onto a common core framework, blending the two and making it easier to integrate custom heuristics or domain specific structure in the algorithms. We've demonstrated that this methodology enables the construction of novel control synthesis algorithms with soundness guarantees. A well tested method to decrease the computational requirements in control synthesis is to capture and encode structure in the synthesis algorithm. This thesis has shown how locality and topological structure can be exploited to reduce the synthesizer computation requirements. It has also shown that decomposing the system dynamics into smaller components eliminates redundant work, and provides a method to derive guarantees about the behavior that arises after interconnecting controlled systems.

Future breakthroughs in control synthesis will rely on a combination of theoretical and algorithmic advances. These breakthroughs will span a variety of areas from more efficient algorithms, to revisiting fundamental assumptions about how a constructed controller will interact with other components in a cyber-physical system. The following areas are potential directions for future research.

Efficient Algorithms for Symbolic Reasoning

A flexible framework for control synthesis ensures that future modifications and intuition can be incorporated into existing workflows. The framework served as a powerful layer of abstraction to encode synthesis algorithms. Thus far, the burden of designing these algorithms at this layer fell on the user who could declare that the dynamics exhibit a specific structural property. An alternative approach instead tasks the synthesizer with automatically identifying structure in the control synthesis problem. For instance, the spectrum between purely monolithic and purely distributed controllers could be characterized by the amount

of information shared amongst the controllers. This value is not known a priori and could be learned while synthesizing the controller.

The relational interfaces framework is a layer that lies on top of software packages for manipulating and reasoning about predicates. This layer is domain agnostic in principle, but is dramatically affected by the choice of this package. The first half of this thesis used binary decision diagrams (BDDs) as an engine to perform quantifier elimination because they empirically exhibited the best performance. While BDDs have outperformed approaches that use multi-dimensional arrays, they still encounter memory bottlenecks when representing higher dimensional sets. Swapping BDDs for more recent quantified Boolean formula solvers such as `CADET` [73] could lead to substantial runtime and memory improvements.

Stochastic Models and Abstractions

Deploying robust control synthesis in the real world will require revisiting the foundations and assumptions of robust control synthesis, which requires having a reasonable model of an adversarial environment's strength. Mismatches between the environment model and the true real-world environment leads to two undesirable outcomes. When using too benign an environment model the synthesizer result is overly optimistic about its capabilities; when using too adversarial an environment model the resulting controller is conservative or may be non-existent.

Balancing these two can be achieved via a more data-driven approach for calibrating the environment model. Rather than encoding uncertainty about the environment as a non-deterministic adversary, it can instead be represented as a stochastic entity. This approach "smooths" the environmental model and circumvents the conservatism that naturally arises from robust analysis and synthesis. Concepts from measure, information, and coding theory will likely be useful for new definitions of stochastic refinement.

Beyond Zero-Sum Games

A core assumption in robust control is that the environment is adversarial and can react to controller actions in its quest to violate the specification. The "environment" could consist of many different entities with different goals and capabilities.

There is a spectrum between adversarial and cooperative agents and more often than not they lie somewhere between these two extremes. For instance, some agents may intend to be cooperative but be unable to precisely execute on that intention. Others can have their own goals and cooperate the controller's goals as long as they do not conflict with their own. Most agents can only allocate attention and computational resources to a small set of tasks, or have incomplete information about the environment current state. Constructing controllers that operate in the real world reliable will require breaking free from zero-sum games to a broader game theoretic framework that accommodates agents with more complex interactions, incentives, and capabilities.

Synthesizing Controllers with Architectural Constraints

The real world imposes many auxiliary constraints on controller design. While it may be feasible mathematically to synthesize a controller, issues such as actuation and sensing delays and communication constraints could invalidate the synthesized controller's guarantees. Novel control architectures could accommodate these constraints and mitigate shortcomings. We've seen instances of centralized and distributed controllers. Other potential architectures include imposing a priority ordering for sequential decision making or have control inputs chosen over varying time horizons.

Blending Offline and Online Guarantees

Online and offline guarantees play distinct roles in designing autonomous systems. Computing guarantees offline benefits from additional computational resources and ideally is used to compute system invariants that are true over long (or possibly infinite) time horizons. These are especially useful for autonomous system designers who want certificates about control system behavior before deploying them. The offline approach comes at the cost of requiring an accurate generative model of the system environment beforehand to compute long horizon guarantees. Guarantees computed online are tighter because they have much richer information about the environment. These are typically only valid over a short time horizon, but can be used for online planning and runtime verification. Constructing introspective systems that can understand their own capabilities and deficiencies will require bridging the gap between offline and online guarantees.

Bibliography

[1] https://github.com/ericskim/redax/tree/master.

[2] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theoretical computer science* 126.2 (1994), pp. 183–235.

[3] Rajeev Alur et al. "Discrete abstractions of hybrid systems". In: *Proceedings of the IEEE* 88.7 (2000), pp. 971–984.

[4] Yashwanth Annpureddy et al. "S-taliro: A tool for temporal logic falsification for hybrid systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2011, pp. 254–257.

[5] C. Baier and J. P. Katoen. *Principles of model checking*. The MIT Press, 2008.

[6] Somil Bansal et al. "Goal-driven dynamics learning via bayesian optimization". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 5168–5173.

[7] Tamer Basar and Geert Jan Olsder. *Dynamic noncooperative game theory*. Vol. 23. Siam, 1999.

[8] Albert Benveniste et al. *Contracts for System Design*. Research Report RR-8147. INRIA, Nov. 2012, p. 65. URL: https://hal.inria.fr/hal-00757488.

[9] Dimitri P. Bertsekas. *Dynamic Programming and Optimal control*. Vol. 1. Belmont, MA: Athena Scientific, 1995.

[10] Roderick Bloem et al. "How to handle assumptions in synthesis". In: *arXiv preprint arXiv:1407.5395* (2014).

[11] Florence Boillot, Sophie Midenet, and Jean-Claude Claude Pierrelée. "The real-time urban traffic control system CRONOS: Algorithm and experiments". In: *Transportation Research Part C: Emerging Technologies* 14.1 (2006), pp. 18–38. ISSN: 0968090X. DOI: 10.1016/j.trc.2006.05.001.

[12] Greg Brockman et al. "OpenAI gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[13] Florian Brunner et al. "Communication Scheduling in Robust Self-Triggered MPC for Linear Discrete-Time Systems". In: *IFAC-PapersOnLine* 48.22 (2015), pp. 132–137. DOI: 10.1016/j.ifacol.2015.10.319.

[14] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35.8 (1986), pp. 677–691.

[15] Randal E. Bryant. "Symbolic Boolean manipulation with ordered binary-decision diagrams". In: *ACM Computing Surveys (CSUR)* 24.3 (1992), pp. 293–318. DOI: 10.1145/136035.136043.

[16] Oscar Lindvall Bulancea, Petter Nilsson, and Necmiye Ozay. "Nonuniform abstractions, refinement and controller synthesis with novel BDD encodings". In: *CoRR* (). arXiv: 1804.04280. URL: http://arxiv.org/abs/1804.04280.

[17] Krishnendu Chatterjee and Thomas A. Henzinger. "Assume-guarantee synthesis". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2007, pp. 261–275.

[18] Mo Chen et al. "A general system decomposition method for computing reachable sets and tubes". In: *IEEE Transactions on Automatic Control* (2018). DOI: 10.1109/TAC.2018.2797194.

[19] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. 1999.

[20] Giacomo Como, Enrico Lovisari, and Ketan Savla. "Convexity and Robustness of Dynamic Network Traffic Assignment for Control of Freeway Networks". In: *IFAC-PapersOnLine* 49.3 (2016), pp. 335–340.

[21] Samuel Coogan and Murat Arcak. "A Benchmark Problem in Transportation Networks". In: *CoRR* abs/1803.00367 (2018). arXiv: 1803.00367. URL: http://arxiv.org/abs/1803.00367.

[22] Samuel Coogan and Murat Arcak. "Efficient finite abstraction of mixed monotone systems". In: *Proceedings of 18th International Conference on Hybrid Systems: Computation and Control*. 2015, pp. 58–67.

[23] Samuel Coogan et al. "Traffic network control from temporal logic specifications". In: *IEEE Transactions on Control of Network Systems* 3.2 (2016), pp. 162–172.

[24] Carlos F. Daganzo. "The cell transmission model: A Dynamic Representation of Highway Traffic Consistent with the Hydrodynamic Theory". In: *Transportation Research* 28 (4 1994), pp. 269–287.

[25] Eric Dallal and Paulo Tabuada. "Decomposing Controller Synthesis for Safety Specifications". In: *2016 55th IEEE Conference on Decision and Control (CDC)*. 2016. DOI: 10.1109/CDC.2016.7799148.

[26] Eric Dallal and Paulo Tabuada. "On compositional symbolic controller synthesis inspired by small-gain theorems". In: *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE. 2015, pp. 6133–6138.

[27] Brian A Davey and Hilary A Priestley. *Introduction to Lattices and Order*. 2nd. Cambridge University Press.

[28] Charles A. Desoer and Mathukumalli Vidyasagar. *Feedback systems: input-output properties*. Vol. 55. Siam, 2009.

[29] Alexandre Donzé. "Breach, a toolbox for verification and parameter synthesis of hybrid systems". In: *International Conference on Computer Aided Verification*. Springer. 2010, pp. 167–170.

[30] Ioannis Filippidis, Dimos V. Dimarogonas, and Kostas J. Kyriakopoulos. "Decentralized multi-agent control from local LTL specifications". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 6235–6240.

[31] Ioannis Filippidis and Richard M. Murray. "Symbolic construction of GR(1) contracts for synchronous systems with full information". In: *CoRR* abs/1508.02705 (2015). arXiv: 1508.02705. URL: http://arxiv.org/abs/1508.02705.

[32] Jaime F. Fisac et al. "Reach-avoid Problems with Time-varying Dynamics, Targets and Constraints". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. HSCC '15. ACM, 2015, pp. 11–20. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728612. URL: http://doi.acm.org/10.1145/2728606.2728612.

[33] Antoine Girard and George J. Pappas. "Approximation metrics for discrete and continuous systems". In: *IEEE Transactions on Automatic Control* 52.5 (2007), pp. 782–798.

[34] Antoine Girard and George J. Pappas. "Hierarchical control system design using approximate simulation". In: *Automatica* 45.2 (2009), pp. 566 –571. ISSN: 0005-1098. DOI: http://dx.doi.org/10.1016/j.automatica.2008.09.016. URL: http://www.sciencedirect.com/science/article/pii/S0005109808004731.

[35] Antoine Girard, Giordano Pola, and Paulo Tabuada. "Approximately bisimilar symbolic models for incrementally stable switched systems". In: *IEEE Transactions on Automatic Control* 55.1 (2010), pp. 116–126.

[36] Gabriel Gomes and Roberto Horowitz. "Optimal freeway ramp metering using the asymmetric cell transmission model". In: *Transportation Research Part C: Emerging Technologies* 14.4 (2006), pp. 244–262.

[37] T.M.P. Gommans and W.P.M.H. Heemels. "Resource-aware MPC for constrained nonlinear systems: A self-triggered control approach". In: *Systems and Control Letters* 79 (2015), pp. 59 –67. ISSN: 0167-6911. DOI: 10.1016/j.sysconle.2015.03.003. URL: http://www.sciencedirect.com/science/article/pii/S0167691115000481.

[38] Felix Gruber, Eric S. Kim, and Murat Arcak. "Sparsity-Aware Finite Abstraction". In: *2017 IEEE 56th Conference on Decision and Control (CDC)*. 2017.

[39] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2016. URL: http://www.gurobi.com.

[40] WPMH Heemels, Karl Henrik Johansson, and Paulo Tabuada. "An introduction to event-triggered and self-triggered control". In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE. 2012, pp. 3270–3285. DOI: 10.1109/CDC.2012.6425820.

[41] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. "You assume, we guarantee: Methodology and case studies". In: *International Conference on Computer Aided Verification*. Springer. 1998, pp. 440–451.

[42] Thomas A. Henzinger et al. "What's decidable about hybrid automata?" In: *Journal of computer and system sciences* 57.1 (1998), pp. 94–124.

[43] Sylvia L. Herbert et al. "FaSTrack: A modular framework for fast and guaranteed safe motion planning". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 1517–1522.

[44] Kyle Hsu et al. "Multi-Layered Abstraction-Based Controller Synthesis for Continuous-Time Systems". In: *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*. HSCC '18. Porto, Portugal: ACM, 2018, pp. 120–129. ISBN: 978-1-4503-5642-8. DOI: 10.1145/3178126.3178143. URL: http://doi.acm.org/10.1145/3178126.3178143.

[45] PB Hunt et al. *SCOOT-a traffic responsive method of coordinating signals*. Tech. rep. Transport and Road Research Laboratory, 1981.

[46] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[47] Xiaoqing Jin et al. "Mining requirements from closed-loop control models". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.11 (2015), pp. 1704–1717.

[48] Mahmoud Khaled et al. "Synthesis of Symbolic Controllers: A Parallelized and Sparsity-Aware Approach". In: *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2019.

[49] Eric S. Kim, Murat Arcak, and Sanjit A. Seshia. "Compositional controller synthesis for vehicular traffic networks". In: *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*. IEEE. 2015, pp. 6165–6171.

[50] Eric S. Kim, Murat Arcak, and Sanjit A. Seshia. "Directed Specifications and Assumption Mining for Monotone Dynamical Systems". In: *19th International Conference on Hybrid Systems: Computation and Control*. ACM. 2016, pp. 21–30.

[51] Marius Kloetzer and Calin Belta. "A fully automated framework for control of linear systems from temporal logic specifications". In: *IEEE Transactions on Automatic Control* 53.1 (2008), pp. 287–297.

[52] Ron Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-time systems* 2.4 (1990), pp. 255–299.

[53] Alex Kurzhanskiy and Pravin Varaiya. "CTMSIM—An interactive macroscopic freeway traffic simulator". In: *Dept. of Electrical Engineering and Computer Sciences, Berkeley, CA., USA* (2008).

[54] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. "Mining assumptions for synthesis". In: *Proceedings of the Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign*. IEEE Computer Society. 2011, pp. 43–50.

[55] Yinan Li and Jun Liu. "ROCS: A Robustly Complete Control Synthesis Tool for Nonlinear Dynamical Systems". In: *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*. HSCC '18. Porto, Portugal: ACM, 2018, pp. 130–135. ISBN: 978-1-4503-5642-8. DOI: 10.1145/3178126.3178153. URL: http://doi.acm.org/10.1145/3178126.3178153.

[56] Shu Lin et al. "Model Predictive Control for urban traffic networks via MILP". In: *American Control Conference (ACC), 2010* 12.3 (2010), pp. 846–856. ISSN: 0743-1619.

[57] Johan Lofberg. "YALMIP: A toolbox for modeling and optimization in MATLAB". In: *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*. IEEE. 2004, pp. 284–289.

[58] John Maidens and Murat Arcak. "Reachability analysis of nonlinear systems using matrix measures". In: *IEEE Transactions on Automatic Control* 60.1 (2015), pp. 265–270.

[59] David Q. Mayne and Saša V. Raković. "Model predictive control of constrained piecewise affine discrete-time systems". In: *International Journal of Robust and Nonlinear Control* 13.3-4 (2003), pp. 261–279.

[60] Manuel Mazo, Anna Davitian, and Paulo Tabuada. "Pessoa: A tool for embedded controller synthesis". In: *International Conference on Computer Aided Verification*. Springer. 2010, pp. 566–569.

[61] Pierre-Jean Meyer, Antoine Girard, and Emmanuel Witrant. "Compositional abstraction and safety synthesis using overlapping symbolic models". In: *IEEE Transactions on Automatic Control* (2017). DOI: 10.1109/TAC.2017.2753039.

[62] Ian M. Mitchell and Jeremy A. Templeton. "A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2005, pp. 480–494.

[63] Thomas Moor and Jörg Raisch. "Abstraction based supervisory controller synthesis for high order monotone continuous systems". In: *Modelling, Analysis, and Design of Hybrid Systems*. Springer, 2002, pp. 247–265.

[64] Petter Nilsson and Necmiye Ozay. "Synthesis of separable controlled invariant sets for modular local control design". In: *American Control Conference (ACC), 2016*. IEEE. 2016, pp. 5656–5663.

[65] Petter Nilsson, Necmiye Ozay, and Jun Liu. "Augmented finite transition systems as abstractions for control synthesis". In: *Discrete Event Dynamic Systems* 27.2 (2017), pp. 301–340.

[66] Pierluigi Nuzzo. "Compositional Design of Cyber-Physical Systems Using Contracts". PhD thesis. EECS Department, University of California, Berkeley, 2015. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-189.html.

[67] Pierluigi Nuzzo et al. "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems". In: *Proceedings of the IEEE* 103.11 (2015), pp. 2104–2132.

[68] Pierluigi Nuzzo et al. "Are interface theories equivalent to contract theories?" In: *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*. IEEE. 2014, pp. 104–113.

[69] Markos Papageorgiou et al. "Review of road traffic control strategies". In: *Proceedings of the IEEE* 91.12 (2003), pp. 2043–2067. ISSN: 0018-9219. DOI: 10.1109/JPROC.2003.819610.

[70] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of reactive (1) designs". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer. 2006, pp. 364–380.

[71] Amir Pnueli. "The temporal logic of programs". In: *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE. 1977, pp. 46–57.

[72] Giordano Pola, Antoine Girard, and Paulo Tabuada. "Approximately bisimilar symbolic models for nonlinear control systems". In: *Automatica* 44.10 (2008), pp. 2508–2516.

[73] Markus N Rabe and Sanjit A Seshia. "Incremental determinization". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2016, pp. 375–392.

[74] Vasumathi Raman et al. "Model predictive control with signal temporal logic specifications". In: *Decision and Control (CDC), 2014 IEEE Conference on*. IEEE. 2014, pp. 81–87.

[75] Vasumathi Raman et al. "Reactive synthesis from signal temporal logic specifications". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM. 2015, pp. 239–248.

[76] Gunther Reissig, Alexander Weber, and Matthias Rungger. "Feedback Refinement Relations for the Synthesis of Symbolic Controllers". In: *IEEE Transactions on Automatic Control* 62.4 (April 2017), pp. 1781–1796.

[77] Matthias Rungger and Majid Zamani. "Compositional Construction of Approximate Abstractions of Interconnected Control Systems". In: *IEEE Transactions on Control of Network Systems* PP.99 (2016), pp. 1–1. ISSN: 2325-5870. DOI: 10.1109/TCNS.2016.2583063.

[78] Matthias Rungger and Majid Zamani. "SCOTS: A Tool for the Synthesis of Symbolic Controllers". In: *19th International Conference on Hybrid Systems: Computation and Control*. ACM, 2016, pp. 99–104.

[79] Sadra Sadraddini and Calin Belta. "A provably correct mpc approach to safety control of urban traffic networks". In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 1679–1684.

[80] Sadra Sadraddini and Calin Belta. "Model predictive control of urban traffic networks with temporal logic constraints". In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 881–881.

[81] Sadra Sadraddini and Calin Belta. "Safety control of monotone systems with bounded uncertainties". In: *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*. 2016, pp. 4874–4879. DOI: 10.1109/CDC.2016.7799014.

[82] Sadra Sadraddini, János Rudan, and Calin Belta. "Formal synthesis of distributed optimal traffic control policies". In: *Cyber-Physical Systems (ICCPS), 2017 ACM/IEEE 8th International Conference on*. IEEE. 2017, pp. 15–24.

[83] Stanley W. Smith, Petter Nilsson, and Necmiye Ozay. "Interdependence quantification for compositional control synthesis with an application in vehicle safety systems". In: *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE. 2016, pp. 5700–5707.

[84] Fabio Somenzi. *CUDD: CU Decision Diagram Package*. http://vlsi.colorado.edu/~fabio/CUDD/. Version 3.0.0. 2015.

[85] Dongyan Su et al. "Coordinated ramp metering and intersection signal control". In: *International Journal of Transportation Science and Technology* 3.2 (2014), pp. 179–192.

[86] Paulo Tabuada. *Verification and Control of Hybrid Systems*. New York, NY,USA: Springer, 2009.

[87] Paulo Tabuada et al. "Towards Robustness for Cyber-Physical Systems". In: *IEEE Transactions on Automatic Control* 59.12 (2014), pp. 3151–3163. ISSN: 0018-9286. DOI: 10.1109/TAC.2014.2351632.

[88] Danielle C Tarraf, Alexandre Megretski, and Munther A Dahleh. "A Framework for Robust Stability of Systems Over Finite Alphabets". In: *IEEE Transactions on Automatic Control* 53.5 (2008), pp. 1133–1146. ISSN: 0018-9286. DOI: 10.1109/TAC.2008.923658.

[89] Alfred Tarski. "A lattice-theoretical fixpoint theorem and its applications". In: *Pacific journal of Mathematics* 5.2 (1955), pp. 285–309. DOI: 10.2140/pjm.1955.5.285.

[90] Stavros Tripakis et al. "A theory of synchronous relational interfaces". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33.4 (2011), p. 14.

[91] Eric M. Wolff and Richard M. Murray. "Optimal control of nonlinear systems with temporal logic specifications". In: *Robotics Research*. Springer, 2016, pp. 21–37.

[92] Tichakorn Wongpiromsarn et al. "Distributed traffic signal control for maximum network throughput". In: *2012 15th international IEEE conference on intelligent transportation systems*. IEEE. 2012, pp. 588–595.

[93] Tichakorn Wongpiromsarn et al. "TuLiP: a software toolbox for receding horizon temporal logic planning". In: *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM. 2011, pp. 313–314.

[94] Majid Zamani et al. "Symbolic control of stochastic systems via approximately bisimilar finite abstractions". In: *IEEE Transactions on Automatic Control* 59.12 (2014), pp. 3135–3150.

[95] Majid Zamani et al. "Symbolic models for nonlinear control systems without stability assumptions". In: *IEEE Transaction on Automatic Control* 57.7 (2012), pp. 1804–1809.

Chapter 10

Appendix

10.1 Temporal Logic

Temporal logic [71] is a powerful formalism to encode complex timing requirements. It augments the standard Boolean logic connectives with temporal operators, allowing one to incorporate timing dependencies into statements about signals. Temporal logic has been used as a specification language for controller synthesis and verification of cyber-physical systems.

Linear temporal logic (LTL) is a specification language for discrete time signals [5]. An LTL formula encodes a set of infinite length signals. Predicates are encoded as statements that are true at a specific instant in time. A collection of temporal operators allows one to make statements with temporal constraints.

Definition 35 (Linear Temporal Logic)**.** *Linear temporal logic formulas are constructed with the syntax below*

$$\phi = \top \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi_1 \mathbf{U} \phi_2$$

where p is a predicate whose satisfaction is determined at each time step. The semantics of the four temporal operators $\mathbf{X}, \mathbf{F}, \mathbf{G}$, and $\mathbf{U}$ are summarized below:

- *Specification $\mathbf{X}\phi$ with the "next" operator $\mathbf{X}$ is true if and only if ϕ is true at the next time step.*
- *Specification $\mathbf{F}\phi$ with the "eventually" operator $\mathbf{F}$ is true if and only if ϕ is true at the current time step or there exists a future time step when ϕ is true.*
- *Specification $\mathbf{G}\phi$ with the "always" operator $\mathbf{G}$ is true if and only if ϕ is true at the present and all future time steps.*
- *Specification $\phi_1 \mathbf{U} \phi_2$ with the "until" operator $\mathbf{U}$ is true if and only if ϕ_2 is eventually true and ϕ_1 is true for all future time instances before ϕ_2 becomes true.*

A proposition is a statement that is either true or false at a given time step, such as "the number of vehicles in link 1 is less than 20" and can be interpreted as a subset of the state-input space $\mathcal{X} \times \mathcal{U}$. Boolean operators $\neg$ (negation), $\wedge$ (intersection/conjunction), and $\vee$ (union/disjunction) are used to make new statements.

Metric temporal logic (MTL) [52] is another variant of temporal logic where the temporal operators temporal operators associated with a given time interval. It has been applied to both discrete time and continuous time signals.

Definition 36 (Metric Temporal Logic)**.** *Metric temporal logic formulas are constructed with the syntax below*

$$\phi = \top \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathbf{F}_{[a,b)}\phi \mid \mathbf{G}_{[a,b)}\phi \mid \phi_1 \mathbf{U}_{[a,b)}\phi_2 \tag{10.1}$$

where p is a predicate and $[a, b)$ represents a continuous or discrete time interval. The semantics of the three temporal operators $\mathbf{F}_{[a,b)}, \mathbf{G}_{[a,b)}$, and $\mathbf{U}_{[a,b)}$ are summarized below:

- $\mathbf{G}_{[a,b)}\phi$*: Specification ϕ holds for all times $t \in [a, b)$*
- $\mathbf{F}_{[a,b)}\phi$*: Specification ϕ holds for some times $t \in [a, b)$*
- $\phi_1 \mathbf{U}_{[a,b)}\phi_2$*: Specification ϕ_2 holds at some time $t \in [a, b)$ and ϕ_1 holds for all time $[a, t)$.*

Because metric temporal logic was designed to accommodate continuous signals, it lacks the next operator $\mathbf{X}$ from LTL because no notion of "next time step" exists. When applied to discrete time signals, the next operator can still be mimicked with either $\mathbf{G}_{[1,2)}$ or $\mathbf{F}_{[1,2)}$ because the interval $[1, 2)$ represents a subsequent time step. Specifications can be composed with one another to express more complex requirements. For instance, $\mathbf{G}_{[0,5)}\mathbf{F}_{[0,3)}\psi \leq 1$ is a recurrence property where for every $t \in [0, 5)$, ψ is satisfied at time $t, t + 1$ or $t + 2$.

10.2 Ordered Relations

Binary relations are used to compare two elements in a set. They recover the typical notions of equality and ordering when restricted to scalar variables, but generalize these notions for more complex spaces.

Partial Orders

Definition 37 (Partially Ordered Set)**.** *A partial order $(A, \preceq_A)$ is a set A equipped with a binary relation $\preceq_A$ that satisfies each of the following properties for all $a, b, c \in A$:*

1. *(Reflexivity) If $a \preceq_A a$*
2. *(Antisymmetry) If $a \preceq_A b$ and $b \preceq_A a$ then $a = b$.*
3. *(Transitivity) If $a \preceq_A b$ and $b \preceq_A c$ then $a \preceq_A c$.*

A partially ordered set $(L, \preceq_L)$ is called a complete lattice if every subset M of L has a least upper bound and a greatest lower bound denoted as $\top_L$ and $\bot_L$ respectively. A common instance of a partial order is the subset relation $\subseteq$ over the set of subsets of a given set. This order is also a complete lattice.

Monotone Functions

Monotone functions are maps that preserve order.

Definition 38 (Monotone Function)**.** *Let $(P, \preceq_P)$ and $(Q, \preceq_Q)$ be two partially ordered sets. A function $f : P \to Q$ is monotone if:*

$$p_1 \preceq_P p_2 \Rightarrow f(p_1) \preceq_Q f(p_2) \tag{10.2}$$

for all $p_1, p_2 \in P$.

Property 1 (Monotone Function Composition)**.** *Let $f : B \to C$ and $g : A \to B$ be two total and monotone functions. Then the composition $f \circ g : A \to C$ is also a monotone function with respect to the orders $(A, \preceq_A)$ and $(C, \preceq_C)$.*

Fixed Points of Monotone Functions

The Knaster-Tarski fixed point theorem guarantees that there exists a fixed point for a monotone function over a complete lattice.

Theorem 9 (Knaster-Tarski Fixed Point Theorem [27])**.** *Let $(L, \preceq_L)$ be a complete lattice and $f : L \to L$ be a monotone function. The set of all fixed points is a complete lattice.*

A corollary of the Knaster-Tarski fixed point theorem is that the map f contains a least and greatest fixed point over $(L \preceq_L)$ because complete lattices are non-empty [27].

Property 2 below signifies that the least and greatest fixed points of a monotone function over a finite domain can be computed in a finite number of steps [19]. This is useful in the context of control synthesis because the control predecessor operation is a monotone function. The finiteness constraint can be satisfied by abstracting a state space into a finite domain.

Property 2. *Let $\bot_L$ and $\top_L$ be the bottom and top element respectively of the finite complete lattice L. Then*

- $f^i(\bot_L) \preceq_L f^{i+1}(\bot_L)$ *and* $f^{i+1}(\top_L) \preceq_L f^i(\top_L)$.
- *There exists an integer i' such that for every $i \geq i'$*
$$f^{i'}(\bot_L) = f^i(\bot_L).$$
- *There exists an integer i' such that for every $i \geq i'$*
$$f^{i'}(\top_L) = f^i(\top_L).$$