Differential Refinement Logic

Sarah M. Loos

CMU-CS-15-144
February 2016

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
André Platzer, Chair
Bruce Krogh
Frank Pfenning
Dexter Kozen (Cornell University)
Stefan Mitsch (Johannes Kepler University Linz)
George Pappas (University of Pennsylvania)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Sarah M. Loos is supported by a Department of Energy Computation Sciences Graduate Fellowship and a National Science Foundation Graduate Research Fellowship.
This research was sponsored by the National Science Foundation under grant numbers CNS-0931985, CNS-1035800, CNS-1054246, CNS-0926181, and DGE-0750271. This research was also supported by the US Department of Transportation's University Transportation Center's TSET grant, award no. DTRT12GUTC11. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Differential Refinement Logic, Differential Dynamic Logic, Cyber-Physical Systems, Hybrid Systems, Distributed Hybrid Systems, Formal Verification, Formal Methods, Theorem Proving

For Jeremy

Abstract

This thesis is focused on formal verification of cyber-physical systems. Cyber-physical systems (CPSs), such as computer-controlled cars, airplanes or robots play an increasingly crucial role in our daily lives. They are systems that we bet our lives on, so they need to be safe. However, ensuring that CPSs are safe is an intellectual challenge due to their intricate interactions of complex control software with physical behavior. Formal verification techniques, such as theorem proving, can provide strong guarantees for these systems by returning proofs that safety is preserved throughout the continuously infinite space of their possible behaviors.

Previously completed work has provided: the first formal verification of distributed car control; the first formal verification of distributed, flyable, collision avoidance protocols for aircraft; and an exploration of control choices within a well-defined safety envelope. Each of these systems presented new verification challenges and required new techniques for proving safety. However, we identified a unifying hurdle for each case study that has thus far remained unaddressed: *it is difficult to compare hybrid systems, even when their behaviors are very similar*.

We introduce *differential refinement logic* (dRL), a logic with first-class support for refinement relations on hybrid systems, and a proof calculus for verifying such relations. The logic dRL simultaneously solves several seemingly different challenges common in theorem proving for hybrid systems:

1. When hybrid systems are complicated, it is useful to prove properties about simpler and related subsystems before tackling the system as a whole.
2. Some models of hybrid systems can be implementation-specific. Verification can be aided by abstracting the system down to the core components necessary for safety, but only if the relations between the abstraction and the original system can be guaranteed.
3. One approach to taming the complexities of hybrid systems is to start with a simplified version of the system, prove it safe, and then iteratively expand it. However, this approach can be costly, since every iteration has to be proved safe from scratch, unless refinement relations can be leveraged in the proof.
4. When proofs become large, it is difficult to maintain a modular or comprehensible proof structure. By using a refinement relation to arrange proofs hierarchically according to the structure of natural subsystems, we can increase the readability, modularity, and reusability of the resulting proof.

The logic dR$\mathcal{L}$ extends an existing specification and verification language for hybrid systems (differential dynamic logic, d$\mathcal{L}$) by adding a refinement relation to directly compare hybrid systems. This thesis gives a syntax, semantics, and proof calculus for dR$\mathcal{L}$.

We also demonstrate the usefulness of dR$\mathcal{L}$ on several examples which the author has previously completed. We show that using refinement results in easier and better-structured proofs, leveraging as first-class citizens in the proof the structure that has only been implicit in previous d$\mathcal{L}$ proofs.

Acknowledgments

There are many people I would like to thank. For many of them, had it not been for their support, encouragement or guidance, I would not have finished (or even started) my Ph.D, and I am eternally grateful to them.

First, I would like to thank my advisor, André Platzer. André is a brilliant researcher, a dedicated teacher, and a spectacular advisor. I have learned so much from him. I want to thank all of my fantastic committee members: Dexter Kozen, Bruce Krogh, Stefan Mitsch, George Pappas, and Frank Pfenning. When I was just starting out in CPS verification, Bruce Krogh was always there to give feedback from a controls theory perspective, keeping our verification approaches more closely tied to realistic systems. It was a pleasure to work with Bruce and his students and postdocs. Stefan has been a first-rate colleague and friend. His undefeatable positivity and determination always kept me going when I felt my problems were too tough to tackle. My interactions with everyone on the committee have been invaluable; thank you for all your advice and support.

I have had the pleasure of working with many wonderful coauthors and collaborators, from whom I've learned so much: Matthias Althoff, Khalil Ghorbal, João Martins, David Henriques, Jan David Quesel, Stefan Mitsch, Ran Ji, Nathan Fulton, Nikos Aréchiga, David Renshaw, Ligia Nistor, Yanni Kouskoulas, Ajinkya Bahave, Akshay Rajhans, Erik Zawadzki, Ivan Ruchkin, David Witmer, David Garlan, Peter Steenkiste, Annika Peterson, Colm Bhandal, and Grant Passmore.

I would also like to thank my undergraduate advisors. Suzanne Menzel, who had the foresight to tell me, "Don't worry, you *will* be a CS major." And David Wise, who gave me my first glimpse into research and the many highs that come from solving problems that have never been solved before.

Grad school is, in the wise words of Mor Harchol-Balter, a roller coaster of ups and downs. I was fortunate to have many friends and family who I leaned on for support throughout my time in grad school: Dana and Yair Movshovitz-Attias, Abigail Chappell, Sam Gottlieb, Patrick Xia, Ankit Sharma, John Wright, Karl Naden, Joe Blaylock and Anami Sheppard, Roger Wolff, Aaditya Ramdas, Kate Taralova, Akshay Krishnamurthy, Jayant Krishnamurthy, João Martins, Mary Wootters, Bruno Vavala, Gabe and Cat Weisz, Yuzi Nakamura, David Naylor, David Witmer, Li-Yang Tan, and Yu Zhao. I also want to thank Deb Cavlovich, Sammi DiNardo, and Catherine Copetas, who in addition to being fantastic people to visit for a chat, seemed to resolve the rare administrative or funding challenges instantaneously.

I thank all my family, Mom, Tryna, Abby, Peggy, Stan, Mark, Joann, Josh, and Nat for supporting my education all the way through the 22nd grade. I thank my dad for a plethora of lectures that have served me well in all aspects of my life.

And most of all, I thank Jeremy, who has been my most ardent supporter, even across the farthest distances. Jeremy, you are the song of hope in my soul that keeps me excited to see what every new day will bring. I can't wait to live this next adventure together.

Contents

Chapter 1

Introduction

1.1 Overview

Differential dynamic logic ($\mathsf{d}\mathcal{L}$) is an existing specification and verification logic for hybrid programs [1–4]. We have had many successes using $\mathsf{d}\mathcal{L}$ and its associated proof calculus in proving safety for complicated cyber-physical systems. Some examples of systems that we have modeled as hybrid programs in $\mathsf{d}\mathcal{L}$ and verified with its theorem prover KeYmaera include: distributed car control [5], distributed aircraft control [6], collision avoidance at intersections [7], and designing safe PID controllers for adaptive cruise control [8, 9].

While these examples have been impressive in their own right, we faced many barriers while exploring these case studies and often our verification results relied on complicated and creative proofs [5, 6]. Even more frustrating were the occasions where it was too challenging to verify a system as-is, and so we instead verified a simpler variant and then relied on informal reasoning to relate results back to the original system [7–9]. *We observed that if only we had direct proof support for systematically relating two systems, such proofs could be entirely formal and greatly simplified.*

For a simple illustration, consider an adaptive cruise controller tasked to set a car's acceleration and braking in a way that optimizes fuel consumption. Of course, even though it is designed for fuel consumption, the controller must also maintain a safe following distance behind other vehicles. Verifying safety for this controller means proving that this safe following distance is indeed maintained. But complicated optimizations designed for fuel efficiency can obfuscate the part of the code that guarantees safety. It would be easier to instead verify a simpler model of cruise control that is designed purely to satisfy the safety requirement. If this simpler model also provably contains all possible behaviors of the original fuel-efficient controller, then the fuel-efficient controller inherits the safety proof. However, in the formal syntax of $\mathsf{d}\mathcal{L}$, the possible behaviors of the two hybrid programs can not be directly compared without significant proving effort, often equal to proving the original system directly.

These and other observed challenges usually fit into one or more of the following categories:

- **Breaking the system into parts.** Some systems naturally decompose into smaller subsystems. We are able to leverage this, for example, when verifying safety for a highway of cars all operating under distributed controllers in [5]. A two-car system makes the basic

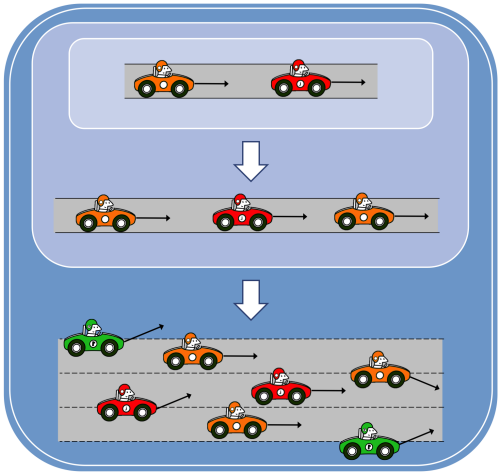

Figure 1.1: Breaking a large system into smaller building blocks was the key insight for verifying distributed car control in [5].

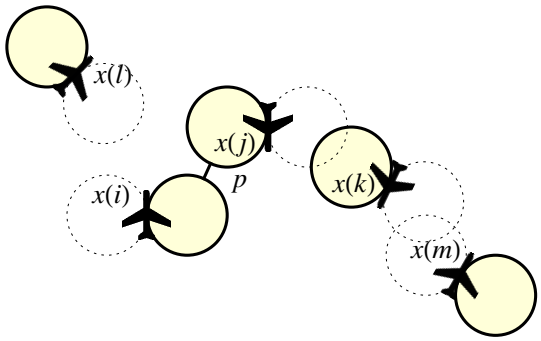

Figure 1.2: Maintaining a modular proof structure was the key insight for verifying this distributed aircraft protocol in [6].

building block for a lane of cars, then a lane of cars is the building block for the highway, as illustrated in Fig. 1.1. In this case, we lucked out that the system could in fact be decomposed in this way. On top of that, we were very careful to write the model in such a way that this decomposition would occur early in the proof, since the hybrid programs would need to be unrolled by proof rules until the point where they differ. This decomposition technique can only be effective in cases where the subsystem is nearly identical to the original hybrid program. And, even in cases where decomposition is possible, figuring out how to model the system to take advantage of this decomposition requires a considerable amount of foresight.

- **Abstracting implementation-specific designs.** When a hybrid program describes a specific system implementation which is not directly related to the safety property (e.g. a PID controller, or a fuel-efficient adaptive cruise controller), it is significantly harder to verify than an abstraction of the system which is only designed for the safety-critical aspects of the system. An over-approximation of the system may be easier to prove, but without a refinement relation in $\mathsf{d}\mathcal{L}$, relating the abstract model to the original model requires either a significant proving effort or must be reasoned about informally.
- **Leveraging iterative system design.** Due to the many subtleties of programming discrete events to control continuous physical systems, it is almost impossible to design a safe cyber-physical system on the first attempt. Even small cyber-physical systems are too hard to get right without careful study. To aid in system design, we almost always require an iterative approach [5–9]. We verify a simplified version of the system first in order to learn about it's behavior and design a correct controller. Then we can extend it to be closer to the original goal. With each iteration, we can localize where new bugs can occur and we can reapply the proof tricks we learned in the simplified system. Currently, using iterative system design requires restarting proofs from scratch with every increase in complexity. While iterative design is extremely useful and sometimes crucial for getting complicated

systems right, it is also very costly.

- **Maintaining a modular proof structure.** As proofs and systems become larger and more complex, there is a growing need for a formal way to inject more modularity into the proof structure. For example, the key to proving safe separation for an arbitrary number of aircraft in [6] is to first prove that each aircraft stays on the edge of its own disc and then prove that the moving discs stay safely separated, as illustrated in Fig. 1.2.

The insight in this thesis is that all of these seemingly different problems, from controlling the structure of large proofs to the high cost of iteratively designing safe systems, can be addressed with the same technique: providing direct proof support for relating two hybrid programs. By adding this technique to our toolbox, we can systematically address on a technical level what we have done before in an ad hoc way.

Several verification case studies in the fields of car control and aircraft control have provided us with a solid understanding of several major challenges in CPS verification (see Section 1.4). In that work, many of the challenges were addressed for the first time in theorem proving for hybrid systems, so our approaches were understandably tailored to each specific problem. Now that we are looking at our solutions retrospectively, there is a clear missing piece: how can we efficiently compare the behavior of a panoply of hybrid programs without losing the strong verification guarantees that come with a proof?

In this thesis we will introduce *differential refinement logic* (dR$\mathcal{L}$), an extension of d$\mathcal{L}$ which will allow for the direct comparison of hybrid programs. We accomplish this by adding a *refinement* relation to the grammar of d$\mathcal{L}$. We say that hybrid program α refines hybrid program β (written $\alpha \leq \beta$) if and only if all possible transitions in hybrid program α are also possible in β. Whenever a refinement relation has been proved between two hybrid programs, the more restrictive program automatically inherits all safety properties proved about the program that is more permissive. Additionally, any liveness properties proved about the more restrictive program will also be satisfied by the more permissive program. The thesis will introduce a syntax, semantics, and proof calculus for dR$\mathcal{L}$ (Chapter 3). In Table 1.1 we list the challenges we faced in each case study that may be addressed by dR$\mathcal{L}$.

We will test the usefulness of dR$\mathcal{L}$ by revisiting the case studies verified using d$\mathcal{L}$ and exploring where dR$\mathcal{L}$ simplifies and strengthens these proofs. They include:

- using verified safety envelopes to let a specific car controller inherit a proof of safety (Section 4.9),
- showing that we can now provide a formal relationship between event- and time-triggered systems (Section 6.3),
- re-verifying the distributed car control system using the dR$\mathcal{L}$ calculus using significantly fewer interactive steps (Section 6.4),
- and a discussion of a distributed aircraft protocol (Section 8.4).

1.2 Impact and Future Work

In addition to addressing many challenges we have already encountered in theorem proving for hybrid systems, dR$\mathcal{L}$ has the potential to make an impact in areas from proof search heuristics

to code generation and synthesis. We have observed that it is often significantly easier to verify models with implicit controllers. In fact, if an implicit controller is taken to an extreme where the requirements of the controller include a nested model of the physical behavior of the system, verification becomes trivial. We also hope to explore new proof search heuristics that leverage the refinement relation in a way that automatically enforces hierarchical proof structures, possibly resulting in more efficient proof search. Additionally, we believe that refinement could reduce some of the challenges faced by synthesis, as each step of the synthesis process could be checked that it is truly a refinement of the system that is verified to adhere to safety requirements.

1.3 Thesis Statement

The goal of this thesis is to provide a formal framework for comparing hybrid programs so that CPS verification can benefit from arguments reasoning explicitly about the relations of multiple hybrid systems. In support of this, we introduce differential refinement logic (dRℒ), develop a proof calculus for it, and demonstrate its usefulness in a variety of CPS domains.

1.4 Introduction to Relevant Case Studies

1.4.1 Distributed Car Control

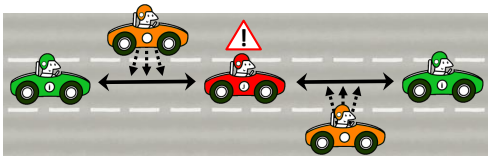

Figure 1.3: Multilane highway verified for an arbitrary number of cars. Cars may change lanes, and enter and exit the highway.

Car safety measures can be most effective when the cars on a street coordinate their control actions using distributed cooperative control. While each car optimizes its navigation planning locally to ensure the driver reaches his destination, all cars coordinate their actions in a distributed way in order to minimize the risk of safety hazards and collisions. These systems control the physical aspects of car movement using cyber technologies like local and remote sensor data and distributed vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication. They are thus cyber-physical systems.

In [10], we considered a distributed car control system inspired by the ambitions of the California PATH project, the CICAS system, SAFESPOT and PReVENT initiatives. We developed a formal model of a distributed car control system in which every car is controlled by adaptive cruise control. One of the major technical difficulties is that faithful models of distributed car control have both distributed systems and hybrid systems dynamics. They form distributed hybrid systems, which makes them very challenging for verification. In a formal proof system, we verify that the control model satisfies its main safety objective and guarantees collision freedom for arbitrarily many cars driving on a street, even if new cars enter the lane from on-ramps or multi-lane streets. The proof of safety for this system relies heavily on a hierarchical proof structure, with lemmas proving safety first for two cars on a single lane, then for an arbitrary number

Challenges found in applications that dRℒ addresses				
	break systems into parts	implementation-specific design	iterative system design	proof modularity
Distributed Car Control (Section 1.4.1)	✓	—	✓	✓
Smart Intersections (Section 1.4.1	—	—	✓	✓
Distributed Aircraft (Section 1.4.2)	—	—	✓	✓
PID Controller (Section 1.4.3)	—	✓	✓	—
Safety Envelopes (Section 1.4.3)	✓	✓	✓	—
Efficiency Analysis (Section 1.4.4)	—	—	✓	—

Table 1.1: Challenges encountered in previously completed case studies. dRℒ can address all of them with a single, unified approach.

of cars on a single lane, and finally for an arbitrary number of cars on an arbitrary number of lanes.

Our main contribution in [10] is that we develop a distributed car control system and a formal proof that this system is collision-free for arbitrarily many cars, even when new cars enter or leave a multi-lane highway with arbitrarily many lanes. Another contribution is that we develop a proof structure that is strictly modular. We reduce the proof to modular stages that can be verified without the details in lower levels of abstraction. Several of the principles behind this *modular proof structure* can be formalized using dRℒ and that such verification techniques are useful for other systems beyond the automotive domain. Further contributions are:

- This is the first case study in distributed hybrid systems to be verified with a generic and systematic verification approach that is not specific to the particular problem.
- We identify a simple invariant that all cars have to obey and show that it is sufficient for safety, even for emergent behavior of multiple distributed car maneuvers.
- We identify generic and static constraints on the input/output parameters that any controller must obey to ensure that cars always stay safe.
- We demonstrate the feasibility of distributed hybrid systems verification.

This case study benefits from *breaking the system into parts*. We will examine how using dR$\mathcal{L}$ might simplify and strengthen the proofs presented in [10].

1.4.2 Distributed Aircraft Control

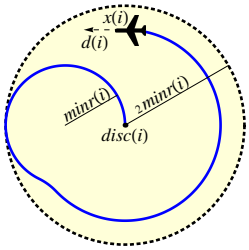

Figure 1.4: Verified *big disc* collision avoidance maneuver.

As airspace becomes ever more crowded, air traffic management must reduce both space and time between aircraft to increase throughput, making onboard collision avoidance systems ever more important. These safety-critical systems must be extremely reliable, and as such, many resources are invested into ensuring that the protocols they implement are accurate. Still, it is challenging to guarantee that such a controller works properly under every circumstance. In tough scenarios where a large number of aircraft must execute a collision avoidance maneuver, a human pilot under stress is not necessarily able to understand the complexity of the distributed system and may not take the right course, especially if actions must be taken quickly.

In this paper we specify and verify two control policies for planar aircraft avoidance maneuvers using the automated theorem prover KeYmaeraD to produce a proof of safety for each of them. We design these policies such that all aircraft adhere to a simple and easy-to-implement separation principle: associated with each aircraft is a disc, within which the aircraft must remain. By *breaking the system into parts* in this way, the problem reduces to proving that i) sufficient separation is maintained between pairs of discs, and ii) individual aircraft always remain inside their associated disc. We model 2D flight dynamics since they are the relevant dynamics for planar maneuvers, but investigating 3D maneuvers and dynamics may make interesting future work.

The complexities which arise from the curved flight trajectories of an arbitrary number of aircraft interacting in a distributed manner, along with the tight coupling of discrete control and continuous dynamics presently make KeYmaeraD the only verification tool capable of proving safety for this system. Our contributions are:

- We provide the first formally verified distributed system of aircraft with *curved flight* dynamics.
- Our controller requires only flyable aircraft trajectories with no corners or instantaneous changes of ground speed.
- We prove our controller is safe for an arbitrarily large number of aircraft. This guarantee is necessary for high-traffic applications such as crowded commercial airspace, unmanned aerial vehicle maneuvers, and robotic swarms.
- Other aircraft may enter an avoidance maneuver already in progress and safety for all aircraft is still guaranteed.
- We prove that even when the interactions of many aircraft cause unexpected emergent

behaviors, all resulting control choices are still safe.

- We present hierarchical and compositional techniques to reduce a very complex system into smaller, provable pieces.

In the design and verification process for this system, it was clear that *leveraging iterative system design* had the potential to substantially reduce the challenge of proving safety for the entire system, as we first proved the controller safe for two aircraft before moving to the distributed space. In Chapter 8, we investigate how a *modular proof structure* could be maintained within the formal framework of dR$\mathcal{L}$ by leveraging refinement.

1.4.3 Verifying Safety Envelopes, Implementing PID Controllers

In [9] we proposed an approach in which the verified *safety envelopes* were used as static conditions to be checked for specific controller designs, without requiring the inclusion of any physical dynamics of the system.

Theorem provers are often most efficient when using generic models that abstract away many of the controller details. These abstract models use very general conditions that can then be used as safety envelopes when designing more detailed controllers.

We demonstrated the use of safety envelopes using the KeYmaera theorem prover for d$\mathcal{L}$ for two examples: adaptive cruise control with two cars and a cooperative intersection collision avoidance system (CICAS) for left-turn assist. In each case, a proof of safety for the closed-loop system provided static safety envelopes that are then used informally to check the controller design.

In [8, 9], we used the concept of refinement to reason informally outside of d$\mathcal{L}$ that a deterministic PID implementation of a non-deterministic hybrid program can inherit the original safety proofs. Revisiting these projects, this time with dR$\mathcal{L}$, could allow us to *abstract implementation-specific designs*, like a PID controller, and provide a verification of the system which does not rely on meta analysis.

1.4.4 Symbolic Verification Allows Efficiency Analysis

In [11] we considered an adaptive cruise control system in which control decisions are made based on position and velocity information received from other vehicles via V2V wireless communication. If the vehicles follow each other at a close distance, they have better wireless reception but collisions may occur when a follower car does not receive notice about the decelerations of the leader car fast enough to react before it is too late. If the vehicles are farther apart, they would have a bigger safety margin, but the wireless communication drops out more often, so that the follower car no longer receives what the leader car is doing. In order to guarantee safety, such a system

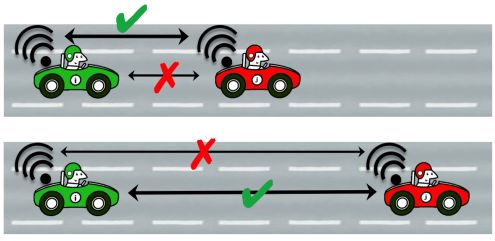

Figure 1.5: There is a tradeoff between signal strength and keeping a safe following distance with a higher tolerance for dropped packets.

must return control to the driver if it does not receive
an update from a nearby vehicle within some timeout period. The value of this timeout parameter encodes a tradeoff between the likelihood that an update is received and the maximum safe acceleration. Combining formal verification techniques for hybrid systems with a wireless communication model, we analyzed how the expected efficiency of a provably-safe adaptive cruise control system is affected by the value of this timeout.

Because this controller needed to be optimally permissive in order for the efficiency analysis to work out, the process of designing the controller required many iterative steps in order to get it to its final state. We will investigate whether using dR$\mathcal{L}$ could use the refinement relation to simplify this *iterative system design* process.

Chapter 2

Related Work

Differential refinement logic (dR$\mathcal{L}$) extends an existing specification and verification language for hybrid systems, *differential dynamic logic* (d$\mathcal{L}$), originally proposed by Platzer [3, 4]. The logic d$\mathcal{L}$ has had many successes in proving safety for challenging CPS. This thesis and dR$\mathcal{L}$ improve upon those successes by streamlining, extending, and formalizing existing d$\mathcal{L}$ proofs. Additionally, dR$\mathcal{L}$ enables users to prove properties that were previously out of scope for d$\mathcal{L}$ due to required user interactions being too challenging or numerous.

2.1 Refinement and d$\mathcal{L}$

One major challenge for formal verification of cyber-physical systems is that there will always be a gap between the behavior of a statically verified model of a system and the behavior of the physical implementation of the system "in the wild." This is because the many continuous environment and state variables can never be fully captured and precisely represented. While dR$\mathcal{L}$ reduces this gap by making more challenging models verifiable, it will not be able to fully verify a system against circumstances that are not explicitly represented in the model. What happens when unpredicted weather conditions or broken actuators cause our assumptions to be violated? Unfortunately, a proof in dR$\mathcal{L}$ can give no guarantees. However, ModelPlex (Mitsch et al. [12]) implements a run-time analysis that samples the observed state of a real system via sensors and checks in real time that this state refines the reachable states of the statically verified model. If this refinement relation is found to be violated, the monitor can then raise a warning. Determining such a refinement relation at run-time is often computationally challenging due to limited time and computing resources on cyber-physical systems, but by using quantifier elimination to statically handle many of the computational complexities in advance of run-time, this technique is in scope for several systems [12]. While ModelPlex does not allow for a notion of refinement directly in the language of formulas, it does use a semantic definition that is compatible with the semantics of refinement for dR$\mathcal{L}$. As a result, dR$\mathcal{L}$ and ModelPlex mutually benefit from one another, as it is reasonable to believe that many of the concepts introduced with ModelPlex can be translated into the formal framework of dR$\mathcal{L}$. This is a great way to bridge the final gap between real-world systems and models, but it will only work when the gap between the verified model and the running system is as small as possible, both to keep computations tractable and

to reduce the frequency of spurious warnings. Finally, for safety-critical systems, it is better to think of run-time monitoring as a last resort, as it may require extremely conservative controllers to be able to make safety guarantees, or catch problems too late due to discrete sampling of the state.

In [13], Mitsch et al. present proof-aware refactorings for hybrid programs in dℒ. By examining a number of common refactoring transformations on hybrid programs, they can make corresponding changes in their correctness proofs, thereby increasing proof reuse between verifications of similar hybrid programs. They introduce the semantic concept of a refinement relation over hybrid programs, which can then be leveraged in meta-level analysis to show that program transformations preserve safety or liveness properties. The analysis is limited to finitely many use cases. We share many goals with this research, and commend it as the first step toward defining refinement based on reachable states and increasing the flexibility of which systems can be considered refinements of each other. In this thesis, we hope to make complementary progress toward these goals by introducing a refinement relation formally into the grammar of dℝℒ. With an associated proof calculus, we hope to leverage syntactic automation to reduce the level of proof awareness required of the user. We also aim to decrease the user's reliance on meta analysis, which is more susceptible to human error. Additionally, we examine refinement within a given context, so $\alpha \leq \beta$ may hold under the current context while not holding generally. Mitsch et al. achieve this by introducing a formula F which can be used to describe the context in which the refinement holds, but by including the refinement directly in the logic, context is automatically defined and updated along various proof branches.

2.2 Refinement and Discrete Programs

Kleene algebra with tests is a system for manipulating programs that are equivalent [14]. Hybrid programs in dℒ form an idempotent semi-ring when you take sequential composition as the multiplicative operator, and nondeterministic choice as an additive operator. Adding in the Kleene star and the test gives us a Kleene algebra with tests (KAT). As a result, we draw heavily on research done on these algebras when designing the corresponding proof calculus for dℝℒ. For example, the proof rules presented in Fig. 3.2 are derived directly from KAT axioms. But this is not the end of the story for dℝℒ, as we still have to handle the complexities of assignments and differential equations. Even simple hybrid programs that would seem trivially unrelated when examined through the lens of KAT, may in fact satisfy the refinement relation. For example, consider the hybrid programs $x := 10$ and $x := 1; x' = x$. The program $x := 10$ simply assigns the value 10 to variable x. The program $x := 1; x' = x$ first sets x to 1 and then follows the solution to the differential equation $x' = x$ for a nondeterministic amount of time. While these hybrid programs appear unrelated, the first hybrid program is actually a refinement of the second. The continuous evolution that follows $x' = x$ evolves for a nondeterministic period of time, thus allowing x to take any value greater than 1, which means it includes a transition where x takes the value 10.

Moreover in formal methods for cyber-physical systems, it is critical to express a refinement relationship between two programs, since we are always trying to refine the programs we verify into programs that more closely represent the real conditions in which they operate. While

KAT has rules for handling refinement behaviors, its focus is on manipulating programs that are equivalent. Specifically, a refinement $a \leq b$ in KAT is rewritten in terms of an equivalence relation as $a \cup b = b$. As a result, the left-hand-side program becomes bigger and possibly more complicated. More importantly, we may lose the ability to make refinement arguments that fall out more easily from structural similarities that commonly occur between a and b.

We strive to make use of the advantages of dℒ and KAT by combining and extending them into dRℒ. Many of the most useful proof rules in dRℒ could be expressed as derived proof rules from KAT, just as all proof rules in dRℒ could be expressed as derived proof rules from dℒ. However, the main contributions of differential refinement logic to the area of theorem proving for hybrid systems are not in a provable increase in proving power, but rather a different stylistic approach to theorem proving. Differential refinement logic focuses on enabling users to iteratively design and verify increasingly complicated hybrid systems, generate proofs of safety that are modular, and soundly abstract away implementation-specific design elements that aren't crucial to establishing safety of the system. Each of these goals are achieved through a focus on the interplay of dℒ's modalities for proving propeties of system dynamics with refinement relations on programs, as opposed to equivalent program transformations.

Because KAT and dRℒ fundamentally have so much in common, we expect many results from research related to KAT to be relevant to dRℒ. For example, a big area of future work for this research lies in developing decision procedures for dRℒ. In such future work, we will further examine work on bisimulation-based decision procedures for NetKAT, a logic for reasoning about packet switching networks [15, 16].

The Z notation [17] provides a formal specification frequently used for specifying discrete software behaviors. Much research has been done based on Z notation [18], including the development and use of the refinement calculus for Z [19]. Such approaches look primarily at how to eliminate nondeterminism in each component of the specification. While this thesis also spends a lot of time looking at refinement as a tool for removing nondeterminism and thereby verifying models that are more implementable, this isn't the end of the story for dRℒ. Ultimately, dRℒ defines refinement according to reachability of states. This means that much more than just removing nondeterminism is possible. Refinement relations may be proved between two programs which may look very different structurally, but still satisfy the refinement relationship by virtue of one program having a subset of the reachable states of the other.

2.3 Refinement and Hybrid Systems

While the refinement relation as a first-class member of dRℒ is new, the concept of refinement for cyber-physical systems has been in use for quite some time. Model checking, which often suffers from statespace explosion, has seen tremendous success in using abstraction to keep the statespace small, then iteratively refining the model to exclude spurious counter examples (CEGAR [20]).

Event-B [21] and its verification tool Rodin [22] has had success defining refinement based on trace inclusion [23], which is more restrictive than our approach of comparing reachable states. However, Event-B focuses on refinement as a crucial step in the development process for discrete systems, as it is important to ensure correctness at every level of development. This is

also a core motivation for dRℒ, as we found it challenging to leverage iterative system design for hybrid systems using dℒ without refinement. The dRℒ logic also handles interactions between modalities and refinements. This means that, in addition to proving two systems satisfy a refinement relation, we can also prove properties of a given system. Through combining refinement relations and modalities, systems can inherit properties proved about the systems they refine, thus providing concrete guarantees about their reachable states.

Recent work by Banach et al. introduces Hybrid Event-B [24], which adds continuous variables with continuous evolution of those variables over time intervals to the Event-B framework (though tool support in Rodin has yet to be extended to explicitly include hybrid systems). Butler et al. define a restricted notion of refinement for Hybrid Event-B [25] which has a two-pronged approach: reducing nondeterminism in the continuous evolution of the system and adding additional discrete actions to a model. Differential refinement logic is general enough to handle both of these special cases. We use dRℒ to prove refinement when additional discrete actions are added to an event-triggered system to transform it into a time-triggered system. In [25], continuous evolution is not explicitly modeled using differential equations, but is rather abstracted as assumptions about the continuous behavior of the system. This is conceptually similar to defining a differential invariant, but without any provable link between the differential invariant and the differential equations that govern the continuous dynamics of the system. In dRℒ, we use differential equations to more accurately model the continuous dynamics of the system. We employ differential invariants, differential cuts, and differential refinement techniques to handle the complex behaviors that are modeled explicitly in the differential equations. Using dRℒ and its proof calculus, we are able to prove that reducing nondeterminism is a refinement, not just in the purely continuous evolutions, but also in discrete and hybrid transitions.

Approximate bisimulation is another approach for relating two models of hybrid systems [26, 27]. Unlike exact bisimulation, which requires two systems to be identical under a mapping between them, approximate bisimulation only requires that the systems be close. A bound can be calculated for the maximal error for the approximate system. Then, by adding a buffer of at least that error to the boundary of the unsafe region, proving safety of the approximate system against the buffered region guarantees safety of the original system as well. In contrast, dRℒ's working principle is compositional by allowing several local reasoning steps about parts of a system to support a refinement argument. It also gives a general way of combining behavioral and refinement arguments.

Applying some of the underlying concepts of approximate bisimulation to dRℒ could result in an interesting extension where refinement is defined as an approximate approximate relation, allowing state variables to be fuzzed by some bounded margin of error. While this thesis does not investigate this extension to dRℒ, it may be a useful approach, particularly in the cases where a useful differential invariant can not be identified.

2.4 Car Control

Because of its societal relevance, numerous aspects of car control have been previously studied [28–45]. Major initiatives have been devoted to developing next generation individual ground transportation solutions, including the California PATH project, the SAFESPOT and PReVENT

initiatives, the CICAS-V system, and many others. The societal relevance of vehicle cooperation for CICAS intersection collision avoidance [38] and for automated highway systems [32, 35] has been emphasized. Horowitz et al. [37] proposed a lane change maneuver within platoons. Varaiya [40] outlines the key features of an IVHS (Intelligent Vehicle/Highway System). A significant amount of work has been done in the pioneering California PATH Project.

Our interest in car control comes first from the challenges that arise when a large system of cars is modeled as a distributed system, with safety properties (e.g. ensuring there are no collisions) that must be guaranteed globally. The distributed nature of car control problems lend them to being broken down into subsystems for the purpose of verification. Secondly, many car control systems are implementation-specific: they should never collide with other cars, but that is not the *only* thing they are designed to do.

Dao et al. [30, 31] developed an algorithm and model for lane assignment. Their simulations suggest [30] that traffic safety can be enhanced if vehicles are organized into platoons, as opposed to having random space between them. Our approach considers an even more general setting: we not only verify safety for platoon systems, but also when cars are driving on a lane without following platooning controllers. Hall et al. [33] also used simulations to find out what is the best strategy of maximizing traffic throughput. Chee et al. [42] showed that lane change maneuvers can be achieved in automated highway systems using the signals available from on-board sensors. Jula et al. [36] used simulations to study the conditions under which accidents can be avoided during lane changes and merges. They have only tested safety partially. In contrast to [30, 31, 33, 36, 42], we do not use simulation but formal verification to validate our hypotheses.

Hsu et al. [34] propose a control system for IVHS that organizes traffic in platoons of closely spaced vehicles. They specify this system by interacting finite state machines. Those cannot represent the actual continuous movement of the cars. We use differential equations to model the continuous dynamics of the vehicles and thus consider more realistic models of the interactions between vehicles, their control, and their movement.

Stursberg et al. [39] applied counterexample-guided verification to a cruise control system with two cars on one lane. Their technique can not scale to an arbitrary number of cars. Althoff et al. [44] use reachability analysis to prove the safety of evasive maneuvers with constant velocity. They verify a very specific situation: a wrong way driver threatens two autonomously driving vehicles on a road with three lanes.

There are several challenges that still need to be solved to make next generation car control a reality. The most interesting challenge for us is that it only makes sense to introduce any of these systems after its correct functioning and reliability has been ensured. Otherwise, the system might do more harm than good. This is the formal verification problem for distributed car control. What makes this problem particularly challenging is its complicated dynamics. Distributed car control follows a hybrid dynamics, because cars move continuously along differential equations and their behavior is affected by discrete control decisions like when and how strongly to brake or to accelerate and to steer.

In [10] we develop a distributed car control system and a formal proof that this system is collision-free for arbitrarily many cars, even when new cars enter or leave a multi-lane highway with arbitrarily many lanes. This was done by explicitly developing the model to take advantage of the compositional aspects of the system (see Fig. 1.1). We reduce the proof to subsystems that can be verified without awareness of details about the system as a whole. We believe the

principles behind this modular structure and verification techniques are useful for other systems and that they can be formalized by using the refinement operator in dR$\mathcal{L}$.

A comprehensive discussion of the current landscape of verification for cyber-physical systems can be found in [46, 47].

2.5 Aircraft Control

Verification of air traffic control is particularly challenging because it lies at the intersection of many fields which already give tough verification problems when examined independently. It is a distributed system, with a large number of aircraft interacting over an unbounded time horizon. Each aircraft has nonlinear continuous dynamics combined with complex discrete controllers. And finally, every protocol must be flyable (i.e. not cause the aircraft to enter a stall, bank too sharply, or require it to turn on sharp corners). The complexity of curved flight dynamics has been difficult for many analysis techniques [48–55], which often resort to unflyable approximations of flight trajectories that require aircraft to turn on corners.

These strong guarantees are especially important in a distributed system with a large number of interacting participants. As in [48, 51, 54, 56, 57], many previous approaches to aircraft control have looked into a relatively small number of agents. But with thousands of aircraft flying through airspace daily, this system is already far too complex for humans to predict every scenario by looking at interactions of only a few aircraft.

Verification methods for systems with an arbitrary number of agents behaving under distributed control fall primarily into one of two categories: theorem proving and parameterized verification. Johnson and Mitra [55] use parameterized verification to guarantee that a distributed air traffic landing protocol (SATS) is collision free. Using backward reachability, they prove safety in the SATS landing maneuver given a bound on the number of aircraft that can be engaged in the landing maneuver. The protocol divides the airspace into regions and models the aircraft flight trajectory within each region by a clock.

Other provably safe systems with a specific (usually small) number of agents are presented in [48, 51, 54, 56]. The work by Umeno and Lynch [51, 54] considers real-time properties of airport protocols using Timed I/O Automata. Duperret et al. [56] verify a roundabout maneuver with three vehicles. Each vehicle is constrained to a specific, pre-defined path, so physical dynamics are simplified to one dimension. Tomlin et al. [48] analyze competitive aircraft maneuvers game theoretically using numerical approximations of partial differential equations. As a solution, they propose roundabout maneuvers and give bounded-time verification results for up to four aircraft using straight-line approximations of flight dynamics.

Flyability is identified as a major challenge in Košecká et al. [58], where planning based on superposition of potential fields is used to resolve air traffic conflicts. This planning does not guarantee flyability but, rather, defaults to classical vertical altitude changes whenever a nonflyable path is detected. The resulting maneuver has not yet been verified. The planning approach has been pursued by Bicchi and Pallottino [59] with numerical simulations.

Numerical simulation algorithms approximating discrete-time Markov Chain approximations of aircraft behavior have been proposed by Hu et al. [49]. They approximate bounded-time probabilistic reachable sets for one initial state. We consider hybrid systems combining discrete

control choices and continuous dynamics instead of uncontrolled, probabilistic continuous dynamics. Hwang et al. [53] have presented a straight-line aircraft conflict avoidance maneuver involving optimization over complicated trigonometric computations, and validate it using random numerical simulation and informal arguments. The work of Dowek et al. [50] and Galdino et al. [52] provides formal geometrical proofs in PVS, but considers straight-line maneuvers, which are unflyable because they require instantaneous changes in direction (i.e. turning on a corner).

Pallottino et al. [60] proposed a distributed collision avoidance policy that provides a thorough empirical description of the system's behavior, emphasizing simulation and physical experiment. They formulate a liveness property and give probabilistic evidence for it using Monte Carlo methods. They also provide an informal proof of safety that is similar in high-level ideas to the proofs in [6], but does not consider a model for flight dynamics.

The collision avoidance maneuvers we present in [6] use curved flight, which is *flyable*, but much more difficult to analyze. We produce *formal*, deductive proofs to verify uncountably many initial states and give unbounded-time horizon verification results. We use symbolic computation so that numerical and floating point errors can not violate soundness. We analyze hybrid system dynamics directly, rather than approximations like clocks. We verify the case of *arbitrarily many* aircraft, which is crucial for dense airspace. In Section 1.4.2, we propose that the proofs presented in [6] may be simplified using differential refinement logic, with an outline described in Section 8.4.

Chapter 3

Differential Refinement Logic

In this chapter we will develop the differential refinement logic (dRℒ) presented in the thesis. In addition to defining the syntax (Section 3.1) and semantics (Section 3.2) of dRℒ, we present a set of proof rules for the logic (Section 3.4) and soundness proofs for each of these rules (Section 3.6). We present some simple example proofs using dRℒ in Section 3.5.

3.1 Syntax

In this section, we extend differential dynamic logic (dℒ), a specification and verification language for hybrid systems, and therefore we make heavy use of constructs introduced in [3]. Both dℒ and dRℒ model cyber-physical systems as *hybrid programs* (HPs). HPs combine differential equations with traditional program constructs and discrete assignments. As has been previously noted [4], HPs form a Kleene algebra with tests [14].

Definition 1 (Hybrid program)**.** HPs are defined by the following grammar (where α, β are HPs, x a variable, θ a term possibly containing x, and H a formula of dRℒ):

$$\alpha, \beta \ ::= \ x := \theta \mid x' = \theta \,\&\, H \mid ?H \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*$$

The effect of *assignment* $x := \theta$ is an instantaneous discrete jump assigning θ to x. The effect of *differential equation* $x' = \theta \,\&\, H$ is a continuous evolution where the differential equation $x' = \theta$ holds *and* (written & for clarity) formula H holds throughout the evolution (the state remains in the region described by H).

The effect of *test* $?H$ is a *skip* (i.e., no change) if formula H is true in the current state and *abort* (blocking the system run by a failed assertion), otherwise. *Non-deterministic choice* $\alpha \cup \beta$ is for alternatives in the behavior of the distributed hybrid system. In the *sequential composition* $\alpha;\beta$, HP β starts after α finishes (β never starts if α continues indefinitely). *Non-deterministic repetition* α^* repeats α an arbitrary number of times, including zero times.

Except for the changes to formulas (addressed in Definition 2), the grammar of hybrid programs is unchanged from that used by dℒ [3].

Definition 2 (dRℒ formula)**.** Formulas in dRℒ are defined by the following grammar (where ϕ, ψ are dRℒ formulas, x is a variable, θ_1, θ_2 are terms, and α, β are HPs):

$$\phi, \psi \ ::= \ \theta_1 \leq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi \mid \boldsymbol{\beta \leq \alpha}$$

In addition to all formulas of first-order real arithmetic, dRℒ allows formulas of the form $\beta \leq \alpha$ with HPs α and β. Formula $\beta \leq \alpha$ is true in a state ν iff all states reachable from ν by following the transitions of β could also be reached from state ν by following transitions of α (the transition semantics of HPs are formally defined in Definition 3). Less formally, the behaviors of α subsume those of β, or we say that β *refines* α from the current state.

Just as in dℒ, we may write formula $[\alpha]\phi$ with an HP α and a formula ϕ in dRℒ. Formula $[\alpha]\phi$ is true in a state ν iff formula ϕ is true in all states that are reachable from ν by following the transitions of α.

The formulas $\beta \leq \alpha$ and $[\alpha]\phi$ make a powerful pair; when both are true, then we know that formula ϕ is true in all states reachable from ν by following the transitions of β, i.e. $[\beta]\phi$.

We use $[\alpha]\phi$, $\beta \leq \alpha$, and other formulas in dRℒ for stating and proving properties of HPs.

3.2 Semantics

The semantics of HPs are defined as a reachability relation. A *state* ν is a mapping from a set V of logical and state variables to $\mathbb{R}$. The set of states is denoted $\mathcal{S}$. The value of term θ in state ν is denoted by $[\![\theta]\!]_\nu$. The transition semantics of HPs remain the same as in dℒ [3, 4].

Definition 3 (Transition semantics of HPs)**.** Each HP α is interpreted semantically as a binary reachability relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ over states, defined inductively and as usual in differential dynamic logic (dℒ):

- $\rho(x := \theta) = \{(\nu, \omega) \ : \ \omega = \nu \text{ except that } [\![x]\!]_\omega = [\![\theta]\!]_\nu\}$
- $\rho(?H) = \{(\nu, \nu) \ : \ \nu \models H\}$
- $\rho(x' = \theta \,\&\, H) = \{(\varphi(0), \varphi(r)) \ : \ \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \text{ for a solution } \varphi : [0, r] \to \mathcal{S} \text{ of any duration } r\}$; i.e., with $\varphi(t)(x') \overset{\text{def}}{=} \frac{d\varphi(\zeta)(x)}{d\zeta}(t)$, φ solves the differential equation and satisfies H at all times.
- $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
- $\rho(\alpha;\beta) = \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\}$
- $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?\mathit{true}$

Definition 4 (dRℒ semantics)**.** The *satisfaction relation* $\nu \models \phi$ for a dRℒ formula ϕ in state ν is defined inductively and, aside from the refinement and equivalence relations, it is defined as usual in first-order modal logic for real arithmetic.

If $\nu \models \phi$, we say that ϕ holds at state ν. A formula ϕ is *valid* iff ϕ holds at all states, i.e. $\nu \models \phi$ for all $\nu \in \mathcal{S}$; we write $\models \phi$ to denote that ϕ is valid. We use ν^d_x to be the state ν with the variable x assigned to real value d.

- $\nu \models (\theta_1 \leq \theta_2)$ iff $[\![\theta_1]\!]_\nu \leq [\![\theta_2]\!]_\nu$.
- $\nu \models \neg F$ iff $\nu \not\models F$, i.e. if it is not the case that $\nu \models F$.
- $\nu \models F \wedge G$ iff $\nu \models F$ and $\nu \models G$.

- $\nu \models \forall x\, F$ iff $\nu_x^d \models F$ for all $d \in \mathbb{R}$.
- $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$
- $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$
- $\boldsymbol{\nu \models \alpha \leq \beta}$ **iff** $\boldsymbol{\{\omega : (\nu, \omega) \in \rho(\alpha)\} \subseteq \{\mu : (\nu, \mu) \in \rho(\beta)\}}$

Differential dynamic logic combines first-order real arithmetic and dynamic logic generalized for hybrid programs [4]. Formulas in first-order arithmetic can be used to precisely specify safety or control regions. In $\mathsf{d}\mathcal{L}$, a box modality can be used to express that a property ϕ holds in *all* states reachable by a transition of hybrid program α; the box modality is written $[\alpha]\phi$. Similarly, the diamond modality requires that ϕ hold in at least one state reachable by a transition of α; the diamond modality is written $\langle\alpha\rangle\phi$.

Differential refinement logic $\mathsf{dR}\mathcal{L}$ introduces the refinement relation for hybrid programs to the semantics. This addition is indicated in bold in Definition 4. The formula $\alpha \leq \beta$ is true in state ν iff the set of all states reachable from ν by following the transitions of HP α is a subset of the states reachable from ν by following the transitions of HP β.

We also use $\alpha = \beta$ to denote equivalence of hybrid programs α and β, but this is defined syntactically as $\alpha \leq \beta \wedge \beta \leq \alpha$.

Notice that the state space, $\mathcal{S}$, is the collection of all mappings of variables to $\mathbb{R}$. Even when there is a variable x in hybrid program β which does not appear in hybrid program α, the variable x is still in the mapping of the state space of ν and the transition $\rho(\alpha)$ will require that the value of x is unchanged.

3.3 Relating $\mathsf{dR}\mathcal{L}$ and $\mathsf{d}\mathcal{L}$

We can show that the refinement relation can actually be written equivalently in $\mathsf{d}\mathcal{L}$, so we could easily lift the axioms from $\mathsf{d}\mathcal{L}$ and add the equivalence transformation to accomplish completeness for $\mathsf{dR}\mathcal{L}$ based on the relative completeness of $\mathsf{d}\mathcal{L}$ [61]:

$$\models_{\mathsf{dR}\mathcal{L}} \alpha \leq \beta \iff \models_{\mathsf{d}\mathcal{L}} \forall \bar{x}\, (\langle\alpha\rangle(x = \bar{x}) \rightarrow \langle\beta\rangle(x = \bar{x})), \tag{3.1}$$

where x is a vector of all bound variables in either α or β, and $\bar{x}$ is a vector of fresh variables of equal length to x. While this observation gives us an easy out for completeness, it's important to keep in mind that converting refinement into a $\mathsf{d}\mathcal{L}$ property in this way completely undermines what we are trying to accomplish with $\mathsf{dR}\mathcal{L}$ – taking advantage of the structure of a hybrid program to prove refinement relations. In practice, the formula on the right-hand side of (3.1) will be significantly more complicated to verify (due to the added diamond modalities and quantifying over variables) than verifying properties about α and β directly, making this a terrible idea in practice. Additionally, (3.1) can only be expressed in $\mathsf{d}\mathcal{L}$ for concrete α, β, but refinement in $\mathsf{dR}\mathcal{L}$ can be defined for all α, β.

The canonical goal of $\mathsf{dR}\mathcal{L}$ is to be able to statically verify safety for a hybrid program α by showing that it refines a verified system β. What we often gain is a verified system α that more accurately models the real-world system than β. However, α will always be a model of the real world, since all the continuous environment and state variables can never be precisely captured by any representation. So, this gap between the model the real world will always exist, even if the gap has been reduced using $\mathsf{dR}\mathcal{L}$.

To address this gap, ModelPlex (Mitsch et al. [12]) implements a run-time analysis of $(\langle\alpha\rangle(x = \bar{x}) \rightarrow \langle\beta\rangle(x = \bar{x}))$ from equation (3.1). Instead of using hybrid program α to represent the system, ModelPlex samples the actual values reached by the system in real time. By doing this, we can replace $\langle\alpha\rangle(x = \bar{x})$ with $x = \dot{x}$, where $\dot{x}$ is the observed state of the system. What remains is for a run-time monitor to check whether the real-world implementation of a system is a refinement of the statically verified model β by checking that $\langle\beta\rangle(x = \bar{x})$ holds for reachable states $\bar{x}$, but this can sometimes be tricky to compute with limited resources in real-time. By using quantifier elimination to statically handle many of the computational complexities in advance of runtime, this technique is in scope for several systems [12]. This is a great way to bridge the final gap between real-world systems and their models. However it is still a worthy goal to keep that gap as small as possible, since run-time monitoring may catch problems too late due to discrete sampling of the state, or require controllers to be more conservative.

Just as it is possible to rewrite refinement statements using dL, it is also possible to encode the box modality of dL purely as a refinement relation:

$$\models_{\mathsf{dL}} [\alpha]\phi \iff \models_{\mathsf{dRL}} \alpha \leq x := *; ?\phi, \tag{3.2}$$

where x again is a vector of all bound variables in α, and $x := *$ is a nondeterministic assignment. We can define nondeterministic assignment as syntactic sugar: $(x := *) = (x := 0; x' = 1; x' = -1)$. The HP $x := *; ?\phi$ transitions from some starting state ν to any state that may differ from ν only on the bound variables of α and in which ϕ is satisfied.

Lemma 1 (Expressiveness)**.** *dRL and dL are equally expressive: every formula in one logic has a formula in the other logic that is equivalent. Even when dropping modalities from dRL, dL and dRL are still equally expressive.*

Proof. dL is a fragment of dRL. dRL provides the extension of adding $\alpha \leq \beta$ as a logical formula, which is definable by an equivalent dL formula according to (3.1). To show that dRL without modalities can express all dL formula, we need to show by induction that all modal formulas $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ of dL can be expressed in dRL without modalities. By induction hypothesis, ϕ can be assumed to have been replaced by an equivalent without modalities already. For the formula $[\alpha]\phi$, (3.2) is an equivalent encoding in dRL. For the formula $\langle\alpha\rangle\phi$, which is equivalent to $\neg[\alpha]\neg\phi$, the dRL formula $\neg(\alpha \leq x := *; ?\neg\phi)$ is an equivalent encoding. □

These encodings illustrate that this thesis does not set out to prove that dRL has more expressivity than dL. Neither is our goal to show that the proof calculus of dRL improves upon the theoretical completeness results already established for dL ([1] proves dL is complete with respect to differential equations). However, in practice, we observe utility in combining these modes of reasoning. The background and familiarity with dL that we have gained through previously completed work (discussed in Section 1.4) has shown a serious need for proper refinement.

The semantic definitions of terms and hybrid programs in dL are identical to the semantic definitions of terms and hybrid programs without refinement in dRL. With the transformation in (3.1), and because the calculus for dL is completed, any property proved using the sound calculus of dRL is also provable in dL. In other words, proofs in dL may depend on theorems proved in dRL.

We develop a proof calculus for dRℒ in order to directly leverage structural similarities between HPs with comparable behaviors. The improvements that dRℒ brings to dℒ will depend on the quality of refinement-specific proof rules. To evaluate dRℒ's usefulness, we will show that it aids verification of the examples with which we are most familiar (see Section 1.4).

3.4 Proof Calculus

A proof calculus associated with a logical language such as dRℒ is a set of syntactic transformations that are each proved sound. By combining many of these transformations on a complicated formula, we may simplify and break apart the formula until we are left with formulas that can be proved true using quantifier elimination, in which case we have a proof of our original complicated formula. Because this process is entirely syntactic, such a proof can be automatically checked by a computer or, more importantly, automatically generated (this process is called a proof search). In this section we present sequent proof rules for dRℒ. The semantics of a sequent $\Gamma \vdash \Delta$ is that of $\bigwedge_{\phi\in\Gamma}\phi \rightarrow \bigvee_{\psi\in\Delta}\psi$. This collection of rules is not complete, but should be considered a set of rules which we found to be useful in proving common refinement properties.

$$\frac{}{\Gamma \vdash \alpha \leq \alpha, \Delta}(\leq_{refl}) \qquad \frac{\Gamma \vdash \alpha \leq \beta, \Delta \qquad \Gamma \vdash \beta \leq \gamma, \Delta}{\Gamma \vdash \alpha \leq \gamma, \Delta}(\leq_{trans})$$

$$\frac{\Gamma \vdash \alpha \leq \beta, \Delta \qquad \Gamma \vdash \beta \leq \alpha, \Delta}{\Gamma \vdash \alpha = \beta, \Delta}(\leq_{antisym})^1$$

Figure 3.1: dRℒ rules - refinement is a partial order

The rules in Fig. 3.1 capture the fact that the refinement relation over hybrid programs is a partial order. The rules in Fig. 3.2 are derived directly from KAT axioms [14]. For hybrid programs, nondeterministic choice $\cup$ is the additive operator, and sequential composition ; is the multiplicative operator.

The hybrid program $?\bot$, where $\bot$ is the formula false, is a test that always fails. The transition semantics for this hybrid program is the empty set. It is therefore used as the additive identity (see rule $\cup_{id}$ in Fig. 3.2) and the multiplicative annihilator (see rules $;_{annih-r}$ and $;_{annih-l}$ in Fig. 3.2). The hybrid program $?\top$, where $\top$ is the formula true, is a test that always succeeds. We use $?\top$ as the multiplicative identity (see rules $;_{id-r}$ and $;_{id-l}$ in Fig. 3.2), as it does not change the transition semantics of any hybrid program when sequentially composed (i.e. $\rho(?\top)$ is the identity relation).

Two proof rules that provide strong motivation for developing dRℒ can be seen in Fig. 3.3. Recall that a box modality $[\alpha]\phi$ holds only if ϕ holds in every state reachable by following transitions on α. So, the refinement rule for box modalities $[\leq]$ says that if formula ϕ holds in every state reachable on β, and α is a refinement of β, then ϕ must also hold in every state reachable on α. A diamond modality $\langle\alpha\rangle\phi$ holds if there is at least one transition on α to a state

[1]Equivalence of hybrid programs ($\alpha = \beta$) is syntactic sugar for $\alpha \leq \beta \wedge \beta \leq \alpha$, so this rule holds by definition.

$$\frac{}{\Gamma \vdash \alpha \cup (\beta \cup \gamma) = (\alpha \cup \beta) \cup \gamma, \Delta}(\cup_{assoc}) \qquad \frac{}{\Gamma \vdash \alpha \cup \beta = \beta \cup \alpha, \Delta}(\cup_{comm})$$

$$\frac{}{\Gamma \vdash \alpha \cup ?\bot = \alpha, \Delta}(\cup_{id}) \qquad \frac{}{\Gamma \vdash (\alpha \cup \alpha) = \alpha, \Delta}(\cup_{idemp})$$

$$\frac{}{\Gamma \vdash \alpha; (\beta; \gamma) = (\alpha; \beta); \gamma, \Delta}(;_{assoc}) \qquad \frac{}{\Gamma \vdash (?\top; \alpha) = \alpha, \Delta}(;_{id-l}) \qquad \frac{}{\Gamma \vdash (\alpha; ?\top) = \alpha, \Delta}(;_{id-r})$$

$$\frac{}{\Gamma \vdash \alpha; (\beta \cup \gamma) = ((\alpha; \beta) \cup (\alpha; \gamma)), \Delta}(dist\text{-}l) \qquad \frac{}{\Gamma \vdash (\alpha \cup \beta); \gamma = ((\alpha; \gamma) \cup (\beta; \gamma)), \Delta}(dist - r)$$

$$\frac{}{\Gamma \vdash (?\bot; \alpha) = ?\bot, \Delta}(;_{annih-l}) \qquad \frac{}{\Gamma \vdash (\alpha; ?\bot) = ?\bot, \Delta}(;_{annih-r})$$

$$\frac{}{\Gamma \vdash (?\top \cup (\alpha; \alpha^*)) = \alpha^*, \Delta}(unroll_l) \qquad \frac{}{\Gamma \vdash (?\top \cup (\alpha^*; \alpha)) = \alpha^*, \Delta}(unroll_r)$$

$$\frac{\Gamma \vdash [\alpha^*](\alpha; \gamma) \leq \gamma, \Delta \qquad \Gamma \vdash [\alpha^*]\beta \leq \gamma, \Delta}{\Gamma \vdash \alpha^*; \beta \leq \gamma, \Delta}(loop_l) \qquad \frac{\Gamma \vdash \beta \leq \gamma, \Delta \qquad \Gamma \vdash (\gamma; \alpha) \leq \gamma, \Delta}{\Gamma \vdash \beta; \alpha^* \leq \gamma, \Delta}(loop_r)$$

Figure 3.2: Idempotent semiring/KAT axioms for dRL

$$\frac{\Gamma \vdash [\beta]\phi, \Delta \qquad \Gamma \vdash \alpha \leq \beta, \Delta}{\Gamma \vdash [\alpha]\phi, \Delta}([\leq]) \qquad \frac{\Gamma \vdash \langle\alpha\rangle\phi, \Delta \qquad \Gamma \vdash \alpha \leq \beta, \Delta}{\Gamma \vdash \langle\beta\rangle\phi, \Delta}(\langle\leq\rangle)$$

Figure 3.3: dRL modality rules

where ϕ is true. We have a similar rule for diamond modalities $\langle\leq\rangle$, which says that if α is a refinement of β, and ϕ holds in at least one transition on α, then that same state must also be reachable on β and therefore $\langle\beta\rangle\phi$ must be true, since β contains all behaviors that α can have.

In Fig. 3.4, we present three rules for handling differential equations using refinement. The DC rule says that if we prove that a differential equation always evolves within some region H_2, then this program is equivalent to the same hybrid program, but with the conjunction of H_2 in the evolution domain. This rule is so named because it is reminiscent of differential cut from dL.

The DR rule says that if two differential equations differ only in their evolution domain, then a refinement relationship is satisfied if the evolution domain of the smaller program is a

[2]We require that this rule be defined only when no variables of x occur in θ.

$$\frac{\Gamma \vdash [x' = \theta \,\&\, H_1]H_2, \Delta}{\Gamma \vdash (x' = \theta \,\&\, H_1) = (x' = \theta \,\&\, H_1 \wedge H_2), \Delta}(DC) \qquad \frac{\Gamma \vdash \forall x\,(H_1 \rightarrow H_2), \Delta}{\Gamma \vdash (x' = \theta \,\&\, H_1) \leq (x' = \theta \,\&\, H_2), \Delta}(DR)$$

$$\frac{\Gamma \vdash \forall x \Big(\theta_1 \|\theta_2\| = \theta_2 \|\theta_1\| \wedge (\|\theta_1\|^2 = 0 \leftrightarrow \|\theta_2\|^2 = 0)\Big), \Delta}{\Gamma \vdash (x' = \theta_1) = (x' = \theta_2), \Delta}\ (MDF)^2$$

Figure 3.4: dRL rules for handling differential equations.

subset of the evolution domain of the larger program. This rule is so named because a more general version of differential refinement exists for differential algebraic logic [62], where both the evolution domains and the differential algebraic constraints are compared. In the premise, we quantify over the state variables of the two hybrid programs $x' = \theta_1$ and $x' = \theta_2$. This is simply the vector x. Quantifying over x prevents the unsound assumption that the context holds throughout the evolution, when in fact the context is a static property about the initial value of the state x, and is not guaranteed to hold as x evolves.

The *match direction field* (MDF) rule concerns the reachability of two continuous evolutions. Two differential equations are equivalent if they have the same set of reachable states, even if those states are not reached at the same time. We quantify over x for similar reasons as in (DR).

We have proved soundness for MDF for constant differential equations (i.e. in the case where θ does not depend on x); however, we present a version of the proof rule that we believe will be generalizable without this assumption. So, while the premise could be simplified under this assumption, we have decided to leave it in its present form. To give some intuition for the rule in the generalized case, when the premise is valid, it requires that the unit direction field[3] of both differential equation systems be identical. In other words, they share all equilibrium points[4], and $\frac{\theta_1}{\|\theta_1\|} = \frac{\theta_2}{\|\theta_2\|}$ holds everywhere else. When the unit direction fields of two differential equations are identical, then their phase portraits[5] are as well. This means that if the two systems start at the same initial value, then the path their trajectory follows through the state space will also be identical. So, while the two systems do not evolve along their trajectories at the same rates, they do have identical sets of reachable states.

The proof rules presented in Fig. 3.5 generally take advantage of structural similarities between hybrid programs. For example, the *unloop* rule allows both hybrid programs α and β to be unrolled simultaneously. It is important that we update the context appropriately, as the refinement must hold after any number of loop executions, which we accomplish by requiring the

[3]Also known as a slope field, the *direction field* is a graphical representation of a system of differential equations, which plots a vector of the slope of the solution of the differential equation at each point in the state space. A unit direction field is one where each of these vectors has magnitude one.

[4]An *equilibrium point* is a point in the state space where the derivative is zero. More formally, x_0 is an equilibrium point for $\frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x)$ if $f(t, x_0) = 0$ for all t. For linear differential equations, which are polynomial in t, equilibrium points are constant solutions.

[5]A *phase portrait* is the trajectory of a differential equation solution in the state space from a given initial value.

$$\frac{\Gamma \vdash \alpha \leq \gamma \wedge \beta \leq \gamma, \Delta}{\Gamma \vdash \alpha \cup \beta \leq \gamma, \Delta}(\cup_l) \qquad \frac{\Gamma \vdash \alpha \leq \beta \vee \alpha \leq \gamma, \Delta}{\Gamma \vdash \alpha \leq \beta \cup \gamma, \Delta}(\cup_r)$$

$$\frac{\Gamma \vdash [\alpha^*](\alpha \leq \beta), \Delta}{\Gamma \vdash \alpha^* \leq \beta^*, \Delta}(\mathit{unloop}) \qquad \frac{\Gamma \vdash \alpha_1 \leq \alpha_2, \Delta \qquad \Gamma \vdash [\alpha_1](\beta_1 \leq \beta_2), \Delta}{\Gamma \vdash (\alpha_1;\beta_1) \leq (\alpha_2;\beta_2), \Delta}(;)$$

$$\frac{}{\Gamma \vdash (x := \theta) \leq (x := *), \Delta}(:= *) \qquad \frac{\Gamma \vdash \phi \rightarrow \psi, \Delta}{\Gamma \vdash ?\phi \leq ?\psi, \Delta}(?)$$

Figure 3.5: dRL structural rules

$\alpha \leq \beta$ hold after an arbitrary number of executions of α. The $:= *$ proof rule says that assigning x to a specific term θ always refines a program which assigns x nondeterministically to any real value (see Example 4 to see this rule applied in the context of a loop, and Example 5 to see how this rule can be extended to cover guarded nondeterministic assignment). While this list is not exhaustive and there may be additional useful proof rules for structurally decomposing refinement properties, these are the rules that we have found to be most crucial through experience in proving dRL properties.

3.5 Library of Derived Rules

In the following examples we use $*$ at the top of a proof branch to denote that the branch is closed, either by a proof rule, or by using a decidable algorithm to resolve any remaining $\text{FOL}_{\mathbb{R}}$ properties.

Example 1 ($\alpha \cup \beta = \beta \leftrightarrow \alpha \leq \beta$)**.** In an idempotent semiring, we expect there to be a natural partial order induced by: $\alpha \cup \beta = \beta \leftrightarrow \alpha \leq \beta$, as described by Kozen in [63]. And, indeed, we find that this rule can also be derived for the refinement relation for hybrid programs from the above rules as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdash \alpha \leq \beta \leftrightarrow \alpha \leq \beta}ax}{\vdash (\alpha \leq \beta \wedge \beta \leq \beta) \leftrightarrow \alpha \leq \beta}\leq_{refl}}{\vdash \alpha \cup \beta \leq \beta \leftrightarrow \alpha \leq \beta}\cup_l}{\vdash (\alpha \cup \beta \leq \beta \wedge \beta \leq \alpha \cup \beta) \leftrightarrow \alpha \leq \beta}\cup_r, \leq_{refl}}{\vdash \alpha \cup \beta = \beta \leftrightarrow \alpha \leq \beta}\leq_{antisym}$$

Example 2 (Nondeterministic choice and disjunction of tests)**.** In this example we can show, using refinement, that a nondeterministic choice between two tests is equivalent to a single test which joins the two formulas with a disjunction.

A more generic form of this atomic example would be to prove that there is an equivalence between any hybrid program and its test representation. For example, if the reachable states of α

are exactly ϕ, then we would like to be able to prove that $\alpha = (x := *; ?\phi)$. Notice that proving equivalence for these hybrid programs would also prove box and diamond modality properties $[\alpha]\phi$ and $\langle\alpha\rangle\phi$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma,\phi\vdash\phi,\psi,\Delta}\,ax \quad \dfrac{*}{\Gamma,\psi\vdash\phi,\psi,\Delta}\,ax}{\Gamma,\phi\vee\psi\vdash\phi,\psi,\Delta}\,\vee_l}{\Gamma\vdash(\phi\vee\psi)\to\phi,(\phi\vee\psi)\to\psi,\Delta}\,\to_r}{\Gamma\vdash ?(\phi\vee\psi)\le ?\phi,?(\phi\vee\psi)\le ?\psi,\Delta}\,?}{\Gamma\vdash(?(\phi\vee\psi)\le ?\phi)\vee(?(\phi\vee\psi)\le ?\psi),\Delta}\,\vee_r}{\Gamma\vdash ?(\phi\vee\psi)\le(?\phi\cup ?\psi),\Delta}\,\cup_r \quad \dfrac{\dfrac{*}{\Gamma\vdash\phi\to(\phi\vee\psi),\Delta}\,{\scriptstyle\to_r,\vee_r,ax}}{\Gamma\vdash ?\phi\le ?(\phi\vee\psi),\Delta}\,? \quad \dfrac{\dfrac{*}{\Gamma\vdash\psi\to\phi\vee\psi,\Delta}\,{\scriptstyle\to_r,\vee_r,ax}}{\Gamma\vdash ?\psi\le ?(\phi\vee\psi),\Delta}\,?}{\Gamma\vdash(?\phi\cup ?\psi)\le ?(\phi\vee\psi),\Delta}\,{\scriptstyle\cup_l,\wedge_r}}{\Gamma\vdash(?\phi\cup ?\psi)= ?(\phi\vee\psi),\Delta}\,{\scriptstyle\le_{antisym}}$$

Example 3 (Sequential composition and conjunction of tests)**.** In this example we show, using refinement, that sequentially composing two tests is equivalent to a single test which joins the two formulas with a conjunction.

$$\dfrac{}{\Gamma\vdash ?(\phi\wedge\psi)=(?\phi;?\psi),\Delta}\,{\scriptstyle(\wedge?)}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma,\phi,\psi\vdash\phi,\Delta}\,ax}{\Gamma\vdash ?(\phi\wedge\psi)\le ?\phi,\Delta}\,{\scriptstyle ?,\to_r} \quad \dfrac{\dfrac{\dfrac{*}{\Gamma,\phi,\psi\vdash(\top\to\psi),\Delta}\,{\scriptstyle\to_r,ax}}{\Gamma,\phi,\psi\vdash(?\top\le ?\psi),\Delta}\,?}{\Gamma\vdash[?(\phi\wedge\psi)](?\top\le ?\psi),\Delta}\,{\scriptstyle[?],\to_r}}{\Gamma\vdash ?(\phi\wedge\psi)\le(?\phi;?\psi),\Delta}\,; \quad \dfrac{\dfrac{\dfrac{*}{\Gamma\vdash\phi\to\top,\Delta}\,{\scriptstyle\to_r,ax}}{\Gamma\vdash ?\phi\le ?\top,\Delta}\,? \quad \dfrac{\dfrac{*}{\Gamma,\phi\vdash(\psi\to(\phi\wedge\psi)),\Delta}\,{\scriptstyle\to_r,ax}}{\Gamma,\phi\vdash ?\psi\le ?(\phi\wedge\psi),\Delta}\,?}{\Gamma\vdash[?\phi](?\psi\le ?(\phi\wedge\psi)),\Delta}\,{\scriptstyle[?],\to_r}}{\Gamma\vdash(?\phi;?\psi)\le ?(\phi\wedge\psi),\Delta}\,;}{\Gamma\vdash ?(\phi\wedge\psi)=(?\phi;?\psi),\Delta}\,{\scriptstyle\le_{antisym}}$$

□

Example 4 (Decomposing a system inside a loop)**.** dRL performs particularly well when determining refinement between HPs which differ only slightly, but within the context of a large and complicated system. In this example, the only difference between the two programs we are comparing is that the program on the left is setting variable x to a specific value θ, and the program on the right allows x to be assigned any value. This is a great example of a proof that would be challenging in dL, but is straight forward using refinement.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdash\alpha\le\alpha}\,{\scriptstyle\le_{refl}} \quad \dfrac{*}{\vdash[\alpha](x:=\theta)\le(x:=*)}\,{\scriptstyle[]_{gen},\forall_r,:=*} \quad \dfrac{*}{\vdash[\alpha;x:=\theta]\beta\le\beta}\,{\scriptstyle\le_{refl}}}{\vdash(\alpha;x:=\theta;\beta)\le(\alpha;x:=*;\beta)}\,;}{\vdash\forall^{(\alpha;x:=\theta;\beta)}(\alpha;x:=\theta;\beta)\le(\alpha;x:=*;\beta)}\,\forall_r}{\vdash[(\alpha;x:=\theta;\beta)^*](\alpha;x:=\theta;\beta)\le(\alpha;x:=*;\beta)}\,[]_{gen}}{\vdash(\alpha;x:=\theta;\beta)^*\le(\alpha;x:=*;\beta)^*}\,{\scriptstyle unloop}$$

In order to take advantage of the similarities between these two HPs without refinement in dℒ, we would first have to deal with the outermost operator, in this case the Kleene star, by finding an appropriate loop invariant (see [64] for details on why finding invariants for HPs is a necessarily difficult task). By contrast, when using dRℒ, it is possible to be oblivious to the complexities of the two systems where they are the same and focus only on proving refinement in the places where they differ. This is an illustration of *breaking a system into parts*. Note that $\forall^{(\alpha;x:=\theta;\beta)}$ occurs after the generalization rule is applied. This is shorthand notation for all variables that occur bound in the hybrid program $\alpha; x := \theta; \beta$ and over approximates the reachable states of the hybrid program by allowing the bound variables to take on any value.

Example 5 (Guarded nondeterministic assignment)**.** This example illustrates the generalized safety envelopes use case. The nondeterministic assignment $x := *$ assigns an arbitrary real value to x. The test then restricts those values to be the ones satisfying the guard condition $\phi(x)$. Using guarded nondeterministic assignment can make a property easier to verify because it has stripped away all the details of the value of x except whatever is necessary for the proof of safety, in this case $\phi(x)$.

$$\dfrac{\dfrac{*}{\phi(\theta) \vdash (x := \theta) \leq (x := *)}\,{:=}*\quad \dfrac{\dfrac{\dfrac{*}{\phi(\theta) \vdash (\top \to \phi(\theta))}\,{\to_r, ax}}{\phi(\theta) \vdash (?\top \leq ?\phi(\theta))}\,{?}}{\phi(\theta) \vdash [x := \theta](?\top \leq ?\phi(x))}\,{[:=]}}{\phi(\theta) \vdash (x := \theta) \leq (x := *; ?\phi(x))}\,{;}$$

Recall the discussion from Section 1.1 about designing an adaptive cruise control that is fuel efficient. We still want to verify that this controller does not allow the car to crash; however, the code that is specific for fuel efficiency may make it hard to verify that the car is still safe. In this example, we can equate $x := \theta$ with the controller selecting the most fuel efficient choice. Additionally, we can think of $\phi(x)$ as being a condition on x that directly ensures safety. If that choice of θ satisfies property $\phi(\theta)$, in other words, if the fuel efficient controller satisfies the guard condition $\phi(\theta)$, then the fuel efficient controller is a refinement of the controller which is only concerned with safety. This is an example of *abstracting an implementation-specific design*.

Example 6 (Differential equations)**.** In the atomic case, it can often be easy to determine whether two programs are equivalent. For example, $(x := \theta_1) = (x := \theta_2)$ iff $\theta_1 = \theta_2$. However, the same rule does not apply to differential equations. Consider the following formulas:

$$\vdash\ (x' = 2) = (x' = 9) \tag{3.3}$$

$$\vdash\ (x' = 2, t' = 1) \neq (x' = 9, t' = 1) \tag{3.4}$$

In (3.3), because the duration of the evolution is nondeterministic, the effect of evolving for an arbitrary duration with a positive derivative is simply that the value of x after evolution is anything greater than or equal to the initial value of x. These two programs differ only in duration, but their reachability relation is the same so long as there is no variable that observes time. However, in (3.4), time is now being recorded in the variable t. If the two evolutions differ

in their duration, the discrepancy is recorded and therefore these programs are not equivalent.

$$\dfrac{\dfrac{*}{\vdash \forall x.((2 \cdot ||9|| = 9 \cdot ||2||) \wedge (||2||^2 = 0 \leftrightarrow ||9||^2 = 0))}\,QE}{\vdash (x' = 2) = (x' = 9)}\,MDF$$

Example 7 (Differential Cut and Refinement)**.** In this example, we use both differential cut and differential refinement to prove our property. Ultimately we want to show that $x \geq 0$ is an invariant of the differential equation. However, we can not do this directly without first showing that $y \geq 0$ is also invariant. Once both $x \geq 0$ and $y \geq 0$ have been cut in as invariants, we then use DR to ignore the helper invariant $y \geq 0$ and close the proof.

The DC and DR rules can also be especially helpful when a differential equation does not have a solution, or when that solution is computationally intractable.

$$\dfrac{\dfrac{\dfrac{\dots}{x \geq 0 \wedge y \geq 0 \vdash [x' = y, y' = 1]y \geq 0}\,DI}{x \geq 0 \wedge y \geq 0 \vdash (x' = y, y' = 1) = (x' = y, y' = 1 \,\&\, y \geq 0)}\,DC}{\text{BRANCH 2}}$$

$$\dfrac{\text{BRANCH 2} \qquad \dfrac{\dfrac{\dfrac{\dots}{x \geq 0 \wedge y \geq 0 \vdash [x' = y, y' = 1 \,\&\, y \geq 0](x \geq 0 \wedge y \geq 0)}\,DI}{x \geq 0 \wedge y \geq 0 \vdash (x' = y, y' = 1 \,\&\, y \geq 0) = (x' = y, y' = 1 \,\&\, x \geq 0 \wedge y \geq 0)}\,DC}{x \geq 0 \wedge y \geq 0 \vdash (x' = y, y' = 1) = (x' = y, y' = 1 \,\&\, x \geq 0 \wedge y \geq 0)}\,\leq_{trans}}{\text{BRANCH 1}}$$

$$\dfrac{\text{BRANCH 1} \qquad \dfrac{\dfrac{*}{\vdash \forall x, y\,(x \geq 0 \wedge y \geq 0 \rightarrow x \geq 0)}\,QE}{\vdash (x' = y, y' = 1 \,\&\, x \geq 0 \wedge y \geq 0) \leq (x' = y, y' = 1 \,\&\, x \geq 0)}\,DR}{x \geq 0 \wedge y \geq 0 \vdash (x' = y, y' = 1) \leq (x' = y, y' = 1 \,\&\, x \geq 0)}\,\leq_{trans}$$

3.6 Soundness Proofs

3.6.1 Semantic Proofs of Soundness

In the following proofs, we define composition of transition relations $\rho(\alpha) \circ \rho(\beta)$ as the set of transitions that first follow α and then follow β:

$$\rho(\alpha) \circ \rho(\beta) = \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\}$$

We also use this notation when restricting a transition relation to starting from a fixed state ν. In other words, the set of all states reachable from ν through α is denoted as follows:

$$\{\nu\} \circ \rho(\alpha) = \{\omega : (\nu, \omega) \in \rho(\alpha)\}$$

Before we begin the proofs of soundness, we present a few lemmas that will be useful in the remaining proofs. Most of these lemmas rely heavily on the semantic definition of formulas, which was originally presented in Definition 4, but which we repeat here with the notation defined above for easy reference. One small difference is the introduction of the shorthand notation for restricting a transition relation to start from a fixed state: $\{\nu\} \circ \rho(\alpha)$.

The *satisfaction relation* $\nu \models \phi$ for a dRL formula ϕ in state ν is defined inductively and, aside from the refinement and equivalence relations, it is defined as usual in first-order modal logic for real arithmetic.

- $\nu \models (\theta_1 \leq \theta_2)$ iff $[\![\theta_1]\!]_\nu \leq [\![\theta_2]\!]_\nu$.
- $\nu \models \neg F$ iff $\nu \not\models F$, i.e. if it is not the case that $\nu \models F$.
- $\nu \models F \wedge G$ iff $\nu \models F$ and $\nu \models G$.
- $\nu \models \forall x\, F$ iff $\nu_x^d \models F$ for all $d \in \mathbb{R}$.
- $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all $\omega \in \{\nu\} \circ \rho(\alpha)$
- $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some $\omega \in \{\nu\} \circ \rho(\alpha)$
- $\boldsymbol{\nu \models \alpha \leq \beta}$ iff $\boldsymbol{\{\nu\} \circ \rho(\alpha) \subseteq \{\nu\} \circ \rho(\beta)}$

If $\nu \models \phi$, we say that ϕ holds at state ν. A formula ϕ is *valid* iff ϕ holds at all states, i.e. $\nu \models \phi$ for all $\nu \in \mathcal{S}$; we write $\models \phi$ to denote that ϕ is valid. We use ν_x^d to be the state ν with the variable x assigned to real value d.

Lemma 2. $\{\nu\} \circ \rho(\alpha) \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\alpha) \circ \rho(\gamma)$ *iff* $\nu \models [\alpha](\beta \leq \gamma)$

Proof. By the semantic definition of the box modality and refinement, $\nu \models [\alpha](\beta \leq \gamma)$ iff $\{\omega\} \circ \rho(\beta) \subseteq \{\omega\} \circ \rho(\gamma)$ for all $\omega \in \{\nu\} \circ \rho(\alpha)$. Therefore $\{\nu\} \circ \rho(\alpha) \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\alpha) \circ \rho(\gamma)$. □

Lemma 3. *If* $\nu \models [\alpha^*]\beta \leq \gamma$, *then* $\{\nu\} \circ \rho(\alpha^n) \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\alpha^n) \circ \rho(\gamma)$.

Proof. By the semantic definition of the box modality and refinement, $\nu \models [\alpha^*]\beta \leq \gamma$ iff $\{\omega\} \circ \rho(\beta) \subseteq \{\omega\} \circ \rho(\gamma)$ for all $\omega \in \{\nu\} \circ \rho(\alpha^*)$. By the semantic definition of nondeterministic repetition, $\rho(\alpha^*) = \bigcup \rho(\alpha^n)$, so $\{\nu\} \circ \rho(\alpha^n) \subseteq \{\nu\} \circ \rho(\alpha^*)$. Therefore, $\{\nu\} \circ \rho(\alpha^n) \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\alpha^n) \circ \rho(\gamma)$.

□

Soundness of $(\leq_{refl})$

$$\frac{}{\Gamma \vdash \alpha \leq \alpha, \Delta}(\leq_{refl})$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise.

By the semantic definition of refinement, we say that $\nu \models \alpha \leq \beta$ iff $\{\nu\} \circ \rho(\alpha) \subseteq \{\nu\} \circ \rho(\beta)$. For the case of $\nu \models \alpha \leq \alpha$, this holds by reflexivity of the subset relation. □

Soundness of ($\leq_{trans}$)

$$\frac{\Gamma \vdash \alpha \leq \beta, \Delta \qquad \Gamma \vdash \beta \leq \gamma, \Delta}{\Gamma \vdash \alpha \leq \gamma, \Delta}(\leq_{trans})$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. From the premise, we know that $\nu \models \alpha \leq \beta$ and $\nu \models \beta \leq \gamma$.

$$\begin{aligned} \{\nu\} \circ \rho(\alpha) &\subseteq \{\nu\} \circ \rho(\beta) && \text{by the semantic definition of } \nu \models \alpha \leq \beta && (3.5)\\ &\subseteq \{\nu\} \circ \rho(\gamma) && \text{by the semantic definition of } \nu \models \beta \leq \gamma && (3.6) \end{aligned}$$

□

Soundness of ($\leq_{antisym}$)

$$\frac{\Gamma \vdash \alpha \leq \beta, \Delta \qquad \Gamma \vdash \beta \leq \alpha, \Delta}{\Gamma \vdash \alpha = \beta, \Delta}(\leq_{antisym})$$

Proof. Recall from Fig. 3.1 that equivalence over hybrid programs is syntactically defined as $((\alpha \leq \beta) \wedge (\beta \leq \alpha))$. So proof of soundness for this rule follows immediately from the $\wedge_r$ rule. □

Soundness of (;)

$$\frac{\Gamma \vdash \alpha_1 \leq \alpha_2, \Delta \qquad \Gamma \vdash [\alpha_1](\beta_1 \leq \beta_2), \Delta}{\Gamma \vdash (\alpha_1;\beta_1) \leq (\alpha_2;\beta_2), \Delta}(;)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premises are valid, we know $\nu \models \alpha_1 \leq \alpha_2$ and $\nu \models [\alpha_1](\beta_1 \leq \beta_2)$.

$$\begin{aligned} \{\nu\} \circ \rho(\alpha_1;\beta_1) &= \{\nu\} \circ \rho(\alpha_1) \circ \rho(\beta_1) && \text{by semantic definition of } \rho(\alpha;\beta) && (3.7)\\ &\subseteq \{\nu\} \circ \rho(\alpha_1) \circ \rho(\beta_2) && \text{by } \nu \models [\alpha_1](\beta_1 \leq \beta_2) \text{ and Lemma 2} && (3.8)\\ &\subseteq \{\nu\} \circ \rho(\alpha_2) \circ \rho(\beta_2) && \text{by } \{\nu\} \circ \rho(\alpha_1) \subseteq \{\nu\} \circ \rho(\alpha_2) \text{ from left premise} && (3.9)\\ &= \{\nu\} \circ \rho(\alpha_2;\beta_2) && \text{by semantic definition of } \rho(\alpha;\beta) && (3.10) \end{aligned}$$

□

It is tempting to write the (;) rule without the $[\alpha_1]$ as such:

$$\frac{\Gamma \vdash \alpha_1 \leq \alpha_2, \Delta \qquad \Gamma \vdash \beta_1 \leq \beta_2, \Delta}{\Gamma \vdash (\alpha_1;\beta_1) \leq (\alpha_2;\beta_2), \Delta}(\textit{unsound}\ ;)$$

But this updated version of the right premise only guarantees the refinement $\beta_1 \leq \beta_2$ holds in the initial states (i.e. those which satisfy the context Γ). It makes no guarantees that $\beta_1 \leq \beta_2$

is satisfied in whatever states might be reachable through transitioning on α_1 or α_2. By dropping $[\alpha_1]$ from the right premise, it is precisely the context that causes trouble in the following counterexample to the unsound rule:

$$\dfrac{\dfrac{*}{x = 1 \vdash x := 2 \leq x := 2} \qquad \dfrac{\dfrac{*}{x = 1 \vdash x^2 = x}}{x = 1 \vdash ?\top \leq (?(x^2 = x))}?}{x = 1 \vdash (x := 2; ?\top) \leq (x := 2; ?(x^2 = x))}\mathit{unsound}\ ;$$

It is sometimes advantageous to use an alternative, weaker version of the ; rule, where we use $\Gamma \vdash [\alpha_2](\beta_1 \leq \beta_2, \Delta)$ as the second antecedent. Soundness of this rule is immediate by the $[\leq]$ rule.

Soundness of (*unloop*)

$$\dfrac{\Gamma \vdash [\alpha^*](\alpha \leq \beta), \Delta}{\Gamma \vdash \alpha^* \leq \beta^*, \Delta}(\mathit{unloop})$$

The box modality in the premise of this rule is needed for similar reasons as in the ;-rule. It ensures that only context that remains invariant throughout the evolution of α^* can be used as evidence for the proof of refinement. We prove this property by induction, showing that after n sequential runs of α, the refinement relation between α and β still holds.

This proof rule is very commonly used in dRL proofs, as it keeps in tact the structure of hybrid programs α and β, making it especially useful when the two programs are already very similar. Proving similar properties in dL, would require a loop invariant to handle the unrolling of both α^* and β^*.

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models [\alpha^*]\alpha \leq \beta$.

$$\begin{aligned}
\{\nu\} \circ \rho(\alpha^*) &= \{\nu\} \circ \bigcup_{n\in\mathbb{N}} \rho(\alpha^n) && \text{by semantic definition of } \rho(\alpha^*) && (3.11)\\
&\subseteq \{\nu\} \circ \bigcup_{n\in\mathbb{N}} \rho(\beta^n) && \text{by Proposition 1} && (3.12)\\
&= \{\nu\} \circ \rho(\beta^*) && \text{by semantic definition of } \rho(\beta^*) && (3.13)
\end{aligned}$$

□

Proposition 1. *If $\nu \models [\alpha^*]\alpha \leq \beta$, then $\{\nu\} \circ \rho(\alpha^n) \subseteq \{\nu\} \circ \rho(\beta^n)$.*

Proof. Proof is by induction on n. By definition, $\alpha^0 = ?\top = \beta^0$, so $\rho(\alpha^0) = \rho(\beta^0)$.

$$
\begin{aligned}
\{\nu\} \circ \rho(\alpha^{n+1}) &= \{\nu\} \circ \rho(\alpha^n; \alpha) && \text{by definition of } \alpha^n && (3.14)\\
&= \{\nu\} \circ \rho(\alpha^n) \circ \rho(\alpha) && \text{by semantics of ;} && (3.15)\\
&\subseteq \{\nu\} \circ \rho(\alpha^n) \circ \rho(\beta) && \text{by } \nu \models [\alpha^*]\alpha \leq \beta \text{ and Lemma 3} && (3.16)\\
&\subseteq \{\nu\} \circ \rho(\beta^n) \circ \rho(\beta) && \text{by induction hypothesis} && (3.17)\\
&= \{\nu\} \circ \rho(\beta^{n+1}) && \text{by semantics of ; and definition of } \beta^n && (3.18)
\end{aligned}
$$

□

We can prove the following useful variant of the unloop rule simply by using Proposition 2 instead of Proposition 1. While this variant is technically a weaker rule, it can be much more useful in practice (see the proof that a time-triggered system refines event-triggered in Chapter 6).

$$\frac{\Gamma \vdash [\beta^*](\alpha \leq \beta), \Delta}{\Gamma \vdash \alpha^* \leq \beta^*, \Delta}(\mathit{unloop})$$

Proposition 2. *If* $\nu \models [\beta^*]\alpha \leq \beta$*, then* $\{\nu\} \circ \rho(\alpha^n) \subseteq \{\nu\} \circ \rho(\beta^n)$.

Proof. Proof follows similarly to Proposition 1.

$$
\begin{aligned}
\{\nu\} \circ \rho(\alpha^{n+1}) &= \{\nu\} \circ \rho(\alpha^n; \alpha) && \text{by definition of } \alpha^n && (3.19)\\
&= \{\nu\} \circ \rho(\alpha^n) \circ \rho(\alpha) && \text{by semantics of ;} && (3.20)\\
&\subseteq \{\nu\} \circ \rho(\beta^n) \circ \rho(\alpha) && \text{by induction hypothesis} && (3.21)\\
&\subseteq \{\nu\} \circ \rho(\beta^n) \circ \rho(\beta) && \text{by } \nu \models [\beta^*](\alpha \leq \beta) \text{ and Lemma 3} && (3.22)\\
&= \{\nu\} \circ \rho(\beta^{n+1}) && \text{by semantics of ; and definition of } \beta^n && (3.23)
\end{aligned}
$$

□

Soundness of (?)

$$\frac{\Gamma \vdash \phi \rightarrow \psi, \Delta}{\Gamma \vdash ?\phi \leq ?\psi, \Delta}(?)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models \phi \rightarrow \psi$.

Case 1: Let $\nu \models \phi$. In this case, we also know $\nu \models \psi$, from $\nu \models \phi \rightarrow \psi$.

$$
\begin{aligned}
\{\nu\} \circ \rho(?\phi) &= \{\nu\} && \text{by } \nu \models \phi \text{ and definition of } \rho(?\phi) && (3.24)\\
&= \{\nu\} \circ \rho(?\psi) && \text{by } \nu \models \psi \text{ and definition of } \rho(?\psi) && (3.25)
\end{aligned}
$$

Case 2: Let $\nu \not\models \phi$.

$$
\begin{aligned}
\{\nu\} \circ \rho(?\phi) &= \emptyset && \text{by } \nu \not\models \phi \text{ and definition of } \rho(?\phi) && (3.26)\\
&\subseteq \{\nu\} \circ \rho(?\psi) && && (3.27)
\end{aligned}
$$

□

Soundness of $(\cup_l)$

$$\frac{\Gamma \vdash \alpha \leq \gamma \wedge \beta \leq \gamma, \Delta}{\Gamma \vdash \alpha \cup \beta \leq \gamma, \Delta}(\cup_l)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models \alpha \leq \gamma$ and $\nu \models \beta \leq \gamma$.

$$\begin{aligned}
\{\nu\} \circ \rho(\alpha \cup \beta) &= \{\nu\} \circ (\rho(\alpha) \cup \rho(\beta)) && \text{by definition of } \rho(\alpha \cup \beta) && (3.28)\\
&= (\{\nu\} \circ \rho(\alpha)) \cup (\{\nu\} \circ \rho(\beta)) && \text{by set distribution} && (3.29)\\
&\subseteq \{\nu\} \circ \rho(\gamma) && \text{see below} && (3.30)
\end{aligned}$$

We get (3.30) by $\{\nu\} \circ \rho(\alpha) \subseteq \{\nu\} \circ \rho(\gamma)$ (from the semantic definition of $\nu \models \alpha \leq \gamma$) and $\{\nu\} \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\gamma)$ (from the semantic definition of $\nu \models \beta \leq \gamma$).

□

Soundness of $(\cup_r)$

$$\frac{\Gamma \vdash \alpha \leq \beta \vee \alpha \leq \gamma, \Delta}{\Gamma \vdash \alpha \leq \beta \cup \gamma, \Delta}(\cup_r)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know that $\nu \models \alpha \leq \beta$ or $\nu \models \alpha \leq \gamma$.
Case 1: Let $\nu \models \alpha \leq \beta$.

$$\begin{aligned}
\{\nu\} \circ \rho(\alpha) &\subseteq \{\nu\} \circ \rho(\beta) && \text{by the semantic definition of } \nu \models \alpha \leq \beta && (3.31)\\
&\subseteq \{\nu\} \circ (\rho(\beta) \cup \rho(\gamma)) && \text{by set union} && (3.32)\\
&= \{\nu\} \circ \rho(\beta \cup \gamma) && \text{by definition of } \rho(\beta \cup \gamma) && (3.33)
\end{aligned}$$

Case 2: Let $\nu \models \alpha \leq \gamma$. This case follows similarly to Case 1. □

Soundness of $(:= *)$

$$\frac{}{\Gamma \vdash (x := \theta) \leq (x := *), \Delta}(:= *)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise.

$$\begin{aligned}
\{\nu\} \circ \rho(x := \theta) &= \{\mu : \text{where } \mu = \nu \text{ except that } [\![x]\!]_\mu = [\![\theta]\!]_\nu\} && (3.34)\\
&\subseteq \{\omega : \text{where } \omega = \nu \text{ except on the value of } x\} && (3.35)\\
&= \{\nu\} \circ \rho(x := *) && (3.36)
\end{aligned}$$

□

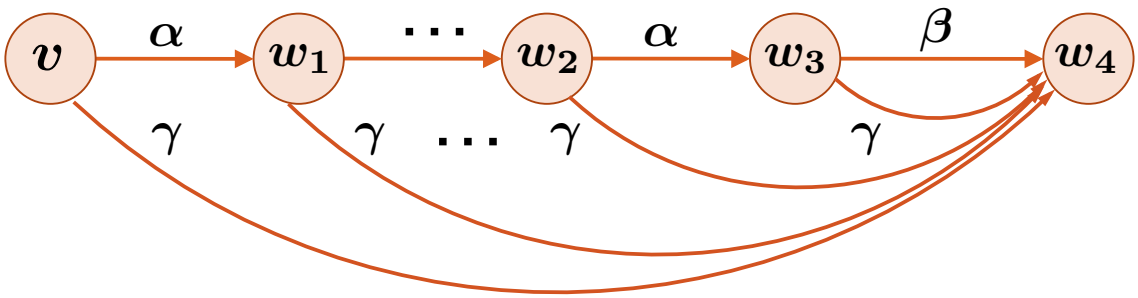

Figure 3.6: A graphical representation of rule $loop_l$.

Soundness of ($loop_l$)

This proof rule, and its partner $loop_r$, is based on KAT proof rules for handling loops. However, to adapt these proof rules to dRL, we have to be very careful in how we handle the context. Notice that it is crucial for soundness in this rule that we add a $[\alpha^*]$ to both the left and right premise. The reason for this becomes clear when comparing Proposition 3 and Proposition 4. When α^* is on the left, as in this rule, the refinement allows $\alpha;\beta$ to collapse into γ, but only after some number of α^n have executed. However, when α^* is on the right, as in the $loop_r$ rule, that collapse happens first.

In Fig. 3.6 we see a graphical representation of a possible state-transition diagram for hybrid programs $\alpha^*;\beta$ and γ. From the right premise of $loop_r$, β refines γ, for all states reachable through α^* (here represented as ω_3 with γ-successor ω_4), so we know that there is an execution of hybrid program γ such that ω_4 is reachable from ω_3 directly. And from the left premise of $loop_r$, we inductively know that the remaining transitions can also be reached through γ, since $\alpha;\gamma \leq \gamma$ for all states reachable from α^*.

$$\frac{\Gamma \vdash [\alpha^*](\alpha;\gamma) \leq \gamma, \Delta \qquad \Gamma \vdash [\alpha^*]\beta \leq \gamma, \Delta}{\Gamma \vdash [\alpha^*;\beta \leq \gamma, \Delta}(loop_l)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models [\alpha^*](\alpha;\gamma) \leq \gamma$ and $\nu \models [\alpha^*]\beta \leq \gamma$.

$$\begin{aligned}
\{\nu\} \circ \rho(\alpha^*;\beta) &= \{\nu\} \circ \rho(\alpha^*) \circ \rho(\beta) && \text{by the definition of } \rho(\alpha;\beta) && (3.37)\\
&\subseteq \{\nu\} \circ \rho(\alpha^*) \circ \rho(\gamma) && \text{by Lemma 2 and } \nu \models [\alpha^*]\beta \leq \gamma && (3.38)\\
&= \{\nu\} \circ \Big(\bigcup_{n\in\mathbb{N}} \rho(\alpha^n)\Big) \circ \rho(\gamma) && \text{by definition of } \rho(\alpha^*) && (3.39)\\
&= \bigcup_{n\in\mathbb{N}} \Big(\{\nu\} \circ \rho(\alpha^n) \circ \rho(\gamma)\Big) && \text{by set distribution} && (3.40)\\
&\subseteq \{\nu\} \circ \rho(\gamma) && \text{by Proposition 3 and } \nu \models [\alpha^*](\alpha;\gamma) \leq \gamma && (3.41)
\end{aligned}$$

□

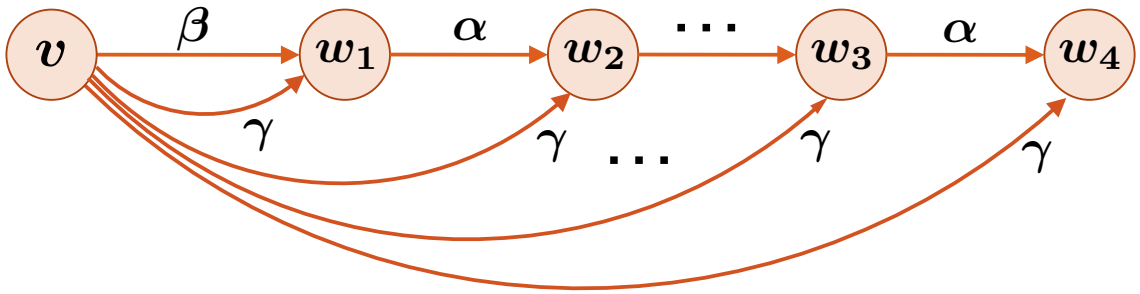

Figure 3.7: A graphical representation of rule $loop_r$.

Proposition 3. *For all* $n \in \mathbb{N}$, $\{\nu\} \circ \rho(\alpha^n) \circ \rho(\gamma) \subseteq \{\nu\} \circ \rho(\gamma)$ *iff* $\nu \models [\alpha^*](\alpha;\gamma) \leq \gamma$.

Proof. Proof by induction on n.
When $n = 0$, $\alpha^0 = ?\top$. Therefore $\{\nu\} \circ \rho(\alpha^0) \circ \rho(\gamma) = \{\nu\} \circ \rho(?\top) \circ \rho(\gamma) = \{\nu\} \circ \rho(\gamma)$.

$$
\begin{aligned}
\{\nu\} \circ \rho(\alpha^{n+1}) \circ \rho(\gamma) &= \{\nu\} \circ \rho(\alpha^n) \circ \rho(\alpha) \circ \rho(\gamma) && \text{by definition of } \rho(\alpha^{n+1}) && (3.42)\\
&= \{\nu\} \circ \rho(\alpha^n) \circ \rho(\alpha;\gamma) && \text{by definition of } \rho(\alpha;\gamma) && (3.43)\\
&\subseteq \{\nu\} \circ \rho(\alpha^n) \circ \rho(\gamma) && \text{by Lemma 3 and } \nu \models [\alpha^*](\alpha;\gamma) \leq \gamma && (3.44)\\
&\subseteq \{\nu\} \circ \rho(\gamma) && \text{by induction hypothesis} && (3.45)
\end{aligned}
$$

□

Soundness of ($loop_r$)

This proof rule is interesting because, in contrast to $loop_l$, it does *not* require modalities in the premises. The reason for this is clearly shown in the graphical representation of a possible state-transition diagram for programs $\beta;\alpha^*$ and γ illustrated in Fig. 3.7. We can see the collapse of β and α^* into γ happens from the front, rather than from the end of the hybrid program as in $loop_l$.

$$\frac{\Gamma \vdash \beta \leq \gamma, \Delta \qquad \Gamma \vdash (\gamma;\alpha) \leq \gamma, \Delta}{\Gamma \vdash \beta;\alpha^* \leq \gamma, \Delta}(loop_r)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models \beta \leq \gamma$ and $\nu \models (\gamma;\alpha) \leq \gamma$. (Notice that the premise for $loop_r$ is slightly different from

the premise for $loop_l$.)

$$\begin{aligned}
\{\nu\} \circ \rho(\beta; \alpha^*) &= \{\nu\} \circ \rho(\beta) \circ \rho(\alpha^*) && \text{by the definition of } \rho(\alpha;\beta) && (3.46)\\
&\subseteq \{\nu\} \circ \rho(\gamma) \circ \rho(\alpha^*) && \text{by } \{\nu\} \circ \rho(\beta) \subseteq \{\nu\} \circ \rho(\gamma) \text{ from } \nu \vDash \beta \leq \gamma && (3.47)\\
&= \{\nu\} \circ \rho(\gamma) \circ \Big(\bigcup_{n\in\mathbb{N}} \rho(\alpha^n)\Big) && \text{by definition of } \rho(\alpha^*) && (3.48)\\
&= \bigcup_{n\in\mathbb{N}} \Big(\{\nu\} \circ \rho(\gamma) \circ \rho(\alpha^n)\Big) && \text{by set distribution} && (3.49)\\
&\subseteq \{\nu\} \circ \rho(\gamma) && \text{by Proposition 4 and } \nu \vDash (\gamma;\alpha) \leq \gamma && (3.50)
\end{aligned}$$

□

Proposition 4. *For all $n \in \mathbb{N}$, $\{\nu\} \circ \rho(\gamma) \circ \rho(\alpha^n) \subseteq \{\nu\} \circ \rho(\gamma)$ iff $\nu \vDash (\gamma;\alpha) \leq \gamma$.*

Proof. Proof by induction on n.

$$\begin{aligned}
\{\nu\} \circ \rho(\gamma) \circ \rho(\alpha^{n+1}) &= \{\nu\} \circ \rho(\gamma) \circ \rho(\alpha) \circ \rho(\alpha^n) && \text{by definition of } \rho(\alpha^{n+1}) && (3.51)\\
&= \{\nu\} \circ \rho(\gamma;\alpha) \circ \rho(\alpha^n) && \text{by definition of } \rho(\gamma;\alpha) && (3.52)\\
&\subseteq \{\nu\} \circ \rho(\gamma) \circ \rho(\alpha^n) && \text{by } \{\nu\} \circ \rho(\gamma;\alpha) \subseteq \{\nu\} \circ \rho(\gamma) && (3.53)\\
&\subseteq \{\nu\} \circ \rho(\gamma) && \text{by induction hypothesis} && (3.54)
\end{aligned}$$

In line (3.53), we know that $\{\nu\}\circ\rho(\gamma;\alpha) \subseteq \{\nu\}\circ\rho(\gamma)$ from the semantic definition of $\nu \vDash (\gamma;\alpha) \leq \gamma$.
□

Soundness of $[\leq]$

$$\frac{\Gamma \vdash [\beta]\phi, \Delta \qquad \Gamma \vdash \alpha \leq \beta, \Delta}{\Gamma \vdash [\alpha]\phi, \Delta}([\leq])$$

Proof. Let ν be an arbitrary state such that $\nu \vDash G$ for all $G \in \Gamma$ and such that $\nu \nvDash D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \vDash [\beta]\phi$ and $\nu \vDash \alpha \leq \beta$.

Since $\nu \vDash \alpha \leq \beta$, we know that $\{\nu\} \circ \rho(\alpha) \subseteq \{\nu\} \circ \rho(\beta)$, by the semantic definition of refinement. From $\nu \vDash [\beta]\phi$, we know that $\omega \vDash \phi$ for all states $\omega \in \{\nu\} \circ \rho(\beta)$. Therefore, for all states $\omega \in \{\nu\} \circ \rho(\alpha)$, we know that $\omega \vDash \phi$.

By the semantic definition of box modality, $\nu \vDash [\alpha]\phi$ iff $\omega \vDash \phi$ for all states $\omega \in \{\nu\} \circ \rho(\alpha)$.

□

Soundness of $\langle\leq\rangle$

$$\frac{\Gamma \vdash \langle\alpha\rangle\phi, \Delta \qquad \Gamma \vdash \alpha \leq \beta, \Delta}{\Gamma \vdash \langle\beta\rangle\phi, \Delta}(\langle\leq\rangle)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models \langle\alpha\rangle\phi$ and $\nu \models \alpha \leq \beta$.

From $\nu \models \langle\alpha\rangle\phi$, we know that there exists an ω such that $\omega \in \{\nu\} \circ \rho(\alpha)$ and $\omega \models \phi$. Since $\nu \models \alpha \leq \beta$, we know that $\{\nu\} \circ \rho(\alpha) \subseteq \{\nu\} \circ \rho(\beta)$, by the semantic definition of refinement. Therefore, since $\omega \in \{\nu\} \circ \rho(\alpha)$, we also know that $\omega \in \{\nu\} \circ \rho(\beta)$.

By the semantic definition of diamond modality, $\nu \models \langle\beta\rangle\phi$ iff there exists an ω such that $\omega \models \phi$ and $\omega \in \{\nu\} \circ \rho(\alpha)$.

□

Soundness of (*DC*)

$$\frac{\Gamma \vdash [x' = \theta \,\&\, H_1]H_2, \Delta}{\Gamma \vdash (x' = \theta \,\&\, H_1) = (x' = \theta \,\&\, H_1 \wedge H_2), \Delta}\ (DC)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models [x' = \theta \,\&\, H_1]H_2$.

In the following, we use shorthand notation for the transition semantics of $x' = \theta \,\&\, H$. For example, let $\varphi : [0, r] \to \mathcal{S}$ be a solution of any duration r to $x' = \theta$, where $\varphi(0) = \nu$.

$$\begin{aligned}
\{\nu\} \circ \rho(x' = \theta \,\&\, H_1) &= \{\varphi(r) : \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H_1 \text{ for all } 0 \leq t \leq r\} && (3.55)\\
&= \{\varphi(r) : \varphi(t) \models x' = \theta, \varphi(t) \models H_1, \text{ and } \varphi(t) \models H_2 \text{ for all } 0 \leq t \leq r\} && (3.56)\\
&= \{\varphi(r) : \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H_1 \wedge H_2 \text{ for all } 0 \leq t \leq r\} && (3.57)\\
&= \{\nu\} \circ \rho(x' = \theta \,\&\, H_1 \wedge H_2) && (3.58)
\end{aligned}$$

We know (3.55) by the transition semantics of $x' = \theta \,\&\, H_1$. In (3.56), we know that $\varphi(t) \models H_2$ already holds for all $0 \leq t \leq r$. This is because, from $\nu \models [x' = \theta \,\&\, H_1]H_2$, we know that for all $\varphi(t) \in \{\nu\} \circ \rho(x' = \theta \,\&\, H_1)$ it must be the case that $\varphi(t) \models H_2$, by the semantic definition of the box modality. And we know that $\varphi(t) \in \{\nu\} \circ \rho(x' = \theta \,\&\, H_1)$ for all $0 \leq t \leq r$, since $\varphi(t)$ is a solution of duration r.

□

Soundness of (*DR*)

$$\frac{\Gamma \vdash \forall x\,(H_1 \to H_2), \Delta}{\Gamma \vdash (x' = \theta \,\&\, H_1) \leq (x' = \theta \,\&\, H_2), \Delta}(DR)$$

Proof. Let ν be an arbitrary state such that $\nu \models G$ for all $G \in \Gamma$ and such that $\nu \not\models D$ for all $D \in \Delta$, since the sequent is trivially valid otherwise. Therefore, when the premise is valid, we know $\nu \models \forall x\,(H_1 \to H_2)$.

In the following, we use shorthand notation for the transition semantics of $x' = \theta \,\&\, H$. For example, let $\varphi : [0, r] \to \mathcal{S}$ be a solution of any duration r to $x' = \theta$, where $\varphi(0) = \nu$.

$$\{\nu\} \circ \rho(x' = \theta \,\&\, H_1) = \{\varphi(r) \ : \ \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H_1 \text{ for all } 0 \leq t \leq r\} \tag{3.59}$$

$$\subseteq \{\varphi(r) \ : \ \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H_2 \text{ for all } 0 \leq t \leq r\} \tag{3.60}$$

$$= \{\nu\} \circ \rho(x' = \theta \,\&\, H_2) \tag{3.61}$$

We know (3.59) by the transition semantics of $x' = \theta \,\&\, H_1$.

We know that $\{\varphi(t) : 0 \leq t \leq r\} \subseteq \{\nu_x^d : d \in \mathbb{R}^n\}$ by the bound variable effect [65] and because $\varphi(0) = \nu$. We also know that $\nu \models \forall x\,(H_1 \to H_2)$ iff for all $d \in \mathbb{R}^n$, $\nu_x^d \models H_1$ implies $\nu_x^d \models H_2$. Therefore, $\{\varphi(t) : \varphi(t) \models H_1 \text{ and } 0 \leq t \leq r\} \subseteq \{\varphi(t) : \varphi(t) \models H_2 \text{ and } 0 \leq t \leq r\}$ and (3.60) holds. Finally, (3.61) is by the transition semantics of $x' = \theta \,\&\, H_2$.

□

Soundness of (*MDF*)

$$\frac{\Gamma \vdash \forall x \Big(\theta_1 \|\theta_2\| = \theta_2 \|\theta_1\| \wedge (\|\theta_1\|^2 = 0 \leftrightarrow \|\theta_2\|^2 = 0)\Big), \Delta}{\Gamma \vdash (x' = \theta_1) = (x' = \theta_2), \Delta}\ (MDF)^6$$

Proof. Let $\nu \models \forall x \Big(\theta_1 \|\theta_2\| = \theta_2 \|\theta_1\| \wedge (\|\theta_1\|^2 = 0 \leftrightarrow \|\theta_2\|^2 = 0)\Big)$.

Let $y_1(t)$ and $y_2(t)$ be real-valued functions defined on the domain $[0, \infty)$ that solve the initial value problems $\dfrac{\mathrm{d}y_1(t)}{\mathrm{d}t} = [\![\theta_1]\!]_\nu$, $y_1(0) = [\![x]\!]_\nu$ and $\dfrac{\mathrm{d}y_2(t)}{\mathrm{d}t} = [\![\theta_2]\!]_\nu$, $y_2(0) = [\![x]\!]_\nu$. Notice that $[\![\theta_1]\!]_\nu$ and $[\![\theta_2]\!]_\nu$ are constant throughout the evolution of the differential equation because we restrict θ to be a term that does not contain any variables of x. We know $y_1(t)$ and $y_2(t)$ exist and are globally defined because constant differential equation systems have closed-form polynomial solutions in t [66, §10.VII]. For the same reason, we also know that $y_1(t)$ and $y_2(t)$ are uniquely defined, i.e. for each differential equation system, each state ν, and each duration $r \geq 0$, there is at most one solution $\varphi : [0, r] \to \mathcal{S}$ satisfying the conditions of Case 3 of Definition 3. More specifically, we know that the solutions to the differential equations are explicitly defined by $y_i(t) = [\![\theta_i]\!]_\nu t + [\![x]\!]_\nu$. Then for our proof of soundness it suffices to show:

$$\forall_{s \geq 0} \exists_{r \geq 0}\ y_1(r) = y_2(s) \tag{3.62}$$

Case 1: Assume $\|[\![\theta_1]\!]_\nu\| = 0$. Then y_1 is at an equilibrium point and the solution to the differential equation is the constant function $y_1(t) = y_1(0) = [\![x]\!]_\nu$.

By the premise we know that $\|[\![\theta_1]\!]_\nu\| = 0$ iff $\|[\![\theta_2]\!]_\nu\| = 0$. Then y_2 is also at an equilibrium point and the solution to the differential equation is the constant function $y_2(t) = y_2(0) = [\![x]\!]_\nu$. So we can easily see that (3.62) holds from the following:

$$\forall_{s \geq 0, r \geq 0}\ y_1(r) = [\![x]\!]_\nu = y_2(s) \tag{3.63}$$

[6] We require that this rule be defined only when no variables of x occur in θ.

Case 2: Assume $\|[\![\theta_1]\!]_\nu\| > 0$. Define $\lambda(s) : [0,\infty) \to [0,\infty)$ to be the continuous, real-valued function $\lambda(s) = \frac{\|[\![\theta_2]\!]_\nu\|}{\|[\![\theta_1]\!]_\nu\|}s$. We know that $\lambda(s)$ is always defined on its domain, since $\|[\![\theta_1]\!]_\nu\| > 0$. We also know that $\lambda(s) \geq 0$, since the domain of s is $[0,\infty)$. By the premise and since x does not occur in θ, we know that $[\![\theta_1]\!]_\nu\|[\![\theta_2]\!]_\nu\| = [\![\theta_2]\!]_\nu\|[\![\theta_1]\!]_\nu\|$. Therefore,

$$
\begin{aligned}
y_1(\lambda(s)) &= [\![\theta_1]\!]_\nu \lambda(s) && \text{using the closed-form solution of } y_1(t) && (3.64)\\
&= [\![\theta_1]\!]_\nu \frac{\|[\![\theta_2]\!]_\nu\|}{\|[\![\theta_1]\!]_\nu\|}s && \text{by the definition of } \lambda(s) && (3.65)\\
&= [\![\theta_2]\!]_\nu \frac{\|[\![\theta_1]\!]_\nu\|}{\|[\![\theta_1]\!]_\nu\|}s && \text{by } [\![\theta_1]\!]_\nu\|[\![\theta_2]\!]_\nu\| = [\![\theta_2]\!]_\nu\|[\![\theta_1]\!]_\nu\| && (3.66)\\
&= [\![\theta_2]\!]_\nu s && && (3.67)\\
&= y_2(s) && \text{using the closed-form solution of } y_2(t) && (3.68)
\end{aligned}
$$

□

3.6.2 Equivalence Proofs from dL Axioms

Instead of continuing to prove dRL proof rules semantically as in Section 3.6.1, we would like to leverage dL to do the heavy lifting on our remaining soundness proofs. To do that, we first prove Lemma 4, which allows us to translate any equivalence properties over hybrid programs in dRL into modality properties in dL, allowing us to use the dL proof calculus to prove soundness for each equivalence rule.

Lemma 4. *If $[\alpha]\phi \leftrightarrow [\beta]\phi$ is a sound axiom schema, then $\alpha = \beta$.*

Proof. Assume $\nu \not\models \alpha = \beta$ for the sake of contradiction. Without loss of generality, let $\nu \not\models \alpha \leq \beta$. This implies that $\exists\omega.(\nu,\omega) \in \rho(\alpha) \setminus \rho(\beta)$. Notice that for all states μ where $\mu = \omega$ on $BV(\alpha) \cup BV(\beta)$, we know that $(\nu,\mu) \notin \rho(\beta)$ by the bound variable effect [65]. Let $\{x_i\} \coloneqq BV(\alpha) \cup BV(\beta)$. Note that the cardinality of $\{x_i\}$ is finite, since all hybrid programs must be finite in length and therefore in the number of bound variables. Let $r_i \coloneqq \omega(x_i)$, so $\omega \models \bigwedge r_i = x_i$. Assume that we have temporarily extended the language of dRL to include constant variables for all real numbers. Then,

$\nu \models \langle\alpha\rangle \bigwedge r_i = x_i$ since $(\nu,\omega) \in \rho(\alpha)$ and $\omega \models \bigwedge r_i = x_i$, but
$\nu \not\models \langle\beta\rangle \bigwedge r_i = x_i$ since $(\nu,\mu) \notin \rho(\beta)$ for all $\mu \models \bigwedge r_i = x_i$.

Therefore, we have a contradiction to $[\alpha]\phi \leftarrow [\beta]\phi$, and thus a contradiction to $[\alpha]\phi \leftrightarrow [\beta]\phi$. □

Soundness of $(\cup_{assoc})$

$$\frac{}{\Gamma \vdash \alpha \cup (\beta \cup \gamma) = (\alpha \cup \beta) \cup \gamma, \Delta}(\cup_{assoc})$$

Proof. We show that $\alpha \cup (\beta \cup \gamma) = (\alpha \cup \beta) \cup \gamma$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash ([\alpha]\phi \wedge ([\beta]\phi \wedge [\gamma]\phi)) \leftrightarrow (([\alpha]\phi \wedge [\beta]\phi) \wedge [\gamma]\phi), \Delta}\,{\scriptstyle \wedge_{assoc},\, refl}}{\Gamma \vdash ([\alpha]\phi \wedge [\beta \cup \gamma]\phi) \leftrightarrow ([\alpha \cup \beta]\phi \wedge [\gamma]\phi), \Delta}\,{\scriptstyle [\cup]}}{\Gamma \vdash [\alpha \cup (\beta \cup \gamma)]\phi \leftrightarrow [(\alpha \cup \beta) \cup \gamma]\phi, \Delta}\,{\scriptstyle [\cup]}$$

□

Soundness of $(\cup_{comm})$

$$\dfrac{}{\Gamma \vdash \alpha \cup \beta = \beta \cup \alpha, \Delta}(\cup_{comm})$$

Proof. We show that $\alpha \cup \beta = \beta \cup \alpha$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash ([\alpha]\phi \wedge [\beta]\phi) \leftrightarrow ([\alpha]\phi \wedge [\beta]\phi), \Delta}\,{\scriptstyle refl}}{\Gamma \vdash ([\alpha]\phi \wedge [\beta]\phi) \leftrightarrow ([\beta]\phi \wedge [\alpha]\phi), \Delta}\,{\scriptstyle \wedge_{comm}}}{\Gamma \vdash [\alpha \cup \beta]\phi \leftrightarrow [\beta \cup \alpha]\phi, \Delta}\,{\scriptstyle [\cup]}$$

□

Soundness of $(\cup_{id})$

$$\dfrac{}{\Gamma \vdash \alpha \cup ?\bot = \alpha, \Delta}(\cup_{id})$$

Proof. We show that $\alpha \cup ?\bot = \alpha$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma, ([\alpha]\phi \wedge [?\bot]\phi) \vdash [\alpha]\phi, \Delta}\,{\scriptstyle \wedge_l,\, ax} \qquad \dfrac{\dfrac{*}{\Gamma, [\alpha]\phi \vdash [\alpha]\phi, \Delta}\,{\scriptstyle ax} \qquad \dfrac{\dfrac{*}{\Gamma, [\alpha]\phi, \bot \vdash \phi, \Delta}\,{\scriptstyle \bot_l}}{\Gamma, [\alpha]\phi \vdash [?\bot]\phi, \Delta}\,{\scriptstyle [?],\, \rightarrow_r}}{\Gamma, [\alpha]\phi \vdash ([\alpha]\phi \wedge [?\bot]\phi), \Delta}\,{\scriptstyle \wedge_r}}{\Gamma \vdash ([\alpha]\phi \wedge [?\bot]\phi) \leftrightarrow [\alpha]\phi, \Delta}\,{\scriptstyle \wedge_r,\, \leftarrow_r,\, \rightarrow_r}}{\Gamma \vdash [\alpha \cup ?\bot]\phi \leftrightarrow [\alpha]\phi, \Delta}\,{\scriptstyle [\cup]}$$

□

Soundness of $(\cup_{idemp})$

$$\frac{}{\Gamma \vdash (\alpha \cup \alpha) = \alpha, \Delta}(\cup_{idemp})$$

Proof. We show that $(\alpha \cup \alpha) = \alpha$ using Lemma 4.

$$\dfrac{\dfrac{*}{\Gamma \vdash ([\alpha]\phi \wedge [\alpha]\phi) \leftrightarrow [\alpha]\phi, \Delta}}{\Gamma \vdash [\alpha \cup \alpha]\phi \leftrightarrow [\alpha]\phi, \Delta}\,[\cup]$$

□

Soundness of $(;_{assoc})$

$$\frac{}{\Gamma \vdash \alpha;(\beta;\gamma) = (\alpha;\beta);\gamma, \Delta}(;_{assoc})$$

Proof. We show that $\alpha;(\beta;\gamma) = (\alpha;\beta);\gamma$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash [\alpha][\beta][\gamma]\phi \leftrightarrow [\alpha][\beta][\gamma]\phi, \Delta}\,refl}{\Gamma \vdash [\alpha][(\beta;\gamma)]\phi \leftrightarrow [(\alpha;\beta)][\gamma]\phi, \Delta}\,[;]}{\Gamma \vdash [\alpha;(\beta;\gamma)]\phi \leftrightarrow [(\alpha;\beta);\gamma]\phi, \Delta}\,[;]$$

□

Soundness of $(;_{id-l})$

$$\frac{}{\Gamma \vdash (?\top;\alpha) = \alpha, \Delta}(;_{id-l})$$

Proof. We show that $(?\top;\alpha) = \alpha$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash (\top \rightarrow [\alpha]\phi) \leftrightarrow [\alpha]\phi, \Delta}\,refl}{\Gamma \vdash [?\top][\alpha]\phi \leftrightarrow [\alpha]\phi, \Delta}\,[?]}{\Gamma \vdash [?\top;\alpha]\phi \leftrightarrow [\alpha]\phi, \Delta}\,[;]$$

□

The proof of soundness for $(;_{id-r})$ follows similarly.

Soundness of (*dist-l*)

$$\frac{}{\Gamma \vdash \alpha;(\beta \cup \gamma) = ((\alpha;\beta) \cup (\alpha;\gamma)), \Delta}(\textit{dist-l})$$

Proof. We show that $\alpha;(\beta \cup \gamma) = ((\alpha;\beta) \cup (\alpha;\gamma))$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash [\alpha][\beta]\phi \wedge [\alpha][\gamma]\phi \leftrightarrow [\alpha][\beta]\phi \wedge [\alpha][\gamma]\phi, \Delta}\,\textit{refl}}{\Gamma \vdash [\alpha]([\beta]\phi \wedge [\gamma]\phi) \leftrightarrow [\alpha][\beta]\phi \wedge [\alpha][\gamma]\phi, \Delta}\,[]\ \textit{split}}{\Gamma \vdash [\alpha][\beta \cup \gamma]\phi \leftrightarrow [\alpha;\beta]\phi \wedge [\alpha;\gamma]\phi, \Delta}\,[\cup], [;]}{\Gamma \vdash [\alpha;(\beta \cup \gamma)]\phi \leftrightarrow [((\alpha;\beta) \cup (\alpha;\gamma))]\phi, \Delta}\,[;], [\cup]$$

□

The proof of soundness for (*dist-r*) follows similarly.

Soundness of ($;_{annih-r}$)

$$\frac{}{\Gamma \vdash \alpha; ?\bot = ?\bot, \Delta}(;_{annih-r})$$

Proof. We show that $\alpha; ?\bot = ?\bot$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash [\alpha]\top, \Delta}\,[]gen}{\Gamma \vdash [\alpha](\bot \rightarrow \phi) \leftrightarrow (\bot \rightarrow \phi), \Delta}\,\rightarrow, \leftrightarrow}{\Gamma \vdash [\alpha][?\bot]\phi \leftrightarrow [?\bot]\phi, \Delta}\,[?]}{\Gamma \vdash [\alpha; ?\bot]\phi \leftrightarrow [?\bot]\phi, \Delta}\,[;]$$

□

The proof of soundness for ($;_{annih-l}$) follows similarly.

Soundness of ($unroll_l$)

$$\frac{}{\Gamma \vdash (?\top \cup (\alpha;\alpha^*)) = \alpha^*, \Delta}(\textit{unroll}_l)$$

Proof. We show that $(?\top \cup (\alpha;\alpha^*)) = \alpha^*$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash (\phi \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,\textit{refl}}{\Gamma \vdash ((\top \rightarrow \phi) \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,\rightarrow}{\Gamma \vdash ([?\top]\phi \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,[?]}{\Gamma \vdash [(?\top \cup (\alpha;\alpha^*))]\phi \leftrightarrow [\alpha^*]\phi, \Delta}\,[\cup], [^*n]$$

□

Proof. We show that $(\alpha; ?T) = \alpha$ using Lemma 4.

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\Gamma \vdash (\phi \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,\mathit{refl}}{\Gamma \vdash ((\top \rightarrow \phi) \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,\rightarrow}{\Gamma \vdash ([?\top]\phi \wedge [\alpha;\alpha^*]\phi) \leftrightarrow (\phi \wedge [\alpha;\alpha^*]\phi), \Delta}\,[?]}{\Gamma \vdash [(?\top \cup (\alpha;\alpha^*))]\phi \leftrightarrow [\alpha^*]\phi, \Delta}\,[\cup], [^*n]$$

□

Chapter 4

Distributed Car Control System

Car safety measures can be most effective when the cars on a street coordinate their control actions using distributed cooperative control. While each car optimizes its navigation planning locally to ensure the driver reaches his destination, all cars coordinate their actions in a distributed way in order to minimize the risk of safety hazards and collisions. These systems control the physical aspects of car movement using cyber technologies like local and remote sensor data and distributed V2V and V2I communication. They are thus cyber-physical systems. In this chapter, we consider a distributed car control system that is inspired by the ambitions of the California PATH project, the CICAS system, SAFESPOT and PReVENT initiatives. We develop a formal model of a distributed car control system in which every car is controlled by adaptive cruise control. One of the major technical difficulties is that faithful models of distributed car control have both distributed systems and hybrid systems dynamics. They form distributed hybrid systems, which makes them very challenging for verification. In a formal proof system, we verify that the control model satisfies its main safety objective and guarantees collision freedom for arbitrarily many cars driving on a street, even if new cars enter the lane from on-ramps or multi-lane streets. The system we present is in many ways one of the most complicated cyber-physical systems that has ever been fully verified formally.

4.1 Introduction

Because of its societal relevance, numerous parts of car control have been studied before [28–45]. Major initiatives have been devoted to developing next generation individual ground transportation solutions, including the California PATH project, the SAFESPOT and PReVENT initiatives, the CICAS-V system, and many others. Chang et al. [28], for instance, propose CICAS-V in response to a report that crashes at intersections in the US cost $97 Billion in the year 2000. The promise is tempting. Current uncontrolled car traffic is inefficient and has too many safety risks, which are caused, e.g., by traffic jams behind curves, reduced vision at night, inappropriate reactions to difficult driving conditions, or sleepy drivers. Next generation car control aims to solve these problems by using advanced sensing, wireless V2V (vehicle to vehicle) and V2I (vehicle to roadside infrastructure) communication, and (semi)automatic driver assistance technology that prevents accidents and increases economical and ecological efficiency.

Yet, there are several challenges that still need to be solved to make next generation car control a reality. The most interesting challenge for us is that it only makes sense to introduce any of these systems after its correct functioning and reliability has been ensured. Otherwise, the system might do more harm than good. This is the formal verification problem for distributed car control, which we consider in this chapter.

What makes this problem particularly exciting is its practical relevance. What makes it particularly challenging is its complicated dynamics. Distributed car control follows a hybrid dynamics, because cars move continuously along differential equations and their behavior is affected by discrete control decisions like when and how strongly to brake or to accelerate and to steer. It is in the very nature of distributed car control, however, to go beyond that with *distributed* traffic agents that interact by local sensing, broadcast communication, remote sensor data, or cooperative networked control decisions. This makes distributed car control systems prime examples of what are called *distributed hybrid systems*. In fact, because they form distributed cyber-physical multi-agent systems, the resulting systems are distributed hybrid systems regardless of whether they are built using explicitly distributed V2V and V2I network communication infrastructure or just rely on the distributed effects of sensor readings about objects traveling at remote locations (e.g., laser-range sensors measuring the distance to the car in front).

Cars reach maneuvering decisions locally in a distributed way. Is the global dynamics that emerges from the various local choices safe? What can a car assume about other cars in its maneuver planning? How do we ensure that multiple maneuvers that make sense locally do not cause conflicts or collisions globally? Formal verification of distributed hybrid systems had been an essentially unsolved challenge until recently [67].

Our main contribution is that we develop a distributed car control system and a formal proof that this system is collision-free for arbitrarily many cars, even when new cars enter or leave a multi-lane highway with arbitrarily many lanes. Another contribution is that we develop a proof structure that is strictly modular. We reduce the proof to modular stages that can be verified without the details in lower levels of abstraction. We believe the principles behind our modular structure and verification techniques are useful for other systems beyond the automotive domain. Further contributions are:

- This is the first case study in distributed hybrid systems to be verified with a generic and systematic verification approach that is not specific to the particular problem.
- We identify a simple invariant that all cars have to obey and show that it is sufficient for safety, even for emergent behavior of multiple distributed car maneuvers.
- We identify generic and static constraints on the input output parameters that any controller must obey to ensure that cars always stay safe.
- We demonstrate the feasibility of distributed hybrid systems verification.

4.2 Related Work

Car control is a deep area that has been studied by a number of different communities. The societal relevance of vehicle cooperation for CICAS intersection collision avoidance [38] and for automated highway systems [32, 35] has been emphasized. Horowitz et al. [37] proposed a lane

change maneuver within platoons. Varaiya [40] outlines the key features of an IVHS (Intelligent Vehicle/Highway System). A significant amount of work has been done in the pioneering California PATH Project. Our work is strongly inspired by these systems, but it goes further and sets the groundwork for the modeling and formal verification of their reliability and safety even in distributed car control.

Dao et al. [30, 31] developed an algorithm and model for lane assignment. Their simulations suggest [30] that traffic safety can be enhanced if vehicles are organized into platoons, as opposed to having random space between them. Our approach considers an even more general setting: we not only verify safety for platoon systems, but also when cars are driving on a lane without following platooning controllers. Hall et al. [33] also used simulations to find out what is the best strategy of maximizing traffic throughput. Chee et al. [42] showed that lane change maneuvers can be achieved in automated highway systems using the signals available from on-board sensors. Jula et al. [36] used simulations to study the conditions under which accidents can be avoided during lane changes and merges. They have only tested safety partially. In contrast to [30, 31, 33, 36, 42], we do not use simulation but formal verification to validate our hypotheses.

Hsu et al. [34] propose a control system for IVHS that organizes traffic in platoons of closely spaced vehicles. They specify this system by interacting finite state machines. Those cannot represent the actual continuous movement of the cars. We use differential equations to model the continuous dynamics of the vehicles and thus consider more realistic models of the interactions between vehicles, their control, and their movement.

Stursberg et al. [39] applied counterexample-guided verification to a cruise control system with two cars on one lane. Their technique can not scale to an arbitrary number of cars. Althoff et al. [44] use reachability analysis to prove the safety of evasive maneuvers with constant velocity. They verify a very specific situation: a wrong way driver threatens two autonomously driving vehicles on a road with three lanes.

Wongpiromsarn et al. [41] verify safety of the planner-controller subsystem of a single autonomous ground vehicle. Their verification techniques restrict acceleration changes to fixed and perfect polling frequency, while our model of an arbitrary number of cars allows changes in acceleration at any point in time, with irregular sensor updates.

Damm et al. [29] give a verification rule that is specialized to collision freedom of traffic agents. To show that two cars do not collide, they need to manually prove eighteen verification conditions. Lygeros and Lynch [68] prove safety only for one deceleration strategy for a string of vehicles: the leading vehicle applies maximum deceleration until it stops, while at the same time, the cars following it in the string decelerate to a stop. The instantaneous, globally synchronized reaction of the cars is an unrealistic assumption that we do not make in our case study. Dolginova and Lynch [69] verify that no collisions with big relative velocity can occur when two adjacent platoons do a merge maneuver. This does not prove the absence of small relative velocity collisions, nor the behavior of 3 platoons or when not merging. In contrast to the manual semantic reasoning of [29, 68, 69], our techniques follow a formal proof calculus [67], which can be mechanized. In the case studies analyzed by [68, 69] safety is proved only for a particular scenario, while our modular formal proofs deal with the general case. In our case study, the cars have more flexibility and an arbitrary number of control choices.

Unlike [29, 39, 41, 44], we prove safety for an arbitrary number of cars. Our techniques and results are more general than the case-specific approaches [29, 39, 41, 44, 68, 69], as we

prove collision-freedom for any number of cars driving on any finite number of lanes. None of the previously cited papers have proved safety for distributed car control in which cars can dynamically enter the highway system, change lanes, and exit.

4.3 Preliminaries: Quantified Differential Dynamic Logic

Distributed car control systems are distributed hybrid systems, with many cars on the road, acting independently. It would be impossible to write down a distinct model for each car explicitly, let alone find the computational power to analyze the exponential interactions between every car to ensure that no two cars collide. Instead, we use *quantified hybrid programs* (QHPs) [67] to model the system, as it adds quantification over assignment and evolution to HPs. This allows us to model an arbitrary number of cars in the system and more readily analyze their interactions using invariants.

QHPs are defined by the grammar (α,β are QHPs, θ a term, i a variable, f a function symbol, and H a formula of first-order logic):

$$\alpha,\beta \quad ::= \quad \forall i:C\ f(i) \ := \ \theta \mid \forall i:C\ f(i)' = \theta \,\&\, H \mid f(i) \ := \ * \mid ?H \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*$$

The effect of *quantified assignment* $\forall i:C\ f(i) := \theta$ is an instantaneous discrete jump assigning θ to $f(i)$ simultaneously for all objects i of type C. Usually i occurs in θ. The effect of *quantified differential equation* $\forall i:C\ f(i)' = \theta \,\&\, H$ is a continuous evolution where, for all objects i of type C, all differential equations $f(i)' = \theta$ hold *and* (written & for clarity) formula H holds throughout the evolution (the state remains in the region described by H). Usually, i occurs in θ. Here $f(i)'$ is intended to denote the derivative of the interpretation of the term $f(i)$ over time during continuous evolution, not the derivative of $f(i)$ by its argument i. For $f(i)'$ to be defined, we assume f is an $\mathbb{R}$-valued function symbol. The effect of the random assignment $f(i) := *$ is to non-deterministically pick an arbitrary number or object (of type the type of $f(i)$) as the value of $f(i)$. The definitions of the remaining hybrid program operators are identical to their definitions in dL and dRL.

For stating and proving properties of QHPs, we use *quantified differential dynamic logic* QdL [67] with the grammar:

$$\phi,\psi \ ::= \ \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall i:C\ \phi \mid \exists i:C\ \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

In addition to all formulas of first-order real arithmetic, QdL allows formulas of the form $[\alpha]\phi$ with a QHP α and a formula ϕ. Formula $[\alpha]\phi$ is true in a state ν iff ϕ is true in all states that are reachable from ν by following the transitions of α; see [67] for details.

From Lemma 1, we know that dRL and dL are equally expressive. Proofs in QdL may use theorems proved in dL as lemmas. As discussed in Section 3.3, any property proved using the sound calculus of dRL is also provable in dL, meaning that proofs in QdL may also use theorems proved in dRL as lemmas.

In Section 4.5, we use dL to prove safety of a two-car system. However, we can not use dL to prove systems-level reasoning about an arbitrary number of cars. We instead extend the two-car proof using QdL to verify safety for the multi-lane system with an arbitrary number of cars. The

proof of the two-car system allows for a non-deterministic discrete controller, which is meant to be an over-approximation of any implementation. Later, in Section 4.9, we use dRℒ to prove that a specific, deterministic controller is a refinement of this controller, thereby further reducing the gap between an implemented system and this proof of safety.

4.4 The Distributed Car Control Problem

Our approach to proving safety of a distributed car control system is to break the verification into modular pieces. In this way, we simplify what would otherwise be a very large and complex proof. The ultimate result of this chapter is a formally verified model of any straight stretch of highway on which each car is following adaptive cruise control. On any highway, there will be an arbitrary number of lanes and an arbitrary number of cars, and the system will change while it runs when cars enter and leave the highway.

This would be an incredibly complex system to verify if we were to tackle it at this level. Each lane has a group of cars driving on it. This group is constantly changing as cars weave in and out of surrounding traffic. Each car has a position, velocity, and acceleration, and must obey the laws of physics. On top of that, in order to ensure complete safety of the system, every car must be certain at all times that its control choices will not cause a collision anywhere else in the system at any time in the future.

These issues are compounded by the limits of the sensory and communications networks. On a highway that stretches hundreds of miles, we could not hope for any car to collect and analyze real-time data from every other car on the interstate. Instead, we must assume each car is making decisions based on its local environment, e.g., within the limitations of sensors, V2V and V2I communication, and real-time computation.

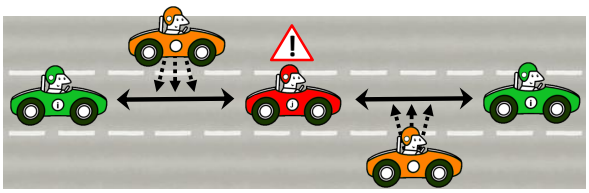

Figure 4.1: Emergent highway collision risk

Additionally, once you split your system into reasonably local models, it is still difficult to reason about how these local groups of cars interact. For example, consider a local group of three cars for a lane change maneuver: the car changing lanes, and the two cars that will be ahead and behind it. It is tempting to signal the car ahead to speed up and the car behind to slow down in order to make space for the car changing lanes. This is perfectly reasonable on the local level; however, Fig. 4.1 demonstrates a problem that appears when we attempt to compose these seemingly safe local cases into a global system. Two cars are attempting safe and legal lane changes simultaneously, but the car which separates the merging cars is at risk. The car in the middle simultaneously receives requests to slow down and speed up. It cannot comply, which could jeopardize the safety of the entire system.

To avoid complex rippling cases that could result in a situation similar to the one in Fig. 4.1, we organize our system model as a collection of hierarchical modular pieces. The smallest piece consists of only two cars on a single lane. We present a verification of this model in Section 4.5 and build more complex proofs upon it throughout the chapter.

In Section 4.6, we prove that a lane with an arbitrary number of cars driven by any distributed homogeneous adaptive cruise control system is safe, assuming the system has been proved safe for two cars. We generate our own verified adaptive cruise control model for this system, but, due to the modular proof structure, it can be substituted with *any* implementation-specific control system which has been proved safe for two cars.

The verification of this one lane system, as well as the verification we present in Section 4.8 for a highway with multiple lanes, will hold independently with respect to the adaptive cruise control specifications. In Section 4.7, we look at the local level of a multi-lane highway system. We verify the adaptive cruise control for a single lane, where cars are allowed to merge in and out of the lane. Finally in Section 4.8, we compose the lane systems verified in Section 4.7 to provide a full verification of the highway system.

4.5 Local Lane Control

The local car dynamics problem that we are solving is: we have two cars on a straight lane that independently control their rate of acceleration or braking and we want to prove that they will not collide. This system contains complex physical controls as well as discrete and continuous dynamics, thus, is a hybrid system. Once the model for the local problem is verified, we will use it in a compositional fashion to prove safety for more complicated scenarios, such as multiple cars driving on a lane or on parallel lanes. We can apply modular composition because we have structured the models in a hierarchical order, we have found the right decomposition of the sub-problems and we have identified the right invariants.

4.5.1 Modeling

We develop a formal model of the local car dynamics as a QHP. Each car has state variables that determine how it operates: position, velocity, and acceleration. For follower car f, x_f represents its position, v_f its velocity, and a_f its acceleration (similarly for leader car ℓ).

The continuous dynamics for f are described by the following differential equation system: $x_f' = v_f$, $v_f' = a_f$. This is the ideal-world dynamics that is adequate for a kinematic model of longitudinal lane maneuvers. The rate with which the position of the car changes is given by x_f', i.e., the velocity. The velocity itself changes continuously according to the current acceleration a_f. We do not assume permanent control over the acceleration, but tolerate delays since sensor readings are not available continuously, control decisions may need time, and actuators may take time to react. For simplicity, though, we still assume that, once set, the acceleration a_f takes instant effect. We assume a global limit for the maximum acceleration and we denote it by $A \geq 0$. We assume that all cars have an emergency brake with a braking power between a maximum value B and a minimum value b, where $B \geq b > 0$. The two values have to be positive, otherwise the cars cannot brake. They may be different, however, because we cannot expect all cars to realize exactly the same emergency braking power and it would be unrealistic to build a system based on the assumption that all reactions are equal.

In Fig. 4.2, we see that leader ℓ brakes unexpectedly at time $t1$ with its maximum braking power, $-B$. Unfortunately, f did not follow ℓ at a safe distance, and so when sensor and network data finally inform f at time $t2$ that ℓ is braking, it is already too late for f to prevent a collision. Although f applies its full braking power, $-b$, at time $t2$, the cars will inevitably crash at time $t3$. The same problem can happen if ℓ brakes with $-b$ and f brakes with $-B$. This example shows that control choices which look good early on can cause problems later. Adding cars to the system amplifies these errors.

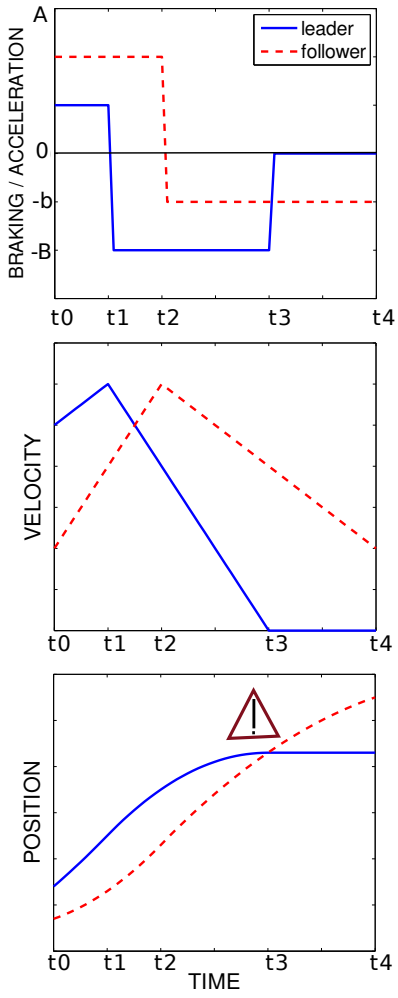

Figure 4.2: Local car crash

We present the entire specification of the local lane control (`llc`), consisting of the discrete control and the continuous dynamics, in Model 1. This system evolves over time, which is measured by a clock, i.e., variable t changing with slope $t' = 1$ as in (4.9). The differential equation system (4.9) formalizes the physical laws for movement, which are restricted to the evolution domain (4.10). Neither human drivers nor driver assistance technology are able to react immediately and each vehicle or driver will have a specific reaction time. Therefore we have a constant parameter, ε, which serves as an upper bound on the reaction time for all vehicles. We verify car control for arbitrary values of ε. Cars can react as quickly as they want, but they can take no longer than ε.

The leading car is not restricted by the car behind, so it may accelerate, coast, or brake at will. In Model 1, a_ℓ is first randomly assigned a real value, non-deterministically through (4.3). The model continues if a_ℓ is within the physical limits of the car's brakes and engine, i.e. between -B and A. On the other hand, f depends on the distance to ℓ and has a more restrictive set of possible moves. Car f can take some choices only if certain safety constraints about the distance and velocities are met.

Braking is allowed at all times, so a human driver may always override the automated control to brake in an emergency. In fact, braking is the only option if there is not enough distance between the cars for f to maintain its speed or accelerate. This is represented in (4.5), where there is no precondition for any force between $-B$ and $-b$.

The second possibility, (4.6), is that there is enough distance between the two cars for f to take any choice. This freedom is only given when (4.8) is satisfied. If (4.8) holds, then ℓ will still be safely in front of f until the controllers can react again (i.e., after they drive for up to ε time units), no matter how ℓ accelerates or brakes. This distance is greater than the minimum distance required for safety if they both brake simultaneously. The ε terms in (4.8) add this extra distance to account for the possibility that f accelerates for time ε even when ℓ decides to brake, which f may not notice until the next sensor update. These terms represent the distance traveled

Model 1 Local lane control (`llc`)

$$\texttt{llc} \equiv (ctrl; dyn)^* \tag{4.1}$$
$$ctrl \equiv \ell_{ctrl} \parallel f_{ctrl} \tag{4.2}$$
$$\ell_{ctrl} \equiv (a_\ell := *;\ ?(-B \leq a_\ell \leq A)) \tag{4.3}$$
$$f_{ctrl} \equiv \texttt{brake} \cup \texttt{safe}_* \cup \texttt{stopped} \tag{4.4}$$
$$\texttt{brake} \equiv (a_f := *;\ ?(-B \leq a_f \leq -b)) \tag{4.5}$$
$$\texttt{safe}_* \equiv (?\mathbf{Safe}_\varepsilon;\ a_f := *;\ ?(-B \leq a_f \leq A)) \tag{4.6}$$
$$\texttt{stopped} \equiv (?(v_f = 0);\ a_f := 0) \tag{4.7}$$
$$\mathbf{Safe}_\varepsilon \equiv x_f + \frac{v_f^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v_f\right) < x_\ell + \frac{v_\ell^2}{2B} \tag{4.8}$$
$$dyn \equiv (t := 0;\ x_f' = v_f,\ v_f' = a_f,\ x_\ell' = v_\ell,\ v_\ell' = a_\ell, t' = 1 \tag{4.9}$$
$$\&\ v_f \geq 0 \ \wedge\ v_\ell \geq 0 \ \wedge\ t \leq \varepsilon) \tag{4.10}$$

during one maximum reaction cycle of ε time units with worst-case acceleration A, including the additional distance needed to reduce the speed down to v_f again after accelerating with A for ε time units.

Now the third possibility. If f had previously chosen to brake by $a_f = -b$ then the continuous evolution *dyn* cannot continue with the current acceleration choices below velocity $v_f = 0$ due to constraint (4.10). Thus, we add the choice (4.7) saying that the car may always choose to stand still at its position if its velocity is 0 already.

The two cars can repeatedly choose from the range of legal accelerations. This nondeterministic repetition is represented by operator * in (4.1). The controllers of the two cars operate in parallel as seen in (4.2). Notice that the controllers are independent with respect to read and write variables (which also makes sense for implementation purposes), so in this case, parallel ($\parallel$) is equivalent to sequential composition (;).

4.5.2 Verification

To verify the local lane control problem modeled in Section 4.5.1, we use a formal proof calculus for Qd$\mathcal{L}$ [67]. In the local lane control problem, we want f to be safely behind ℓ at all times. To verify that a collision is not possible, we show that there is always a reasonable distance between ℓ and f; enough distance that if both cars brake instantly, the cars would not collide. We verify this property for all times and under any condition which the system can run, so if a car can come so close to another car that even instant braking would not prevent a crash, the system is already unsafe.

For two cars f and ℓ, we have identified the following crucial relation ($f \ll \ell$), i.e., follower f is *safely behind* leader ℓ:

$$(f \ll \ell) \equiv (x_f \leq x_\ell) \wedge (f \neq \ell) \rightarrow \left(x_f < x_\ell \wedge x_f + \frac{v_f^2}{2b} < x_\ell + \frac{v_\ell^2}{2B} \wedge v_f \geq 0 \wedge v_\ell \geq 0\right)$$

If ($f \ll \ell$) is satisfied, then f has a safe distance from ℓ. The formula states that, if ℓ is the leading car (i.e., $x_f \leq x_\ell$ for different cars $f \neq \ell$), then the leader must be strictly ahead of the follower, and there must be enough distance between them such that the follower can stop when the leader is braking. Also both cars must be driving forward.

The safe distance formula ($f \ll \ell$) is the most important invariant. The system must satisfy it at all times to be verified. This is not to be confused with the definition of $\mathbf{Safe}_\varepsilon$ in the control, which must foresee the impact of control decisions for the future of ε time. For simplicity, these formulas do not allow cars to have non-zero length; however, adding the car length to x_f would eliminate this requirement.

Proposition 5 (Safety of local lane control `llc`)**.** *If car f is safely behind car ℓ initially, then the cars will never collide while they follow the* `llc` *control model; therefore, safety of* `llc` *is expressed by the provable formula:* $(f \ll \ell) \rightarrow [\texttt{llc}](f \ll \ell)$

We proved Proposition 5 using KeYmaera, a theorem prover for hybrid systems (proof files available online [70]). A proof sketch is presented in Section 4.11.1.

4.6 Global Lane Control

Figure 4.3: Lane risk

In Section 4.5 we show that a system of two cars is safe, which gives a local version of the problem to build upon. However, our goal is to prove safety for a whole highway of high-speed vehicles. The next step toward this goal is to verify safety for a single lane of n cars, where n is arbitrary and finite, and the ordering of the cars is fixed (i.e., no car can pass another). Each car follows the same control we proved safe for two cars in Section 4.5, but adding cars to the system and making it distributed has introduced new risks. It is now necessary to show, for example, if you are driving along and the car in front of you slows while the car behind simultaneously accelerates, you won't be left sandwiched between with no way to avoid a collision (as in Fig. 4.3).

4.6.1 Modeling

Because we are now looking at a lane of cars, our model will require additional features. First, we will need to represent the position, velocity, and acceleration of each car. If these variables were represented as primitives, the number of variables would be large and difficult to handle. Using only primitive variables, we cannot verify a system for any arbitrary number of cars, i.e., we could verify for, say, 5 cars, but not for any n cars. Therefore, we give each car an index, i, and use first-order variables $x(i)$, $v(i)$, and $a(i)$ to refer to the position, velocity and acceleration of car i. With these first-order variables, our verification applies to a lane of any number of cars.

Of course, the cars are all driving along the road at the same time, so we evolve the positions of the cars simultaneously along their differential equations. The acceleration, $a(i)$, of all cars is also set simultaneously in the control. We need notation for this parallel execution, so we use the universal quantifier ($\forall$) in the definition of the overall control and continuous dynamics (see (4.12) and (4.17) in Model 2). The control of all cars in the system is defined by $ctrl^n$ (4.12).

Model 2 Global lane control (`glc`)

$$\texttt{glc} \equiv (ctrl^n; dyn^n)^* \tag{4.11}$$
$$ctrl^n \equiv \forall i : C\ (ctrl(i)) \tag{4.12}$$
$$ctrl(i) \equiv (a(i) := *; ?(-B \leq a(i) \leq -b)) \tag{4.13}$$
$$\cup\ (?\mathbf{Safe}_\varepsilon(i);\ a(i) := *;\ ?(-B \leq a(i) \leq A)) \tag{4.14}$$
$$\cup\ (?(v(i) = 0);\ a(i) := 0) \tag{4.15}$$
$$\mathbf{Safe}_\varepsilon(i) \equiv x(i) + \frac{v(i)^2}{2b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v(i)\right) < x(L(i)) + \frac{v(L(i))^2}{2B} \tag{4.16}$$
$$dyn^n \equiv t := 0;\ \forall i : C\ (dyn(i), t' = 1\ \&\ v(i) \geq 0 \wedge t \leq \varepsilon) \tag{4.17}$$
$$dyn(i) \equiv x(i)' = v(i), v(i)' = a(i) \tag{4.18}$$

This says that for each car i, we execute $ctrl(i)$. This control is exactly the control defined in Section 4.5 - under *any* conditions the car may brake (4.13); if the car is safely following its leader, it may choose any valid acceleration between $-b$ and A (4.14); and if the car is stopped, it may remain stopped (4.15). There are only two distinctions between the control introduced in `glc` and the control used in `llc` described in Section 4.5. First, we change primitive variables to first-order variables. Second, with so many cars in the system, we have to determine which car is our leader.

It is vital that every car be able to identify, through local sensors or V2V/V2I communication networks, which car is directly in front of it. It is already assumed that the sensor and communication network is guaranteed to give accurate updates to every car within time ε. We now also make the reasonable assumption that with each update, every car is able to identify which car is directly ahead of it in its lane. This may be a bit tricky if the car only has sensor readings to guide it, but this assumption is reasonable if all cars are broadcasting their positions (and which lane they occupy in the case of multiple lanes). For some car i, we call the car directly ahead of it $L(i)$, or the *leader of car i*. More formally, we assume the following properties about $L(i)$:

$$L(i) = j \equiv x(i) \leq x(j) \wedge \forall k : C \backslash \{i, j\}\ (x(k) \leq x(i) \vee x(j) \leq x(k))$$
$$(i \ll L(i)) \equiv \forall j : C((L(i) = j) \rightarrow (i \ll j))$$

The equation $L(i) = j$ is expanded to mean that the position of j must be ahead of the position of i, and there can be no cars between. The second formula states that for a car, i, to be safely behind its leader, denoted $(i \ll L(i))$, we require that i should be safely behind any car which fulfills the requirements of the first equation. At the end of the finite length lane, we position a stationary car.

The constraint $\mathbf{Safe}_\varepsilon$ from Section 4.5 has been updated to a first-order variable as well (4.16). It now uses $L(i)$ to identify which car is directly ahead of car i, and then determines if i is following safely enough to accelerate for ε time. This constraint is applied to all cars in the system when the individual controls set acceleration.

The continuous dynamics are the same as those described in Section 4.5, but with the added dynamics of the other cars in the system (4.17). Once $a(i)$ has been set for all cars by $ctrl^n$ (4.12), each car evolves along the dynamics of the system for no more than ε time (maximum reaction

time). The position of each car evolves as the second derivative of the acceleration set by the control (4.18). The model requires that the cars never move backward by adding the constraint $v(i) \geq 0$. We still have a global time variable, t, that is introduced in the definition of dyn^n (4.17). Since $t' = 1$, all cars evolve along their respective differential equations in an absolute timeframe. Note that t is never read by the controller, thus, `glc` has no issues with local clock drift.

We model all cars in the system as repeatedly setting their accelerations as they synchronously receive sensor updates (4.12) and following the continuous dynamics (4.17). When put together and repeated non-deterministically with the * operator, these QHPs form the `glc` model (4.11) for global lane control. The `glc` model is easy to implement since each car relies on local information about the car directly ahead. Our online supplementary material shows a demo of an implementation of this model [70].

4.6.2 Verification

Now that we have a suitable model for a system of n cars in a single lane, we identify a suitable set of requirements and prove that our model never violates them. In Section 4.5, since there were only two cars on the road, it was sufficient to show that the follower car was safely behind its leader at all times. However, in this model it is not enough to only ensure safety for each car and its direct leader. We must also verify that a car is safely following all cars ahead – each car has to be safely behind its leader, and the leader of its leader, and the car in front of that car, and so on.

For example, suppose there were a long line of cars following each other very closely (they could, for instance, be in a platoon). If the first car brakes, then one-by-one the cars behind each react to the car directly in front of them and apply their brakes. In some models, it would be possible for these reaction delays to add up and eventually result in a crash [72]. Our model is not prone to this fatal error, because our controllers are explicitly designed to tolerate reaction delays. Each car is able to come to a full stop no matter what the behavior of the cars in front of it (so long as all cars behave within the physical limits of their engines and brakes). To show this, we must verify that under the system controls every car is always safely behind all cars ahead until the lane ends. We do this by first defining *transitive leaders*, $L^*(i)$ as follows:

$$(i \ll L^*(i)) \;\equiv\; [k := i;\; (k := L(k))^*](i \ll k)$$

The QHP, $k := i;\; (k := L(k))^*$, continually redefines k to be the next car in the lane (until the lane ends). Because this QHP is encapsulated in [ ], all states that are reachable in the program must satisfy the formula $(i \ll k)$. In other words, starting with $(k := i)$, we check that i is safely behind k, or $(i \ll i)$. Next, $k := L(k)$, so $k := L(i)$, and we prove that i is safely behind k: $(i \ll L(i))$. Then we redefine k to be its leader again $(k := L(k))$, and we check that i is safely behind k: $(i \ll L(L(i)))$. This check is continued indefinitely: $(i \ll L(L(...\, L(i))))$. Hence the notation, $(i \ll L^*(i))$.

Proposition 6 (Safety of global lane control `glc`)**.** *For every configuration of cars in which each car is safely following the car directly in front of it, all cars will remain in a safe configuration (i.e., no car will ever collide with another car) while they follow the distributed control. This is expressed by the following provable formula:*

$$\forall i : C(i \ll L(i)) \;\rightarrow\; [\texttt{glc}](\forall i : C(i \ll L^*(i)))$$

This means that as the cars move along the lane, every car in the system is safely following all of its transitive leaders.

Using Gödel's generalization rule (described in Section 4.11.2), our proof for a lane of cars splits immediately into two branches: one which relies on the verification of the control and dynamics in the local, two car case, and one which verifies the rest of the system. These two branches are independent, and furthermore, the control and dynamics of the cars are only expanded in the verification of the local model. This is good news for two reasons. First, it keeps the resulting proof modular, which makes it possible to verify larger and more complex systems. Second, if the control or dynamics of the model are modified, only an updated verification of safety for two cars will be needed to verify the new model for the whole system. Proof details are available in Section 4.11.2.

4.7 Local Highway Control

In Section 4.6, we verified an automated control system for an arbitrary, but constant, number of cars on a lane. Later, we will put lots of these lanes together to model highway traffic. In our full highway model, cars will be able to pass each other, change lanes, and enter or leave the highway. We first study how this full system behaves from the perspective of a single lane. When a car changes into or out of that lane, it will look like a car is appearing or disappearing in the middle of the lane: in front of and behind existing cars. Now it is crucial to show that these appearances and disappearances are safe.

If a new car cuts into the lane without leaving enough space for the car behind it, it could cause an accident. Furthermore, when two cars enter the lane simultaneously, if there are several cars between them, we must prove that there will not be a ripple effect which causes those cars between to crash (also see Fig. 4.1). Faithful verification must apply to all kinds of complex maneuvers and show safety for all cars in the system, not just those involved locally in one maneuver.

Our verification approach proves separate, modular properties. This allows us to compose these modular proofs and verify collision freedom for the entire system for any valid maneuver, no matter how complex, even multiple maneuvers at different places.

4.7.1 Modeling

We have additional challenges in modeling this new system where cars can appear and disappear dynamically. First of all, in previous sections we have used $\forall i : C$ to mean "for all cars in the system." We will now abuse this notation and take it to mean "for all cars which currently exist on this lane." (In our formal proof we use an actualist quantifier to distinguish between these situations. This technique is described in detail in another paper [67].) Secondly, our model must represent what physical conditions in the lane must be met before a car may disappear or appear safely. And finally, the model must be robust enough to allow disappearances and appearances to happen throughout the evolution of the system (i.e., a car may enter or leave the lane at any time).

Model 3 Local highway control (`lhc`)

$$\texttt{lhc} \equiv (\mathit{delete}^*; \mathit{create}^*; \mathit{ctrl}^n; \mathit{dyn}^n)^* \tag{4.19}$$

$$\mathit{create} \equiv n := \mathit{new};\ ?((F(n) \ll n) \wedge (n \ll L(n))) \tag{4.20}$$

$$(n := \mathit{new}) \equiv n := *;\ ?(\mathsf{E}(n) = 0);\ \mathsf{E}(n) := 1 \tag{4.21}$$

$$(F(n) \ll n) \equiv \forall j : C\ (L(j) = n \rightarrow (j \ll n)) \tag{4.22}$$

$$\mathit{delete} \equiv n := *;\ ?(\mathsf{E}(n) = 1);\ \mathsf{E}(n) := 0 \tag{4.23}$$

Recall that a car, n, has three real values: position, velocity and acceleration. Now that cars can appear and disappear, we add a fourth element: existence. The existence field is just a bit that we flip on ($\mathsf{E}(n) := 1$) when the car appears and flip off ($\mathsf{E}(n) := 0$) when the car disappears.

When we create a new car, n, we start by allowing the car to be anything. This can be written in dynamic logic as a random assignment $n := *$. Of course, when we look at the highway system as a whole, we won't allow cars to pop out of thin air onto the lane. This definition can be restricted to cars which already exist on an adjacent lane. However, since the choice of $*$ is non-deterministic, we are verifying our model for all possible values of n. This means that the verification required for an entire highway system will be a subset of the cases covered by this model of a single lane. Because $n := *$ allows n to be any car, one that exists on the lane or one that doesn't, we first must check that this "new" car isn't already on the lane. If it doesn't exist, i.e. $?(\mathsf{E}(n) = 0)$, then we can flip our existence bit to on and it will join the existing cars on this lane (4.21).

Now that we have defined appearance, we can define its dual: disappearance. We delete cars by choosing a car, n, non-deterministically, checking that it exists, and then flipping that bit so that it no longer exists on this lane (4.23). After a delete, notice that while the car ceases to exist physically on our lane, we are still able to refer to it in our model and verification as car n – a car that used to be in the lane.

A car may leave the lane at any time (assuming there is an adjacent lane which it can move into safely), but it should only be allowed to enter the lane if it is safely between the car that will be in front of it and the car that will be behind it. Because of this, when creating a car in the lane, our model will check that the car is safely between the car in front and behind. Note that this covers all behavior choices of the nearby cars as well, so it is possible that the car could have communicated with the other cars and requested that they make space before it switches lanes, but our proof can be oblivious to this extra layer of complexity on the system model. If we have a test which follows a creation of a new car, as in our definition of *create* in (4.20), a new car will only appear if the test succeeds. The formula $(F(i) \ll i)$ evaluates to true if car i is safely ahead of the car behind it. This is the dual of $(i \ll L(i))$. We define this formally in terms of $(i \ll L(i))$ as shown in (4.22).

The `lhc` model is identical to the `glc` model in Section 4.6, but at the beginning of each control cycle it includes zero or more car *delete*s or *create*s as shown by delete^* and create^* in (4.19). It is important to note that the verification will include interleaving and simultaneous *create*s and *delete*s since the continuous dynamics (dyn^n) are allowed to evolve for zero time and start over immediately with another *delete* and *create* cycle.

4.7.2 Verification

Now that we have a model for local highway control, we have to describe a set of requirements that we want the model to satisfy in order to ensure safety. These requirements will be identical to the requirements necessary in the global lane control. We want to show that every car is a safe distance from its transitive leaders: $\forall i : C(i \ll L^*(i))$. Because these requirements are identical to those presented in Proposition 6, the statement of Proposition 7 is identical except for the updated model.

Proposition 7 (Safety of local highway control `lhc`)**.** *Assuming the cars start in a controllable state (i.e. each car is a safe distance from the car in front of it), the cars may move, appear, and disappear as described in the (*`lhc`*) model, then no cars will ever collide. This is expressed by the following provable formula:*

$$\forall i : C(i \ll L(i)) \;\rightarrow\; [\texttt{lhc}]\forall i : C(i \ll L^*(i))$$

We keep the proof of Proposition 7 entirely modular just as we did in the previous section for Proposition 6. The proof is presented in Section 4.11.3.

4.8 Global Highway Control

So far, we have verified an automated car control system for cars driving on one lane. A highway consists of multiple lanes, and cars may change from one lane to the other. Just because a system is safe on one lane does not mean that it would operate safely on multiple lanes. When a car changes lanes, it might change from a position that used to be safe for its previous lane over to another lane where that position becomes unsafe. Lane change needs to be coordinated and not chaotic. We have to ensure that multiple local maneuvers cannot cause global inconsistencies and follow-up crashes; see Fig. 4.1.

4.8.1 Modeling

The first aspect we need to model is which lane is concerned. The quantifier $\forall i : C$, which in Section 4.7 quantified over "all cars which exist on the lane", now needs to be parametrized by the lane that it is referring to. We use the notation $\forall i : C_l$ to quantify over all cars on lane l. Likewise, instead of the existence function $\mathsf{E}(i)$, we now use $\mathsf{E}(i, l)$ to say whether car i exists on lane l. A car could exist on some l but not on others. A car can exist on multiple lanes at once if its wheels are on different lanes (e.g., when crossing dashed lines). We use subscripted $ctrl^n_l, dyn^n_l, L_l(i), L^*_l(i)$ etc. to denote variants of $ctrl^n, dyn^n, L(i), L^*(i)$ in which all quantifiers refer to lane l. Similarly, we write $\forall l : L\; ctrl^m_l$ for the QHP running the controllers of all cars on all lanes at once.

In addition to whatever a car may do in terms of speeding up or slowing down, lane change corresponds to a sequence of changes in existence function $\mathsf{E}(i, l)$. A model for an instant switch of car i from lane l to lane l' would correspond to $\mathsf{E}(i, l) := 0$; $\mathsf{E}(i, l') := 1$, i.e., disappearance from l and subsequent appearance on l'. This is mostly for adjacent lanes $l' = l \pm 1$, but we allow arbitrary lanes l, l' to capture highways with complex topology. Real cars do not change lanes instantly, of course. They gradually move from one lane over to the other while (partially)

occupying both lanes simultaneously for some period of time. This corresponds to the same car existing on multiple lanes for some time.

Gradual lane change is modeled by an appearance of i on the new lane ($\mathsf{E}(i, l') := 1$) when the lane change starts, then a period of simultaneous existence on both lanes while the car is in the process of moving over, and then, eventually, disappearance from the old lane ($\mathsf{E}(i, l) := 0$) when the lane change has been completed and the car occupies no part of the old lane anymore. Consequently, gradual lane change is over-approximated by a series of deletes from all lanes ($\forall l : L\ delete_l^*$) together with a series of appearances on all lanes ($\forall l : L\ new_l^*$). In other words, a single car can appear in any new lane with the hybrid program ($\forall l : L\ new_l^*$), and, after the system has evolved for zero or more time, that same car could then be deleted from any lane with the hybrid program ($\forall l : L\ delete_l^*$). Any interleaving of *new* and *delete* combinations is possible, thus ghcis an over-approximation of the behavior where those lanes and positions are necessarily adjacent. Global highway control with multiple cars moving on multiple lanes and non-deterministic gradual lane changing can be modeled by QHP:

$$\texttt{ghc} \equiv (\forall l : L\ delete_l^*;\ \forall l : L\ new_l^*;\ \forall l : L\ ctrl_l^n;\ \forall l : L\ dyn_l^n)^*$$

4.8.2 Verification

Global highway control ghc is safe, i.e., guarantees collision freedom for multi-lane car control with arbitrarily many lanes, cars, and gradual lane changing.

Theorem 1 (Safety of global highway control ghc)**.** *The global highway control system (*ghc*) for multi-lane distributed car control is collision-free. This is expressed by the provable formula:*

$$\forall l : L \forall i : C_l(i \ll L_l(i)) \rightarrow [(\forall l : L\ delete_l^*; \forall l : L\ new_l^*; \forall l : L\ ctrl_l^n; \forall l : L\ dyn_l^n)^*]\ \forall l : L \forall i : C_l(i \ll L_l^*(i))$$

For the proof see Section 4.11.4. Note that the constraints on safe lane changing coincide with those identified in Section 4.7 for safe appearance on a lane.

4.9 Using dR$\mathcal{L}$ to Verify a Specific Controller

Now let's make a minor change to the local lane control (llc) from Model 1. In (4.28), where it is safe for the follower to accelerate, we now choose a specific value, $\theta(x_f, x_\ell, v_f, v_\ell)$, for that acceleration, rather than setting it nondeterministically as before. This function θ could be very complicated. For example, it could optimize for passenger comfort or fuel efficiency. But whatever it does, we still want to verify that this choice of acceleration does not violate our safety property.

Intuitively, if θ only accelerates when $\mathbf{Safe}_\varepsilon$ holds and brakes otherwise, then this deterministic controller should inherit the proof of safety from Section 4.5. However, without dR$\mathcal{L}$, we would have to completely reprove this property in order to formally verify this closely related property. In this section, we show that $\texttt{llc}_\theta$ refines llc, and therefore inherits the proof of safety.

Notice that two open goals remain in Fig. 4.5 (indicated in red). These two properties relate directly to the choice of $\theta(x_f, x_\ell, v_f, v_\ell)$. First, in the left branch, $\theta(x_f, x_\ell, v_f, v_\ell)$ should never ask

Model 4 Deterministic local lane control ($\texttt{llc}_\theta$)

$$\texttt{llc}_\theta \equiv (ctrl_\theta; dyn)^* \tag{4.24}$$
$$ctrl_\theta \equiv \ell_{ctrl} \parallel f_{ctrl_\theta}; \tag{4.25}$$
$$\ell_{ctrl} \equiv (a_\ell := *;\ ?(-B \leq a_\ell \leq A)) \tag{4.26}$$
$$f_{ctrl_\theta} \equiv \texttt{brake} \cup \texttt{safe}_\theta \cup \texttt{stopped} \tag{4.27}$$
$$\texttt{safe}_\theta \equiv a_f := \theta(x_f, x_\ell, v_f, v_\ell) \tag{4.28}$$
$$\texttt{brake} \equiv (a_f := *;\ ?(-B \leq a_f \leq -b)) \tag{4.29}$$
$$\texttt{stopped} \equiv (?(v_f = 0);\ a_f := 0) \tag{4.30}$$
$$dyn \equiv (t := 0;\ x_f' = v_f,\ v_f' = a_f,\ x_\ell' = v_\ell,\ v_\ell' = a_\ell, t' = 1 \tag{4.31}$$
$$\&\ v_f \geq 0 \ \wedge\ v_\ell \geq 0 \ \wedge\ t \leq \varepsilon) \tag{4.32}$$

the car to brake at a rate stronger than B, which is the physical limit of the car. Second, in the right branch, $\theta(x_f, x_\ell, v_f, v_\ell)$ should not ask the car to accelerate at a rate stronger than A, again the physical limit of the car. We also find our most important condition in the open goal on the right branch, if $\theta(x_f, x_\ell, v_f, v_\ell)$ is not braking harder than rate b, the property **Safe**$_\varepsilon$ must hold. The proof of refinement will only hold if these two goals can be closed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{Fig. 4.5}}{\vdash f_{ctrl_\theta} \leq f_{ctrl}}\,\textit{rule} \qquad \dfrac{*}{\vdash [f_{ctrl_\theta}]\ell_{ctrl} \leq \ell_{ctrl}}\,\leq_{\textit{refl}}}{f_{ctrl_\theta}; \ell_{ctrl} \leq f_{ctrl}; \ell_{ctrl}}\,;}{\vdash ctrl_\theta \leq ctrl}\,\textit{subst} \qquad \dfrac{*}{\vdash [ctrl_\theta]dyn \leq dyn}\,\leq_{\textit{refl}}}{\vdash (ctrl_\theta; dyn) \leq (ctrl; dyn)}\,;}{\vdash \forall^\alpha((ctrl_\theta; dyn) \leq (ctrl; dyn))}\,\forall\text{r}}{\vdash [(ctrl_\theta; dyn)^*]((ctrl_\theta; dyn) \leq (ctrl; dyn))}\,[]\,\textit{gen}}{\vdash (ctrl_\theta; dyn)^* \leq (ctrl; dyn)^*}\,\textit{unloop}}{\vdash \texttt{llc}_\theta \leq \texttt{llc}}\,\textit{subst}$$

Figure 4.4: Proof in dRL of refinement for deterministic local lane control

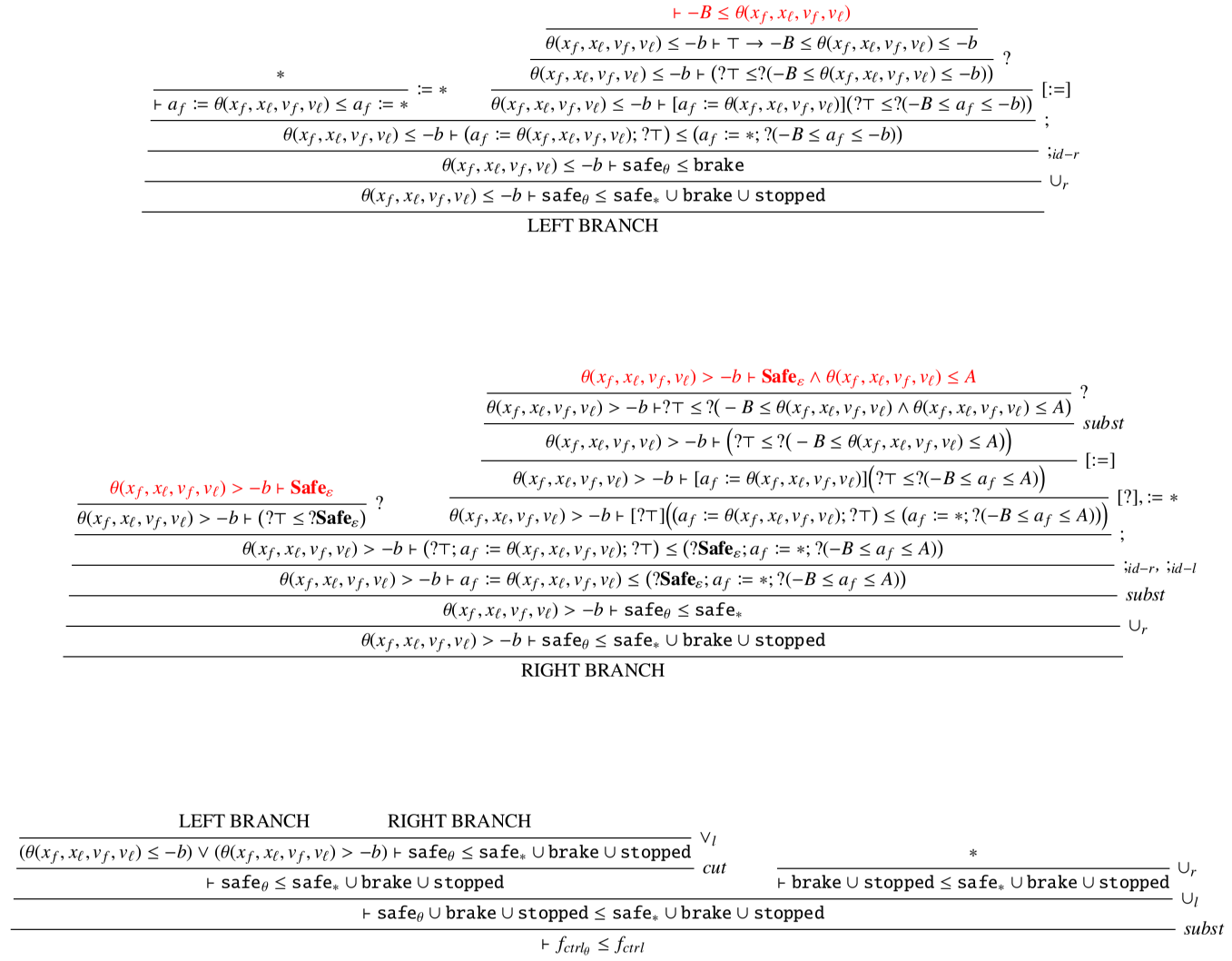

Figure 4.5: Proof in dRL of refinement for deterministic follower discrete controller.

We've shown in this section that we can give a proof of refinement that allows a deterministic controller, $\texttt{llc}_\theta$, to inherit the safety proof for the nondeterministic controller, `llc`. It is not hard to imagine a similar proof of refinement for a set of deterministic controllers, each of which refining `llc` independently. By allowing each car to choose from that set of deterministic controllers, we can model a system where there are an arbitrary number of cars on the road, each being operated by a heterogeneous controller. Using dR$\mathcal{L}$ to show refinement is the obvious choice for making a proof of such a system feasible. We consider this a very promising future application for dR$\mathcal{L}$.

4.10 Conclusion

Distributed car control has been proposed repeatedly as a solution to safety and efficiency problems in ground transportation. Yet, a move to this next generation technology, however promising it may be, is only wise when its reliability has been ensured. Distributed car control dynamics has been out of scope for previous formal verification techniques. We have presented formal verification results guaranteeing collision freedom in a series of increasingly complex settings, culminating in a safety proof for a class of distributed car controllers despite an arbitrary and evolving number of cars moving between an arbitrary number of lanes.

Additionally, we have shown how to use dR$\mathcal{L}$ to formally connect the proof for a class of controllers to a proof for specific controllers, even when different cars are using different controllers. This allows us to easily reuse a single, challenging proof, even when new controllers are introduced or modified. Our research is an important basis for formally assured car control. The modular proof structure we identify in this chapter generalizes to other scenarios, e.g., variations in the local car dynamics or changes in the system design.

4.11 Proofs

In this section we present and explain the proofs for the results presented in the main body of this chapter.

4.11.1 Proofs for Local Lane Control

The proof of local lane control was completed in KeYmaera. To see the full proof, the file can be downloaded from `http://www.ls.cs.cmu.edu/dccs/llc.key.proof` and opened after launching KeYmaera from `http://symbolaris.com/info/KeYmaera.jnlp`. (Mathematica 7 is required, Linux is recommended.)

Safety of Local Lane Control The system in Model 1 consists of a global loop and we use $(f \ll \ell)$ as an invariant of this loop. It can be shown easily that the invariant is initially valid and implies that $(f \ll \ell)$. Proving that the invariant is preserved by the loop body $\mathit{ctrl}; \mathit{dyn}$ is the most difficult part of the proof in KeYmaera.

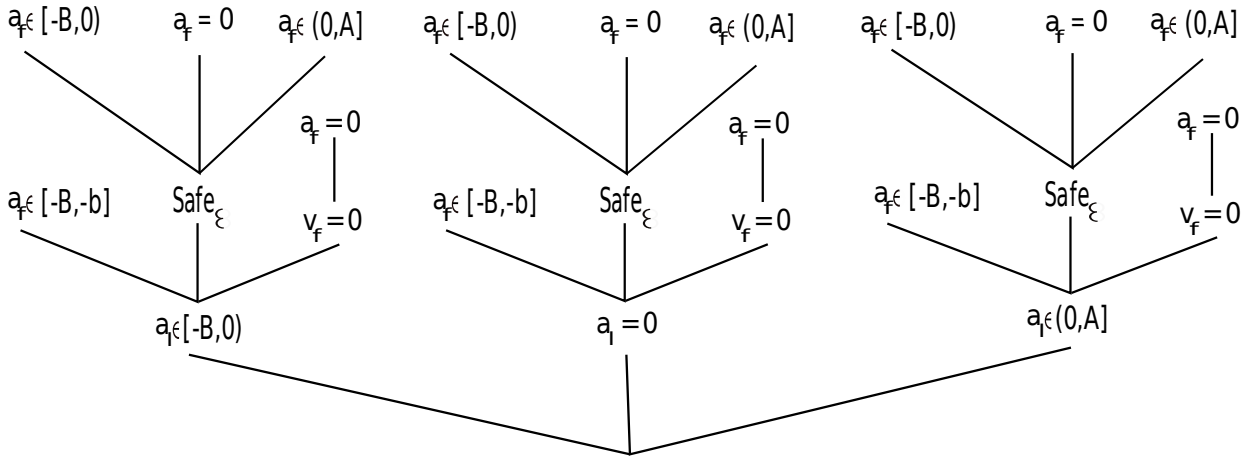

Figure 4.6: Proof Structure

We split the proof into multiple cases, depending on the value of a_ℓ and a_f. All cases are presented in Fig. 4.6. In (4.3) of Model 1, a_ℓ is assigned a value between $-B$ and A. In our proof, we break this assignment up into three cases: $-B \leq a_\ell < 0$, $a_\ell = 0$ and $0 < a_\ell \leq A$. For each of these three cases, there are three possibilities: it can happen that $a_f \in [-B, -b]$, that $\mathbf{Safe}_\varepsilon$ holds or that $v_f = 0$. Each possibility is represented by another subcase in the proof. If $\mathbf{Safe}_\varepsilon$ holds, the proof is further broken up into three subcases: $-B \leq a_f < 0$, $a_f = 0$ and $0 < a_f \leq A$.

There are many branches that are similar in our proof, as shown in Fig. 4.6. We will discuss only the left branch: when $-B \leq a_\ell < 0$, $\mathbf{Safe}_\varepsilon$ holds and $0 < a_f \leq A$. Now, the situation most susceptible to a collision is when the leader ℓ brakes with maximum braking power $-B$ and the follower f accelerates with maximum acceleration A. We first proved that this dangerous situation is collision-free using the following insights. We identified the following useful formula that we could conclude from the assumptions in the antecedent (left of $\rightarrow$):

$$x_\ell > x_f + \frac{v_f^2}{2b} - \frac{v_\ell^2}{2B} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}t^2 + tv_f\right) \tag{4.33}$$

Using a lemma (which formally corresponds to a cut), we proved that this formula follows from the assumptions and then used it to prove the invariant in the remainder of the branch. The formula (4.33) was obtained by combining $\varepsilon \geq t$ and $\mathbf{Safe}_\varepsilon$: we applied transitivity (in the variables $\varepsilon > 0$ and $t > 0$) to the right hand side of the inequality $\mathbf{Safe}_\varepsilon$. The manual introduction of this formula was enough for KeYmaera to prove safety automatically from then on (with a small number of user interactions to simplify arithmetic reasoning and hide extra formulas). After proving that the most dangerous situation, when the leader ℓ brakes with maximum braking power $-B$ and the follower f accelerates with maximum acceleration A, is collision-free, all other situations in this subcase (left branch) can be proved collision-free. All other situations in this subcase turn out to be less dangerous, since the leader ℓ could brake with a braking power strictly bigger than $-B$, or the follower f could accelerate with an acceleration strictly smaller than A. Thus, it was possible to use a formal version of the following monotonicity argument for proving safety: if ($f \ll \ell$) holds when the leader applies braking power $-B$, we prove that it also holds

when he applies not so powerful a braking power. Similarly, if ($f \ll \ell$) holds when the follower accelerates with acceleration A, we prove that it will hold when he applies an acceleration strictly smaller than A.

4.11.2 Proofs for Global Lane Control

In this section, we present a proof of Proposition 6, which was originally introduced in Section 4.6. It is restated here for convenience:

Proposition 6 (Safety of global lane control `glc`**).** *For every configuration of cars in which each car is safely following the car directly in front of it, all cars will remain in a safe configuration (i.e., no car will ever collide with another car) while they follow the distributed control, $ctrl^m$. This is expressed by the following provable formula:*

$$\forall i : C(i \ll L(i)) \rightarrow [\texttt{glc}](\forall i : C(i \ll L^*(i)))$$

This means that as the cars move along the lane, every car in the system is safely following all of its transitive leaders.

In proving Proposition 6, our primary objective is to keep the proof modular. In this way the control, dynamics, and verification can all be changed at the local level without affecting the global level verification. The left branch of the proof in Fig. 4.8 shows the early introduction of the following lemma (Lemma 5), which serves to separate the local and global proofs.

Lemma 5 (Safety of leader)**.** *For any car, i, which is initially following the car in front of it at a safe distance, ($i \ll L(i)$), car i will remain at a safe following distance while it follows the distributed control, $ctrl^n$. That is, the following formula is provable.*

$$\forall i : C(i \ll L(i)) \rightarrow [\texttt{glc}](\forall i : C(i \ll L(i)))$$

In other words, as the cars move along the lane, every car will remain safely behind the car directly in front of it.

Lemma 5 follows as a corollary to Proposition 5 with two modifications: we now have an arbitrary car, i, and its respective leader, $L(i)$, instead of specific cars f and ℓ and we have control and dynamics for n cars instead of 2 cars.

In order to replace f with i and ℓ with $L(i)$ in Proposition 5, we need to guarantee that even if the leader car changes, the proof is not affected. This is very important because when we build on this to prove the safety of lane changes, the order of the cars will change frequently. By defining the lead car, $L(i)$, to be identified by a logical formula, we assure that it has all the required properties for verification, independent of a change in the cars ahead. That is to say, we don't assume that the leader is always the same car, just that it is any car which satisfies the properties of a leader. One plausible alternative would be to consider $L(i)$ to be a data field which keeps track of the leading car. However, if we were to use this approach, we would also have to go through the trouble of checking that the data field updates are always correct.

Our definitions of $ctrl^n$ and dyn^n require the control and dynamics of all cars to be executed in parallel. In our system, the control for any car, i, will only read the position and velocity fields, $x(L(i))$ and $v(L(i))$, of the car ahead and will only write its own acceleration field, $a(i)$. Because all reads and writes are disjoint, the control of one car is independent from the control of all

other cars. This means that executing the car controls sequentially in any order is equivalent to executing the controls in parallel. So, without loss of generality, we may replace the universal car i in Lemma 5 with an arbitrary car, call it I (this is the logical technique of skolemization). Next we apply the hybrid control programs for all cars except I and $L(I)$. Since cars I and $L(I)$ are the only remaining cars in our formula, applying the control for the other cars has no effect and we are left with:

$$(I \ll L(I)) \rightarrow [(ctrl(I); ctrl(L(I)); dyn^n)^*]((I \ll L(I)))$$

Now the safety of an arbitrary car and its leader (Lemma 5) has been reduced to a form where Proposition 5 can be applied directly to prove it.

We will use Lemma 5 along with the following lemma to prove the safety of global lane distributed car control (Proposition 6).

Lemma 6 (Safety of transitive leader)**.** *If all cars are safely following their immediate leaders, then any car, i, is also following all of its transitive leaders, $L^*(i)$:*

$$\forall i : C(i \ll L(i)) \rightarrow \forall i : C(i \ll L^*(i)))$$

Lemma 6 tells us that if every car is safely behind its leader, then every car is also safely behind the leader of its leader, and the car in front of that, and so on down the lane. The proof of Lemma 6 is done by induction and follows from the algebraic property that safety is transitive. A formal proof is presented in Fig. 4.7.

Returning to the proof of Proposition 6, we see in Fig. 4.8 that the property we actually need to complete the proof is

$$[\texttt{glc}]\forall i : C(i \ll L(i)) \rightarrow [\texttt{glc}]\forall i : C(i \ll L^*(i))).$$

However, Lemma 6 is just a more general statement. If $(\phi \rightarrow \psi)$ and $[\alpha]\phi$ are valid (i.e., ϕ always holds while some QHP α is executed), then $[\alpha]\psi$ will also be valid. This is known as Gödel's generalization rule and is more formally stated as:

$$\frac{\phi \rightarrow \psi}{[\alpha]\phi \rightarrow [\alpha]\psi} \text{ ([] GEN)}$$

When looking at the complete proof structure in Fig. 4.8, it is important to notice that the QHP which contains the distributed control and physical dynamics of the cars is only needed in Lemma 5. Because of Gödel's generalization rule, the proof only relies on the verification of the control and dynamics in the local, two car case. It is independent of everything else. This is good news for two reasons. First, it keeps the resulting proof modular, which makes it possible to verify larger and more complex systems. Second, if the engineer who designs the system makes a change in the control or dynamics of the model later in development, under normal circumstances a new proof of safety would have to be created from scratch. However, with our modular proof structure, a new verification of safety for two cars, along with the original verification for the entire system, will be sufficient to ensure safety. The formal proof of Proposition 6 is presented in Fig. 4.8. (Note that we also commute $\wedge$ when we apply the ($\wedge$-R) rule.)

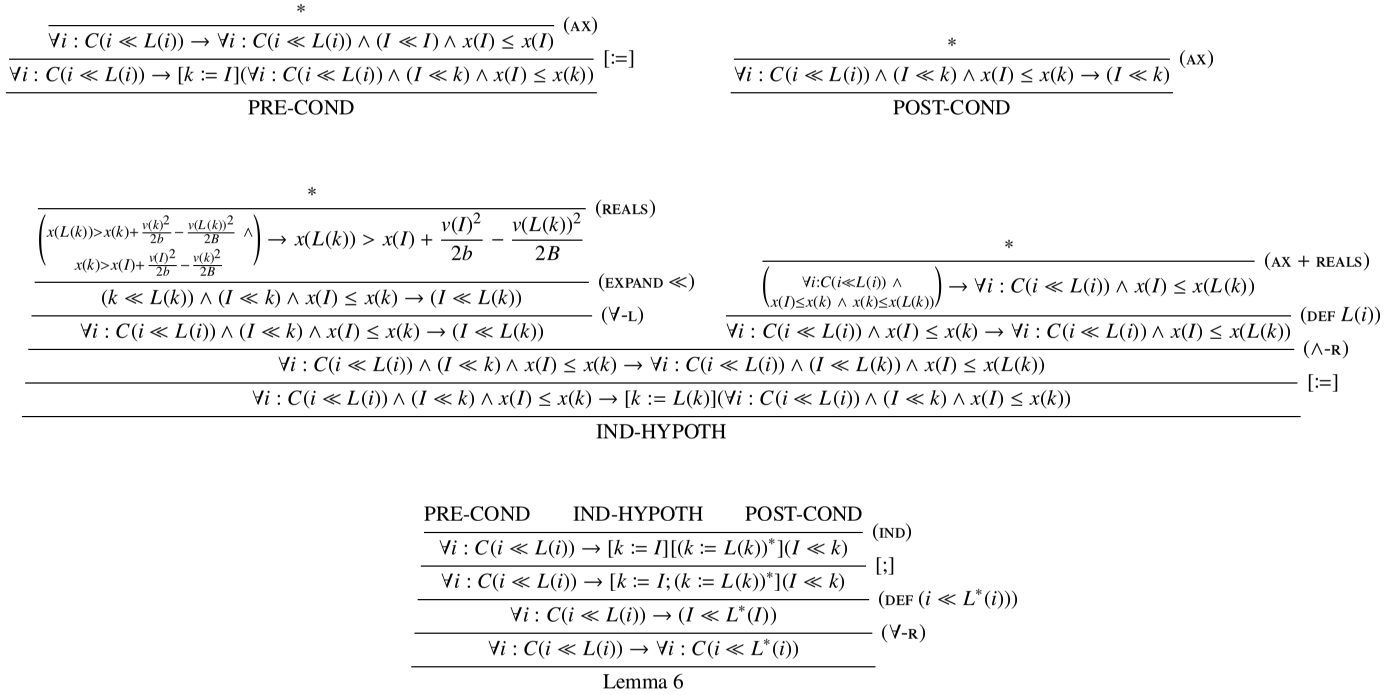

Figure 4.7: Proof of safety for transitive leader (Lemma 6)

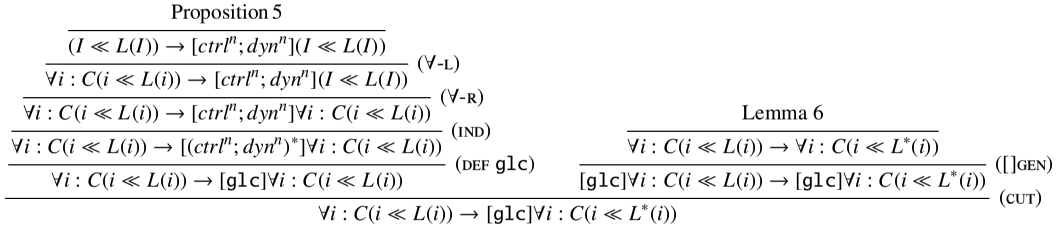

Figure 4.8: Proof of safety for global lane control

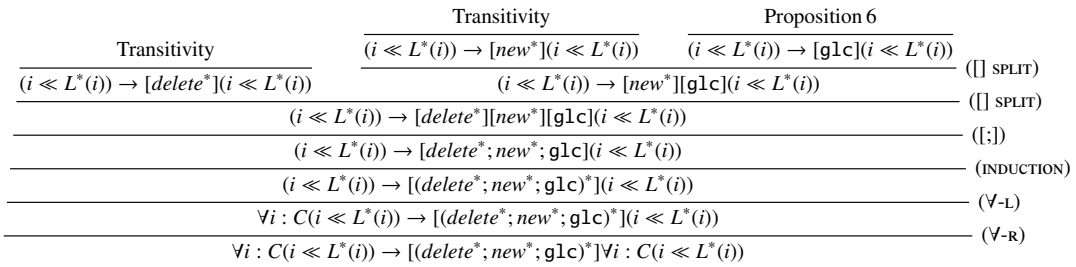

Figure 4.9: Proof of safety for local highway control

4.11.3 Proofs for Local Highway Control

To keep this proof modular, we need one crucial proof rule, ([] SPLIT):

$$\frac{\phi \to [\alpha]\phi \qquad \phi \to [\beta]\phi}{\phi \to [\alpha][\beta]\phi}\ ([]\ \text{SPLIT})$$

Intuitively, ([] SPLIT) makes sense as a proof rule. In the context of our distributed car control system, ϕ is the property that all cars are safely behind their transitive leaders. The QHPs, α and β, could be *delete* and *create* respectively. This rule says that as long as all the cars are safe before, during and after deleting some existing car, and all the cars are safe before, during and after creating a new car, then all cars are safe through the QHP which first deletes and then creates cars. Thus $[\alpha][\beta]\phi$ is valid. More formally, ([] SPLIT) is the combination of two rules: CUT and ([] GEN):

$$\frac{\phi \to [\alpha]\phi \qquad \dfrac{\phi \to [\beta]\phi}{[\alpha]\phi \to [\alpha][\beta]\phi}\ ([]\ \text{GEN})}{\phi \to [\alpha][\beta]\phi}\ (\text{CUT})$$

The proof of Proposition 7, presented in Fig. 4.9, applies ([] SPLIT) twice to split up the `lhc` model into three natural pieces: $delete^*$, new^*, and `glc`. This allows us to use the proof of Proposition 6. All that is left to prove are these two simplified statements about *delete* and *new*.

$(i \ll L^*(i)) \ \to\ [delete^*](i \ll L^*(i))$

$(i \ll L^*(i)) \ \to\ [new^*](i \ll L^*(i))$

The first formula says that if all the cars are safely following their leaders before the $delete^*$, then all the cars will be safely following their leaders after the $delete^*$. We prove this with induction, so we must show that $(i \ll L^*(i))$ holds true after exactly one *delete*. Our definition of safety, $(i \ll j)$, is transitive. This means that when any car, n, is removed from the system, the car previously behind n (i.e., previous $F(n)$) is now safely following the car previously in front of n (i.e., previous $L(n)$).

The argument for the safety of creating a new car is equally straight forward. When a new car is allowed on the lane, it must meet certain conditions, mainly, that it is safely ahead of the car behind it and safely behind the car in front of it. Since our new car n is safely behind the car in front of it ($L(n)$) and we know that the car in front of it is safely behind all of its transitive leaders ($L^*(L(n))$), we also know that our new car is safely behind all of its own transitive leaders ($L^*(n)$). The rest of the argument follows similarly. Note that the top left branches are using the transitivity reasoning in Fig. 4.9. The actual proof uses lots of real arithmetic for this purpose.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Proposition 7}}{\forall i : C_l(i \ll L_l(i)) \to [(delete_l^*; new_l^*; ctrl_l^n; dyn_l^n)^*]\ \forall i : C_l(i \ll L_l^*(i))}\ (\text{RENAME})}{\forall l : L(\forall i : C_l(i \ll L_l(i)) \to [(delete_l^*; new_l^*; ctrl_l^n; dyn_l^n)^*]\ \forall i : C_l(i \ll L_l^*(i)))}\ (\forall\text{GEN})}{\forall l : L \forall i : C_l(i \ll L_l(i)) \to \forall l : L[(delete_l^*; new_l^*; ctrl_l^n; dyn_l^n)^*]\ \forall i : C_l(i \ll L_l^*(i))}\ (\forall\text{DIST})}{\forall l : L \forall i : C_l(i \ll L_l(i)) \to [(\forall l : L\, delete_l^*; \forall l : L\, new_l^*; \forall l : L\, ctrl_l^n; \forall l : L\, dyn_l^n)^*]\ \forall l : L \forall i : C_l(i \ll L_l^*(i))}\ (\text{INDEP})$$

Figure 4.10: Proof of safety for global highway control

4.11.4 Proofs for Global Highway Control

In global highway control verification, we show that the ghc system is collision-free. The primary extra challenge compared to the previous proofs is that we need to consider multiple lanes and prove safe switching between the lanes. What we can work with in this proof is that we have already shown in Proposition 7 that an arbitrary number of cars on one lane with arbitrarily many cars appearing and disappearing is still safe. We need to show that the cars with lane interactions work out correctly.

The proof of the global highway safety Theorem 1 is shown in Fig. 4.10. Theorem 1 follows from Proposition 7, which shows validity of the safety property for an arbitrary lane l. Here we make the lane l explicit in the notation of the following validity:

$$\forall i : C_l(i \ll L_l(i)) \to [(delete_l^*; new_l^*; ctrl_l^n; dyn_l^n)^*]\ \forall i : C_l(i \ll L_l^*(i))$$

This formula is an immediate corollary to Proposition 7, just by a notational change in the proof step marked by (RENAME).

In particular, the universal closure by $\forall l : L$ is still valid by $\forall$-generalization:

$$\forall l : L(\forall i : C_l(i \ll L_l(i)) \to [(delete_l^*; new_l^*; ctrl_l^n; dyn_l^n)^*]\ \forall i : C_l(i \ll L_l^*(i)))$$

This entails the formula in Theorem 1 using the fact that

$$\forall l : L\ (\phi(l) \to \psi(l))\ \text{ implies }\ (\forall l : L\ \phi(l)) \to (\forall l : L\ \psi(l))$$

by $\forall$-distribution and the fact that the formula

$$\forall l : L\ [\alpha]\phi(l)\ \text{ implies }\ [\forall l : L\ \alpha]\forall l : L\ \phi(l),$$

which we mark by (RENAME) in Fig. 4.10. The latter implication does not hold in general. But it does hold for the car control system, because the lane controllers satisfy the read/write independence property discussed in Section 4.6. The control of one lane is independent of the control of another lane, because we have isolated lane interaction into successive local appearance and disappearance steps. The only constraints are the appearance constraints, which are local per lane. Finally note that safety of car appearance and disappearance on the various lanes during ghc follows from the safety of appearance and disappearance that has been proven safe in Proposition 7.

Chapter 5

Efficiency Analysis of Adaptive Cruise Control

We consider an adaptive cruise control system in which control decisions are made based on position and velocity information received from other vehicles via V2V wireless communication. If the vehicles follow each other at a close distance, they have better wireless reception but collisions may occur when a follower car does not receive notice about the decelerations of the leader car fast enough to react before it is too late. If the vehicles are farther apart, they would have a bigger safety margin, but the wireless communication drops out more often, so that the follower car no longer receives what the leader car is doing. In order to guarantee safety, such a system must return control to the driver if it does not receive an update from a nearby vehicle within some timeout period. The value of this timeout parameter encodes a tradeoff between the likelihood that an update is received and the maximum safe acceleration. Combining formal verification techniques for hybrid systems with a wireless communication model, we analyze how the expected efficiency of a provably-safe adaptive cruise control system is affected by the value of this timeout.

5.1 Introduction

We present a class of adaptive cruise controllers that safely set the acceleration of the controlled car based on vehicle to vehicle (V2V) communication data from the car ahead. In the design of such a controller, we can take advantage of fast reaction times resulting from V2V communication, allowing cars to drive close together and increasing highway throughputs. However, a wireless transmission may not be received because of interference, physical obstructions, or the distance between the cars being too large. As a result, any controller that depends on V2V communication must also be able to request help from the driver when that communication fails. If the distance between two cars is kept large, then the car controller is more robust to such communication errors because it has more space to maneuver safely, but the probability that a transmission fails increases at larger distances. Furthermore, the throughput of the highway is reduced. For small distances between two cars, communication works more reliably, but there is less room for errors. At close range, as soon as a single wireless message fails to be delivered,

the follower car would already have to brake, because slowing down is the only guaranteed safe action at close distance when the car no longer has reliable position and velocity data on the leader car. The same issues arise for systems that ask for driver assistance instead of decelerating automatically, as minimizing driver intervention is one goal of such systems.

We use a symbolic class of adaptive cruise controllers to investigate this tradeoff between efficiency and robustness to communication failures quantitatively. This analysis requires both a safety argument for the hybrid system cruise control and an efficiency argument for a probabilistic communication model. The control model for the physical dynamics of the car is a hybrid system with discrete control decisions and an analysis of their impact on the car's continuous motion. The communication model, instead, depends on the physical relationships like distances, but has a probabilistic nature, because communication attempts may succeed or fail at random according to a corresponding distribution.

In Section 5.3, we introduce a control function which, given the current position and velocity of the two cars, chooses a new acceleration for the following car. We provide a formal verification proof that our proposed control function does not allow the following vehicle to collide with the leader, under the assumption that if no message update is received within a bounded limit, the driver assumes control of the vehicle. In Section 5.4, we prove that this control function is optimal by showing that any larger choice of acceleration may admit a collision, and would therefore be unsafe. We then use this safe and optimal control function to analyze how changing the time the controller waits before requesting human assistance affects the range of safe acceleration choices. The methodology and results of this analysis are presented in Section 5.5.

While we are not using dR$\mathcal{L}$ for the proof presented in this chapter, we do use the concept of refinement in the performance analysis of the controller in Section 5.5. In that section, we refine a wide range of safe acceleration choices down to specific values, and show which implementation is likely to have the best results in a tradeoff between requiring frequent oversight from the driver, and having to follow at too great a distance.

5.2 Related Work

We would like our adaptive cruise control system to be as efficient as possible, but it must also behave safely, even in the unlikely event that a communication update is not received for a prolonged period. In this chapter we work within the context of a controller for which safety has been verified over the full continuous state space which results from all possible discrete actions taken by the controller.

This is a stronger guarantee than what can be given by verification methods that discretize the continuous state space, such as the probabilistic model checker Prism [73]. Other methods for verifying hybrid systems, such as SpaceEx [74], only verify linear hybrid systems, and therefore can not handle the nonlinearities required for setting the maximal choice of acceleration as we do in this chapter, and which is necessary for analyzing efficiency of the timeout choice.

While the formal verification of adaptive cruise control presented in [5] can and does handle nonlinear hybrid systems, it uses an implicit choice of acceleration. This means that the adaptive cruise control model presented for two cars in [5] is less challenging to prove safe, but its nondeterminisms make it difficult to implement and not well suited for arguing about the efficiency

of acceleration choices. The assumptions in [5] also require that the maximum time elapsed between transmission broadcasts be bounded by a known constant. For wireless communication, it is not possible to guarantee that any communication is ever successful, so this assumption is infeasible. Instead, we require the driver to take control of the vehicle anytime the communication delay exceeds a given timeout ε. We build on the results presented in [5] to prove safety for a class of controllers that use explicit assignments. We then analyze this class of controllers to discover the optimal timeout ε for passing control of the vehicle back to the driver in case of network failure. This analysis incorporates the probability of successful reception associated with wireless packets sent at varying distances.

Wireless communication between vehicles is a promising tool for improving highway safety. Hartenstein and Laberteaux [75] give an overview of vehicular ad-hoc networks. Jiang et al. [76] propose a specific protocol for safety-related wireless communications between vehicles. The VSC-A report [77] describes an extensive series of experiments testing the performance of wireless V2V communication in a variety of situations that are typically problematic for autonomous vehicle safety systems. Sepulcre and Gozalvez [78] observe that achieving safety in different roadway scenarios could impose very different specifications on the underlying wireless communications system. Meireles et al. [79] experimentally study the effects of physical obstructions, including other vehicles, on packet delivery ratio and signal strength in wireless V2V communication.

Much work has been done on modeling V2V networks. The survey by Stanica, Chaput, and Beylot [80] provides a good overview of current techniques. While modeling wireless networks is relatively well understood, V2V networks pose a unique challenge because vehicles move quickly relative to each other and their environment. The Doppler effect and effects due to the presence of cars, buildings, and other structures could be significant. Dhoutaut et al. [81] propose a simple model that switches between a small number of predefined interference states. Moser et al. [82], on the other hand, give a much more accurate model using raytracing that relies on having a detailed model of the environment and requires much greater computational resources to simulate.

In this chapter, we will use the Nakagami fading model [83] to compute the probability that an individual transmission is passed successfully from one car to another as a function of the distance between the two vehicles. We use this model for simplicity. Another more complicated model could be more accurate and our techniques would allow the use of such a model in our efficiency analysis.

5.3 Verified Adaptive Cruise Control

Hybrid systems have tightly coupled discrete and continuous dynamics. As a result, the consequences of discrete control choices on continuous system dynamics are usually complex and difficult to analyze.

In this chapter, we take as a canonical hybrid system an adaptive cruise controller (ACC). The controller uses discrete message updates via V2V about the car ahead to inform discrete acceleration control decisions, which result in continuous changes in position and velocity of the vehicle.

First, we present a similar motivating example as in [5], shown in Figure Fig. 4.2. The leader car l brakes at time $t1$ with its full braking power, -B. But there is a delay before the follower receives a wireless communication update about the behavior of car l, at which point the follow car applies braking power $-b$. But the deceleration is too little and too late, and a collision occurs. The choice of acceleration must be accurate, even when it experiences delays between communication updates from the car ahead.

However, the controller cannot drive the car indefinitely while waiting for updates about its environment. If no update is received, it will timeout and request driver assistance. In this system, the cars wirelessly broadcast their position and velocity at a set frequency, but these broadcasts may not be received. In our model of the controller, we define this timeout by the symbolic parameter ε. While other parameters in the model are symbolic, many of them are not flexible in a particular implementation of the system, for example the upper and lower bounds on acceleration and braking, which are defined by the physical limits of the car. However, the timeout parameter ε can be set arbitrarily in software. It is crucial to the efficiency of the controller to set ε optimally, however finding the optimal value for ε is nontrivial, as we will discuss in greater detail in Section 5.5.

In Controller 5, we model two cars driving along a straight road, where the lead car may choose its acceleration arbitrarily (line 5.4), but the acceleration of the follow car is chosen by an automated control system (line 5.5). We assume that neither car may travel backward (line 5.10). The adaptive cruise control system presented in Controller 5 is specified in Differential Dynamic Logic (d$\mathcal{L}$) [61, 84, 85]. The controlled follow car f has state variables x_f, v_f and a_f to represent its position, velocity and acceleration (similarly for the leader car l). The continuous dynamics for f are described by the differential equation system $x_f' = v_f, v_f' = a_f$ (lines 5.8, 5.9). The position and velocity of the vehicles change continuously, however we don't assume permanent control over the acceleration, since V2V message updates are only available discretely. In this section, we assume that upon receipt of a packet from the leader car with current values of x_l and v_l, the controller for car f sets its acceleration and maintains it until receiving a subsequent message from l. This behavior is captured by the nondeterministic repetition * in line 5.2. We assume a maximum acceleration for both cars and denote it by $A > 0$. We also assume a maximum braking power $B > 0$.

In order to automatically control a car, a computer must have specific algorithms for setting acceleration and these algorithms must be guaranteed to keep the two cars separated under all circumstances. The complex interactions between discrete and continuous components make even the simplest control systems challenging to implement safely. The formulas required to calculate an appropriate acceleration in every circumstance quickly become very complex.

Controller 5 presents a formula for setting the acceleration of car f in lines 5.11-5.13. It is a function of the relative position D and the velocities v_f, v_l of the two cars. It also relies on the choice of the timeout parameter, ε, which determines how long the automated control will wait for an update before requesting assistance from the human driver. In line 5.11, a is designed to be the greatest acceleration such that if car l is braking maximally, and car f accelerates at rate a for up to ε time and then also brakes maximally, the two cars will not collide. Note that a may be positive or negative. In line 5.12, b is designed to be the least braking required to bring f to a stop before the point where l would stop, if it is applying maximum braking $-B$.

In line 5.13, we set the acceleration a_f to be a only if it would not cause the car to stop before

Model 5 Verified Adaptive Cruise Control (ACC)

$$(x_f \leq x_l \wedge v_f^2 \leq v_l^2 + 2DB) \rightarrow [\texttt{ACC}](x_f \leq x_l) \tag{5.1}$$

$$\begin{aligned}
\texttt{ACC} &\equiv (ctrl; dyn)^* && (5.2)\\
ctrl &\equiv \ell_{ctrl} \parallel f_{ctrl}; && (5.3)\\
\ell_{ctrl} &\equiv (a_\ell := *;\ \ ?(-B \leq a_\ell \leq A)) && (5.4)\\
f_{ctrl} &\equiv a_f := a_f(v_f, v_l, D, \varepsilon) && (5.5)\\
D &\equiv x_l - x_f && (5.6)\\
dyn &\equiv (t := 0;\ t' = 1, && (5.7)\\
&\quad x_f' = v_f,\ v_f' = a_f, && (5.8)\\
&\quad x_\ell' = v_\ell,\ v_\ell' = a_\ell && (5.9)\\
&\quad \&\ v_f \geq 0 \ \wedge\ v_\ell \geq 0 \ \wedge\ t \leq \varepsilon) && (5.10)
\end{aligned}$$

$$a \equiv \frac{\sqrt{B^2\varepsilon^2 - 4Bv_f\varepsilon + 8BD + 4v_l^2} - B\varepsilon - 2v_f}{2\varepsilon} \tag{5.11}$$

$$b \equiv \frac{-v_f^2}{2(D + \frac{v_l^2}{2B})} \tag{5.12}$$

$$a_f(v_f, v_l, D, \varepsilon) := \begin{cases} A & \text{if } a \geq A \\ 0 & \text{if } v_f = 0 \wedge a \leq 0 \\ a & \text{if } a \geq \frac{-v_f}{\varepsilon} \wedge -B \leq a \\ b & \text{if } a < \frac{-v_f}{\varepsilon} \wedge -B \leq b \\ -B & \text{o.w.} \end{cases} \tag{5.13}$$

---

time ε, otherwise b is chosen. The choice of acceleration is also limited by the physical bounds of the car, A and $-B$. Finally, if the car is stopped, it may either accelerate or remain stopped, but it can not be made to travel backward.

Using the theorem prover KeYmaera [2], we proved the safety property in line 5.1, which states that while the controller for car f chooses its acceleration based on the formula in line (5.13), the cars will not collide. To prove this property, we must assume car f is initially behind car l and the state is initially in a controllable region, i.e. if both cars were to immediately brake maximally, they could avoid a collision. This property is expressed by the following formula, which we prove is invariant for ACC (i.e. if it holds initially, it continues to hold through all executions

of ACC):

$$v_f^2 \leq v_\ell^2 + 2DB \tag{5.14}$$

This assumption can be seen as an antecedent in line 5.1. This proof of safety requires 334 user interactions, has 661 nodes and takes just 24.2 seconds to prove on a laptop computer. The proof file may be downloaded online from `http://www.ls.cs.cmu.edu/pub/acc/`.

5.4 Optimality

To prove that the function a_f in Controller 5 is optimal, we show that if a_f were chosen to be $a_f + \delta$ for any $\delta > 0$, there could be a collision. Specifically, we show that in the worst case where l is already braking maximally and car f accelerates at rate $a_f + \delta$ for the maximum duration of ε, then even when car f applies maximum braking at time ε, it is not able to avoid a collision.

First, we remind the reader of some useful definitions which can be derived in the standard way from kinematic equations and by integrating the ODE in line 5.8 of Controller 5. Let $x_c[a^t]$ be the position of car c, after accelerating at rate a for time t. Similarly, let $v_c[a^t]$ be the velocity of car c after accelerating at rate a for time t.

$$x_c[(a,t)] = \frac{1}{2}at^2 + v_c t + x_c \qquad v_c[(a,t)] = at + v_c$$

We let $x_c[(b, stop)]$ be the location where car c comes to a stop after decelerating at rate b.

$$x_c[(b, stop)] = x + \frac{v^2}{-2b}$$

From these familiar definitions we can compute the position of car c after it has accelerated at rate a for time ε and then brakes maximally until it comes to a stop:

$$x_c[(a,\varepsilon);(-B, stop)] = x_{f,a}(\varepsilon) + \frac{v_{f,a}(\varepsilon)^2}{2B} \tag{5.15}$$

Case 1: $a_f = a = \frac{\sqrt{B^2\varepsilon^2 - 4Bv_f\varepsilon + 8BD + 4v_l^2} - B\varepsilon - 2v_f}{2\varepsilon}$

Through algebra and the assumption that $a_f \geq -v_f/\varepsilon$ given in line (5.13) of Controller 5, we can show that $x_{f,a_f}(t_{stop}) = x_{l,-B}(t_{stop})$. So the system is safe in this scenario for acceleration choice a_f. This is not surprising since we derived a formal proof of safety for all scenarios in Section 5.3. (Note that since we are assuming cars are infinitesimal points, $x_f = x_l$ is still considered safe, but $x_f > x_l$ violates our safety condition from line 5.1.)

However, when we choose to accelerate at rate $a_f + \delta$, we can derive the following:

$$x_f[(a_f + \delta, \varepsilon);(-B, stop)] \tag{5.16}$$

$$= \frac{1}{2}(a_f + \delta)\varepsilon^2 + v_f\varepsilon + x_f + \frac{((a_f + \delta)\varepsilon + v_f)^2}{2B} \tag{5.17}$$

$$> \frac{1}{2}a_f\varepsilon^2 + v_f\varepsilon + x_f + \frac{(a_f\varepsilon + v_f)^2}{2B} \tag{5.18}$$

$$= x_l + v_l^2/(2B) \tag{5.19}$$

$$= x_l[(-B, stop)] \tag{5.20}$$

The formula in eq. (5.18) is equal to the formula in eq. (5.19) by design through the definition of a_f. It can be easily checked by substituting a for a_f. So, we have shown the system is unsafe in this scenario for the acceleration choice $a_f + \delta$ for any $\delta > 0$. As a result, we know that a_f is the maximal acceleration choice available for this case.

Case 2: $a_f = b = \frac{-v_f^2}{2(D+\frac{v_l^2}{2B})}$

The proof of optimality for this case follows similarly to that in Case 1, however it does not depend on ε, as the cars come to a stop before ε. In Controller 5, $b \leq a$, so in this case $a_f = b \leq a < -v_f/\varepsilon \leq 0$. Therefore, from the definition of a_f and the fact that $a_f < 0$, we get

$$x_f + \frac{v_f^2}{-2a_f} = x_l + \frac{v_l^2}{2B}.$$

As a result, there is an epsilon $\delta > 0$ such that,

$$\begin{aligned} x_f[(a_f + \delta, stop)] &= x_f + \frac{v_f^2}{-2(a_f + \delta)} \\ &> x_l + \frac{v_l^2}{2B} = x_l[(-B, stop)] \end{aligned}$$

So the system is unsafe in this scenario, and a_f is the maximum acceleration choice available for this case.

All other cases are trivial or extend Cases 1 and 2 trivially.

5.5 Efficiency Analysis

Intuitively, we believe that if the follow car can expect to get more frequent communication updates on the position and velocity of the car ahead, it may follow more closely, and therefore improve efficiency. This intuition is quantified by the assignment of a_f in line 5.11 of Controller 5. When the maximum time between updates, ε, is small, the acceleration can be set to a larger value, as demonstrated later in Fig. 5.2.

Unfortunately, reducing ε is not cost-free, since every time the follow car does not receive a communications update within ε, human assistance is required. We want to reduce the frequency of requests for driver intervention, so it might make sense to set the timeout, ε, to be large.

To study this tradeoff quantitatively, we will think of the efficiency of the system as the ratio of the control space we can access averaged over the state space. The ratio of the control space that we can access at a single point in the state space is the normalized acceleration $\bar{a}_f$:

$$\bar{a}_f(v_\ell, v_f, D, \varepsilon) = \frac{a_f(v_\ell, v_f, D, \varepsilon) + B}{A + B}.$$

Since we would like our system to work with little or no required human intervention, we assign the same efficiency value when the system returns control to the driver as we do when it brakes maximally (both will be 0).

Based on our invariant from (5.14), we define the maximum possible safe velocity of the follower $s_f(v_\ell)$ as

$$s_f(v_\ell) = \min\{(v_\ell^2 + 2DB)^{1/2}, v_{\max}\}.$$

We can now calculate the efficiency Eff_{a_f} of the a_f controller as a function of timeout ε as follows:

$$\text{Eff}_{a_f}(\varepsilon) = \frac{1}{Z} \int_0^{D_{\max}} \int_{v_{\min}}^{v_{\max}} \int_{v_{\min}}^{s_f(v_\ell)} \bar{a}_f \, dv_f dv_\ell dD.$$

where $D_{\max}$ is the maximum distance at which the following car can receive updates from the leading car and $v_{\min}$ and $v_{\max}$ are the minimum and maximum permitted velocities. In our computations, we take $v_{\min}$ to be 45 miles per hour and $v_{\max}$ to be 75 miles per hour; this is a typical range for highway driving speeds. We set $D_{\max}$ to be 200 meters. We assume a uniform probability distribution for the initial state over the state space. Z is the volume of the state space, i.e., the integral of 1 over the same region. Bounds on acceleration and braking will be determined by the capabilities of specific vehicles, but in our analysis we use $A = 2$, and $B = 10$.

Note that any choice of acceleration in $[-B, a_f]$ maintains safety. We use $\bar{a}_f$ to measure efficiency because it gives us the ratio of the allowed interval of accelerations that we can safely access at each point in the state space.

However, this is not the whole picture. We expect that we will not be able to receive updates instantaneously, but will instead receive them within some period of time with some probability. The probability that we successfully receive a given transmission will depend on the distance between the vehicles. In their paper [83], Killat and Hartenstein give a model for this reception probability in terms of distance D based on the Nakagami fading model:

$$r(D, \psi) = \left(1 + 3\frac{x^2}{\psi^2} + \frac{9}{2}\frac{x^4}{\psi^4}\right) \exp\left(-3\frac{D^2}{\psi^2}\right),$$

where ψ is the transmission power in meters. For simplicity, we will set $\psi = 100$ for the rest of the chapter. We will then let $r(D) = r(D, 100)$. The resulting relationship between distance and reception probability is shown in Fig. 5.1.

For concreteness, we assume that cars broadcast their position and velocity at a frequency of 10 Hz. Our techniques apply for other values, though 10 Hz is used in, e.g., [76, 78]. If the cars stay at a constant distance D, the probability that we receive an update within ε seconds is then

$$1 - (1 - r(D))^{\lfloor 10\varepsilon \rfloor}.$$

This formula is not applicable, however, because the distance does not stay constant when both cars move with different velocities or different accelerations. To account for this, we use the initial position, velocity, and acceleration of each car to recalculate the distance between the cars each time an update is sent (recall that $x_c[(a, t)]$ is the position of car c after accelerating at rate a for time t, and that position, velocity and acceleration appear in these terms). So, the probability

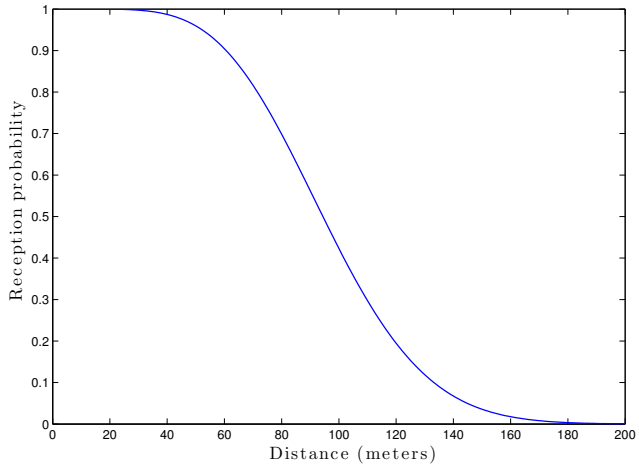

Figure 5.1: Reception probability as a function of distance.

that we receive an update within ε seconds is

$$p(\varepsilon, D, a_\ell, a_f, v_\ell, v_f) = \\ 1 - \prod_{i=1}^{\lfloor 10\varepsilon \rfloor} \left(1 - r\left(x_l[(a_l, \frac{i}{10})] - x_f[(a_f, \frac{i}{10})]\right)\right).$$

We take the acceleration of the follow car to be the value a_f returned by the a_f controller. The acceleration of the lead car is not known, so we instead take the average reception probability over all choices of acceleration for the lead car (the formula is easily modified to assume constant velocity, or any probability distribution over acceleration choices):

$$\bar{p}(\varepsilon, D, a_f, v_\ell, v_f) = \frac{1}{A + B} \int_{-B}^{A} p(\varepsilon, D, a_\ell, a_f, v_\ell, v_f)\, da_\ell.$$

Now we can define the quantity $\text{Eff}_{\text{rec}}(\varepsilon)$, which captures the average likelihood over the state space that we receive an update within timeout ε, as follows:

$$\text{Eff}_{\text{rec}}(\varepsilon) = \frac{1}{Z} \int_0^{D_{\max}} \int_{v_{\min}}^{v_{\max}} \int_{v_{\min}}^{s_f(v_\ell)} \bar{p}\, dv_f dv_\ell dD.$$

Combining the reception probability and the acceleration, we can calculate the expected efficiency of the whole system:

$$\text{Eff}(\varepsilon) = \frac{1}{Z} \int_0^{D_{\max}} \int_{v_{\min}}^{v_{\max}} \int_{v_{\min}}^{s_f(v_\ell)} \bar{a}_f \bar{p}\, dv_f dv_\ell dD.$$

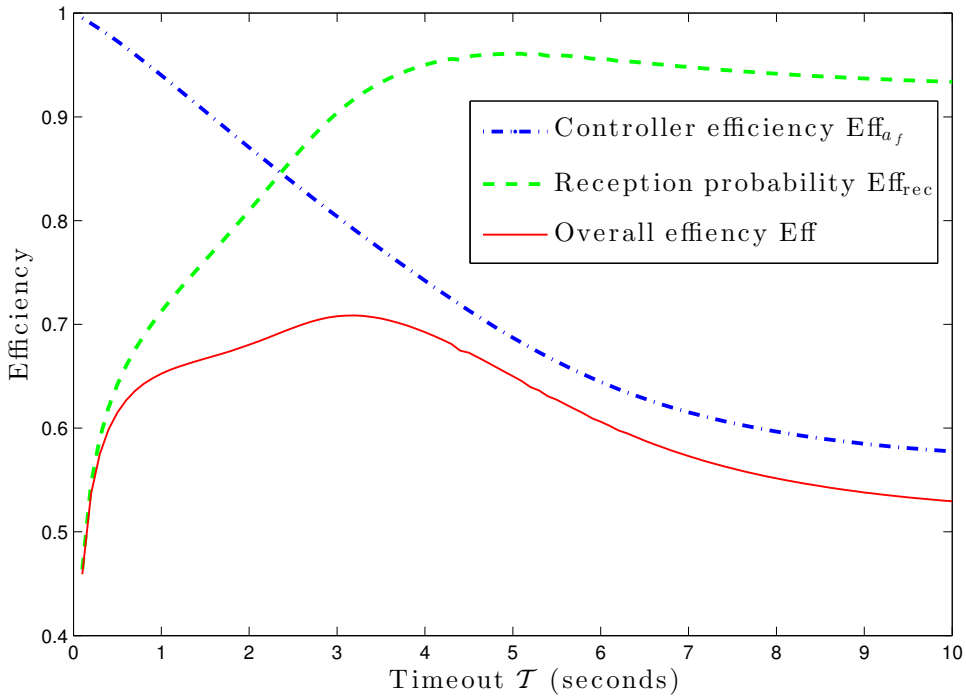

Figure 5.2: Reception probability Eff_{rec}, controller efficiency Eff_{a_f}, and overall efficiency Eff as a function of timeout ε.

We show the three functions Eff_{a_f}, Eff_{rec}, and Eff in Fig. 5.2. Eff_{a_f} decreases as ε decreases, since a longer timeout forces the controller to make more conservative decisions. Eff_{rec} initially increases with ε, as the probability we successfully receive an update increases for a longer timeout. At higher values of ε, the following car may not receive an update from the leading car for a longer period of time. Even though we do not know the behavior of the leading car, the controller must take into account the possibility that the leading car has been continuously applying maximum braking since the last update was received. To ensure safety, the following car must then choose lower acceleration values. However, in much of the state space, the leading car is not actually braking, meaning that the distance to the leading car is increasing in much of the state space. This causes the reception probability to decrease slightly. Eff starts low because we are less likely to receive an update in a short amount of time, then increases because we are more likely to receive updates, and then decreases because the longer timeout forces the controller to make more conservative decisions. It achieves a maximum value of 0.709 at a timeout $\varepsilon = 3.2$ seconds.

5.6 Conclusions

In this chapter, we present a symbolic controller for automated car control on a straight road. We identify its safe region and formally verify that it prevents collisions. This strong formal

guarantee is needed to ensure the safety-critical functioning of the system. We then investigate a particular instance of the controller with exact values for parameters such as update frequency, signal strength and maximum braking and acceleration. We find the timeout value for communication updates that maximizes the range of safe accelerations over the state space. Although we stepped through the analysis using a relatively simple model of wireless communication, our method is general enough that it could be applied using a more complex communication model tailored to the system being analyzed.

Since the probability of receiving a message from the car ahead depends heavily on the distance between the two cars, another reasonable controller might be one which adjusts the timeout as a function of the distance between the two vehicles. This could be interesting future work, and due to the generality of our analysis techniques, the primary challenge will be in the verification step. We may also be able to allow a closer following distance if we incorporate more realistic probability distributions over the state space, rather than uniform distribution.

Chapter 6

Time-Triggered Refines Event-Triggered

Hybrid systems are so called because they have a tight coupling of both discrete and continuous components. The continuous components of a hybrid system are usually easy to identify as they often come from a physical source, like gravity, and are defined by differential equations that are governed by the environment. This also means they fall largely outside of the design space for the system as a whole. Engineers who build hybrid systems will spend most of their time designing and debugging discrete components, for example, software that takes in sensor inputs and then changes actuation. It is these discrete changes in actuation that give us the prototypical graphs of hybrid systems' behavior, like those seen in Fig. 4.2.

But there is an additional source of discrete behavior that also commonly arises in hybrid systems; it comes from how continuous behavior is measured. Continuous values, like position or velocity, are often delivered from sensors or via communication. In either case, these packets or sensor updates are delivered at discrete time intervals. This means that a control decision must be made knowing that updated information may not be available for some time into the future, and the control choice must conservatively take that into account. When the time delay between sensor or communication updates is explicitly modeled, the system is called *time-triggered*. A controller for a time-triggered system would make choices like, "Accelerate at rate a until the next sensor update."

Because time-triggered systems have to make the right choice until the next sensor update, their controllers can be tricky to get right. On top of that, explicitly modeling the sensor delays increases the complexity of the system models. These challenges combine to make time-triggered models tough verification problems. As a result, it is common to make a simplifying assumption that the sensors have continuous access to the values they are measuring. This turns a time-triggered model into an *event-triggered* (also called *event-driven*) model. A controller for an event-triggered system could make choices like, "When the car is 10 feet away from the stop sign, start braking." While this simplification makes modeling and analyzing the behavior of the system easier, continuous sensing is usually not a physical possibility, since it's easy to miss the exact moment when the car is 10 feet away from the stop sign.

This distinction between event-triggered and time-triggered models is an important one [86, 87]. It is usually a modeling decision that is made early on in the design process and considered a tough one to reverse. However, in this chapter we will compare these models using differential refinement logic and, through this lens, we find that they are formally relatable, and perhaps not

so fundamentally different after all.

In Section 6.1, we provide a generic model for event-triggered systems. Because their controllers are more directly connected to the physical dynamics through continuous sensing, event-triggered systems are often easier to verify than time-triggered models. From this event-triggered model, a controller for a time-triggered system can be automatically derived which inherits the proof of safety from the event-driven system. This time-triggered model is presented in Section 6.2, and we prove that it refines the event-triggered model using the dR$\mathcal{L}$ proof calculus in Section 6.3. We then use this refinement relation to simplify the proof of a challenging time-triggered system in Section 6.4. We present an alternate proof of safety for the local lane control case study originally presented in Section 4.5.1. Using the event-/time-triggered refinement property, which was made possible by dR$\mathcal{L}$, the number of interactive steps is reduced by 80%, and the total computation time by 74%. These are significant improvements over the original proof in d$\mathcal{L}$.

While we present just one case study that builds on the proof of refinement between event-triggered and time-triggered systems, we want to make it clear to the reader that we believe the verification of many time-triggered systems could benefit by also building on this proof of refinement. Because of the discrete nature of sensors and computers, a vast majority of safety-critical CPS are time-triggered. And many of those operate as a switched system, like the generic model presented in Model 7. In other words, they have a normal operating mode, but after some threshold is reached, they have a discrete switch in behavior (e.g. evading another aircraft or braking to stop in time for a stop light). Many cyber-physical systems will have a collection of such switching conditions, and we leave as an extension of this work proofs for these more complicated systems. Each of these systems can immediately inherit the proof of refinement to show that it refines its event-triggered counter part. We expect that each of these systems would see a similar reduction in the number of required proof steps as the local lane control case study.

6.1 Event-triggered Model

In this section we introduce a generic template for an event-triggered model of a hybrid system.

Model 6 models an event-triggered system. The system may evolve continuously until an event triggers the controller. The controller can then switch to a different mode. For example, if the system is a car and the controlled variable is acceleration, the event could be that the car passes some point on the road at which time the controller immediately switches into braking. The controller can also change the control variable at any time, even before the event trigger, but

Model 6 Event-triggered model

$$\texttt{event}^* \equiv (ctrl_{Ev}; dyn_{Ev})^* \tag{6.1}$$

$$ctrl_{Ev} \equiv a := c \cup (a := *; ?\text{Safe}(x)) \tag{6.2}$$

$$dyn_{Ev} \equiv t := 0; x_0 := x; ((x' = f(a), t' = 1 \,\&\, H(x) \wedge E(x)) \tag{6.3}$$

$$\cup (x' = f(a), t' = 1 \,\&\, \sim H(x) \wedge E(x))) \tag{6.4}$$

it has the additional guarantee that it will be able to change the control variable exactly when the event occurs.

In Model 6 we can see that the event-triggered model follows the expected high-level structure: discrete control $ctrl_{Ev}$, followed by the continuous evolution dyn_{Ev}, and then nondeterministically repeat these steps as indicated with * in (6.8). In (6.10), if Safe(x) is satisfied, the controller can non-deterministically set the acceleration. Safe(x) will also contain some limits on how the control variable may be set. A good example of this could be physical limits on the control variable. When Safe(x) is not satisfied, the controller must switch to $a := c$.

We model the continuous dynamics of the system in (6.13) and (6.14). The differential equation $x' = f(a)$ is a system of differential equations, which depends on the constant control variable a. $E(x)$ is the evolution domain for the system. This is a domain which the system is not able to evolve beyond and often comes from some physical limit. For example, in the model of local lane control for cars (Model 1), we have an evolution domain of $v \geq 0$, since braking should not cause the car to start moving backwards. $H(x)$ is the event trigger for the system. It must define a closed domain. When the system reaches the boundary of this domain, the evolution is forced to stop, allowing the controller to execute. After the controller executes, the system must be able to continue evolving, thus the second differential equation in (6.14). Notice that the system of differential equations and the evolution domain stay the same. However, we use $\sim H(x)$ to represent the topological closure of the complement of $H(x)$. This means that once the event has been passed and the controller executed, the system will still evolve whether or not the controller made a safe choice, making it possible to detect all unsound control choices in the verification step.

Looking at $ctrl_{Ev}$ and dyn_{Ev} together, we see that whenever we are on the boundary of $H(x)$ it is because we have detected an event. Thus it is imperative that $ctrl_{Ev}$ only be able to choose the left branch $a := c$ at that moment. This means that Safe(x) should only be satisfied if we are inside $H(x)$ and not on its boundary. When Safe(x) is defined in this way, proving safety for Model 6 boils down to making sure that the specific choice $a := c$ is safe. This significantly reduces the verification task.

Variables t and x_0 in (6.13) are never read anywhere in the program, and are therefore considered "ghost" variables, since they don't affect the state of the program. However, they do aid in the refinement verification, since they provide an anchor point between Model 6 and Model 7.

6.2 Time-triggered Model

The template for a time-triggered model, presented in Model 7, again loops between a discrete control action $ctrl_t$, and continuous evolution dyn_t, as show in (6.5). The primary difference between this model and Model 6 is that the time-triggered model can only ensure that the controller will be able to take a control action at least every ϵ seconds. This is expressed in line (6.7) by first setting t to zero, and then evolving continuously along the differential equations, but only within the evolution domain of $t \leq \varepsilon$. Notice that the event trigger $H(x)$ is *not* in the evolution domain for the differential equation in the time-triggered model.

Another major difference between the time-triggered model and the event-triggered model is in the discrete controller. Because the discrete controller in the time-triggered model must make

Model 7 Time-triggered model

$$\texttt{time}^* \equiv (ctrl_t; dyn_t)^* \tag{6.5}$$

$$ctrl_t \equiv a := c \cup (a := *; ?\text{Safe}_\varepsilon(x, a)) \tag{6.6}$$

$$dyn_t \equiv t := 0; x_0 := x; (x' = f(a), t' = 1 \,\&\, t \leq \varepsilon \wedge E(x)) \tag{6.7}$$

a choice that will be safe for up to ε time, we have to change the guard on the nondeterministic assignment of control variable u in (6.6). The guard $\text{Safe}_\varepsilon(x, u)$ depends both on the current choice of u and the time duration ε, in addition to the current state x.

6.3 Proof of Refinement

We expect any safe controller of the time-triggered model to be more conservative than even the most admissible (but still safe) controller for the event-triggered model. In other words, we expect the time-triggered model to not have as wide a range of control choices as the event-triggered model. This is because the event-triggered model has continuous access to sensor data, while the time-triggered model only samples it discretely. As a result, we expect the behavior of the more conservative time-triggered controller to be a subset of the possible behaviors of the event-triggered model. More formally, we expect the time-triggered model to refine the event-triggered model. This is great news for us, since generally speaking, event-triggered models are far easier to verify, but time-triggered models are more reasonable to implement. Now all we have to do is take advantage of this refinement relation in the proof structure using dRL refinement proof rules.

For example, suppose that we want to implement a time-triggered system that always satisfies some safety condition, ϕ. We write the condition that our time-triggered model, $\texttt{time}^*$, always satisfies ϕ using the box modality: $[\texttt{time}^*]\phi$. We would first apply the $[\leq]$ rule, as below, to split the property into two sub-goals. First, that an event-triggered model satisfies the safety condition, $[\texttt{event}^*]\phi$. And second, that the original time-triggered model refines the event-triggered model, $\texttt{time}^* \leq \texttt{event}^*$.

$$\frac{\Gamma \vdash [\texttt{event}^*]\phi, \Delta \qquad \Gamma \vdash (\texttt{time}^* \leq \texttt{event}^*), \Delta}{\Gamma \vdash [\texttt{time}^*]\phi, \Delta}\ [\leq]$$

Recall that $\texttt{event}^*$ and $\texttt{time}^*$ are generic templates for event- and time-triggered systems. Without specific values assigned to crucial components, for example the controllers $ctrl_{Ev}$ and $ctrl_t$, it will not be possible to close this proof. However, we can use dRL proof rules to significantly simplify the remaining open goals. This proof is presented in full in Fig. 6.2-6.5.

In this proof, we assume that the event-trigger is also an invariant for $\texttt{event}^*$. This is a reasonable assumption to make, since the event trigger can be thought of as the last possible moment when switching to the control choice will still guarantee safety for the system. Consider the simple example of an event-triggered controller for a car, where the car applies the brakes 10 feet before a stop sign. This means that we define $H(x) \equiv d \geq 10$, where d is the distance

between the car and the stop sign. This event trigger will not be an invariant for the system, since the car will pass the 10 feet away mark after it starts braking. But also notice that this event trigger doesn't take into account the velocity of the car. If the car is traveling fast enough, braking 10 feet away from the stop sign may not be enough distance for the car to stop in time.

Instead, a better (and provably safe) event trigger will be symbolically defined. By defining the event trigger to be the last possible moment when the car can brake and not run the stop sign, we get $H(x) \equiv d \geq \frac{v^2}{2B}$, where d again is the distance between the car and the stop sign, v is the car's velocity, and B is the braking force that the car will necessarily switch to when the event-trigger happens (i.e. when it reaches the boundary of $H(x)$). In this case, when the car hits the boundary of $H(x)$ and starts to brake, it then evolves along the boundary of $H(x)$, since while the distance to the stop sign decreases, so does the velocity of the car. This makes $H(x)$ an invariant of the system.

We also require for this proof that the solution to the differential equation exists and is expressible as a term in dR$\mathcal{L}$. We define $S_{x,a}(t)$ to be the solution to $x' = f(a)$ at time t.

Additionally, the proof of safety for the event-triggered model (i.e. $[\texttt{event}^*]\phi$) must rely on the same invariant as the time-triggered system in order for the proof to be reused in the proof of ($\texttt{time}^* \leq \texttt{event}^*$). For this proof, we require that this invariant also be the event-trigger, as discussed above.

In the proof that $\texttt{time}^*$ refines $\texttt{event}^*$ (Figures 6.1-6.5), four open goals remain to be proved (indicated in maroon) based on specific implementation choices. Note that only one of these goals is a dynamic property containing a differential equation or hybrid programs - the safety property for the event triggered system! The remaining three open goals are for static properties; they are all expressed in the decidable first-order logic over the reals ($\text{FOL}_{\mathbb{R}}$).

Open Goal 1 - Discrete Controllers Satisfy Refinement: $\Gamma \wedge H(x) \wedge \text{Safe}_\varepsilon(x, \overline{a}) \vdash \text{Safe}(x)$

Fig. 6.1 is the proof branch where we show that the discrete controllers satisfy the refinement relationship, which depends on the specific choices of $\text{Safe}(x)$ and $\text{Safe}_\varepsilon(x, a)$. The open goal in Fig. 6.1 requires that until reaching the event-trigger, $\text{Safe}_\varepsilon(x, a)$ implies $\text{Safe}(x)$.

Open Goal 2 - Evade Mode: $\Gamma \wedge H(\overline{x_0}) \wedge 0 \leq t \leq \varepsilon \wedge E(\overline{x}) \wedge \overline{x} = S_{\overline{x_0},c}(t) \vdash H(\overline{x})$

The open goal in Fig. 6.3 requires that the control choice $a := c$ is enough to ensure that the system will not cross the event-trigger boundary within time ε. In other words, if the event trigger has not been reached initially (i.e. $H(\overline{x_0})$ is satisfied), then for all time t between 0 and ε, the solution at time t ($\overline{x} = S_{\overline{x_0},c}(t)$) should satisfy $H(\overline{x})$.

Open Goal 3 - Normal Mode: $\Gamma \wedge H(\overline{x_0}) \wedge \text{Safe}_\varepsilon(\overline{x_0}, \overline{a}) \wedge 0 \leq t \leq \varepsilon \wedge E(\overline{x}) \wedge \overline{x} = S_{\overline{x_0},\overline{a}}(t) \vdash H(\overline{x})$

The open goal in Fig. 6.4 is very similar to Open Goal 2 from Fig. 6.3, except that the control choice is nondeterministic ($a := *$), since we are in normal mode rather than evade. We represent this arbitrary value with the variable $\overline{a}$. Of course, we are only able to be in normal mode if certain requirements about the state are satisfied. These requirements are exactly expressed by $\text{Safe}_\varepsilon(\overline{x_0}, \overline{a})$. It requires that the control choice ($a := *;\ ?(\text{Safe}_\varepsilon(x, a))$)

ensures that the system will not cross the event-trigger within time ε. In other words, if the event trigger has not been reached initially (i.e. $H(\overline{x_0})$ is satisfied), and the guard was satisfied initially for any choice of acceleration $\overline{a}$ (i.e. $\text{Safe}_\varepsilon(\overline{x_0}, \overline{a})$ is satisfied), then for all time t between 0 and ε, the solution at time t ($\overline{x} = S_{\overline{x_0},\overline{a}}(t)$) should satisfy $H(\overline{x})$.

Open Goal 4 - Event-Triggered is Safe: $H(x) \wedge \Gamma \vdash [\texttt{event}](H(x) \wedge \Gamma)$

To close the open goal in Fig. 6.5, we must show that the event-triggered model always satisfies an invariant (i.e. that the event-triggered system is safe). For this proof, we require that this invariant also be the event-trigger, as discussed above.

It is important to keep in mind that while the proof that time-triggered refines event-triggered is quite involved, since $\texttt{event}^*$ and $\texttt{time}^*$ are generic templates, this proof can be immediately reused for any time-triggered or event-triggered systems that fit the templates, so this hard work only has to be done once.

Also notice that the first three open goals do not contain hybrid programs; they are expressions in first order logic over the reals, which is a decidable fragment of dR$\mathcal{L}$. We expect these open goal properties to be much easier to verify. Additionally, the final goal, contains an event-triggered hybrid program, which we typically find much easier to verify, as controllers for event-triggered hybrid programs can be directly linked to the physical dynamics of the system through continuous sensing.

When using theorem proving to verify hybrid systems, it is not uncommon to first prove safety for an event-triggered model of the system, and then add in modeling of a time delay and reprove safety for the time-triggered model [88]. Event-triggered models are easier to prove because they avoid the complications introduced by delays and reaction times that are modeled in time-triggered architectures. Now, instead of reproving from nothing, the proof of safety for the time-triggered system can be built on top of the proof of safety for the event-triggered system once and for all.

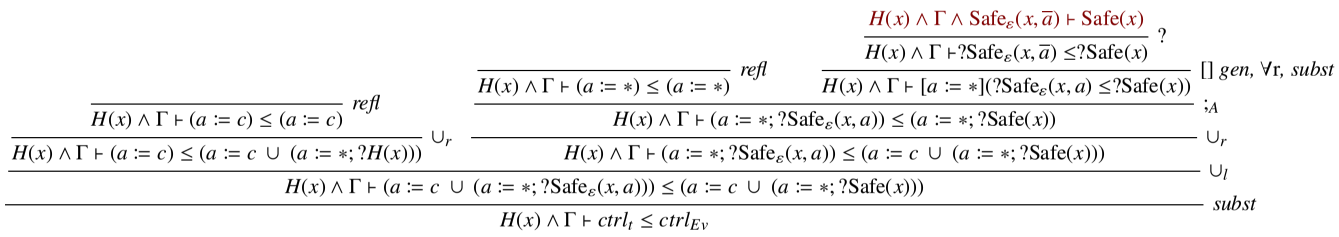

Figure 6.1: Proof in dRL that the discrete time-triggered controller refines the discrete event-triggered controller.

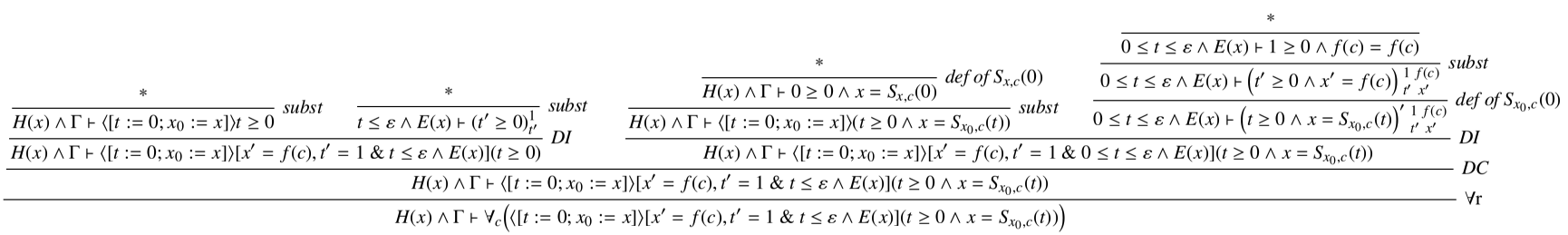

Figure 6.2: We show by differential invariants that time is only increasing ($t \ge 0$) and x is always equal to the solution of its differential equation ($x = S_{x_0,c}(t)$). These invariants hold no matter the chosen value of the control variable c.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{H(\overline{x_0}) \land \Gamma \land 0 \le t \le \varepsilon \land E(\overline{x}) \land \overline{x} = S_{\overline{x_0},c}(t) \vdash H(\overline{x})}}{H(x) \land \Gamma \land \overline{x_0} = x \vdash (0 \le t \le \varepsilon \land E(\overline{x}) \land \overline{x} = S_{\overline{x_0},c}(t)) \to H(\overline{x})}\,{\to}\text{r},\ subst}{H(x) \land \Gamma \land \overline{x_0} = x \vdash \forall_{x,t}\big((0 \le t \le \varepsilon \land E(x) \land x = S_{\overline{x_0},c}(t)) \to H(x)\big)}\,\forall\text{r}}{H(x) \land \Gamma \vdash \forall_{\overline{x_0}}\big(\overline{x_0} = x \to \forall_{x,t}\big((0 \le t \le \varepsilon \land E(x) \land x = S_{\overline{x_0},c}(t)) \to H(x)\big)\big)}\,\forall\text{r},\ {\to}\text{r}}{H(x) \land \Gamma \vdash \langle [t := 0; x_0 := x]\rangle \forall_{x,t}\big((0 \le t \le \varepsilon \land E(x) \land x = S_{x_0,c}(t)) \to H(x)\big)}\,US}{H(x) \land \Gamma \vdash \langle [t := 0; x_0 := x]\rangle \forall_{x,t}\big((x' = f(c) \land t' = 1 \land 0 \le t \le \varepsilon \land E(x) \land x = S_{x_0,c}(t)) \to (x' = f(c) \land t' = 1 \land H(x) \land E(x))\big)}\,simplify}{H(x) \land \Gamma \vdash \langle [t := 0; x_0 := x]\rangle \big((x' = f(c), t' = 1 \,\&\, 0 \le t \le \varepsilon \land E(x) \land x = S_{x_0,c}(t)) \le (x' = f(c), t' = 1 \,\&\, H(x) \land E(x))\big)}\,DR}{H(x) \land \Gamma \vdash \langle [t := 0; x_0 := x]\rangle \big((x' = f(c), t' = 1 \,\&\, t \le \varepsilon \land E(x)) \le (x' = f(c), t' = 1 \,\&\, H(x) \land E(x))\big)}\,DC}{H(x) \land \Gamma \vdash \langle [t := 0; x_0 := x]\rangle \big((x' = f(c), t' = 1 \,\&\, t \le \varepsilon \land E(x)) \le (((x' = f(c), t' = 1 \,\&\, H(x) \land E(x)) \cup (x' = f(c), t' = 1 \,\&\, {\sim}H(x) \land E(x))))\big)}\,\cup_r$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{H(x) \land \Gamma \vdash (t := 0; x_0 := x) \le (t := 0; x_0 := x)}\,refl \qquad H(x) \land \Gamma \vdash [t := 0; x_0 := x]\big((x' = f(c), t' = 1 \,\&\, t \le \varepsilon \land E(x)) \le (((x' = f(c), t' = 1 \,\&\, H(x) \land E(x)) \cup (x' = f(c), t' = 1 \,\&\, {\sim}H(x) \land E(x))))\big)}{H(x) \land \Gamma \vdash t := 0; x_0 := x; (x' = f(c), t' = 1 \,\&\, t \le \varepsilon \land E(x)) \le t := 0; x_0 := x; (((x' = f(c), t' = 1 \,\&\, H(x) \land E(x)) \cup (x' = f(c), t' = 1 \,\&\, {\sim}H(x) \land E(x))))}\,;}{H(x) \land \Gamma \vdash [a := c]\big(t := 0; x_0 := x; (x' = f(a), t' = 1 \,\&\, t \le \varepsilon \land E(x)) \le t := 0; x_0 := x; (((x' = f(a), t' = 1 \,\&\, H(x) \land E(x)) \cup (x' = f(a), t' = 1 \,\&\, {\sim}H(x) \land E(x))))\big)}\,[:=]}{H(x) \land \Gamma \vdash [a := c](dyn_t \le dyn_{Ev})}\,subst$$

Fig. 6.2

Figure 6.3: Proof in dRL that time-triggered continuous dynamics refine event-triggered continuous dynamics in the braking case.

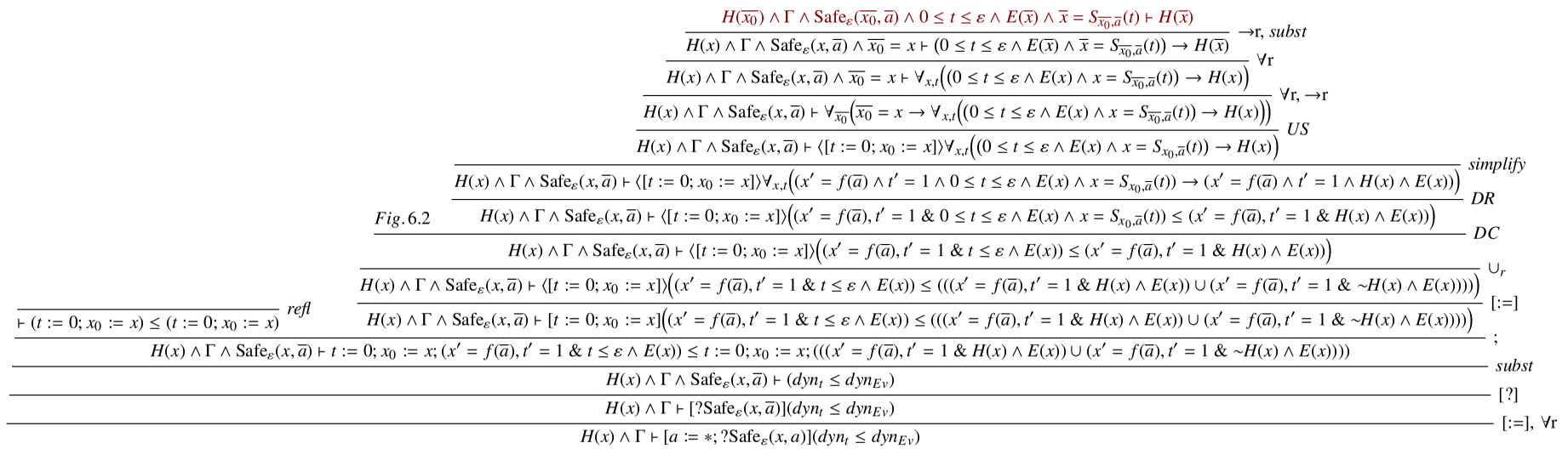

Figure 6.4: Proof in dRℒ that the continuous dynamics of the time-triggered system refines the continuous dynamics of an event triggered system in the acceleration branch.

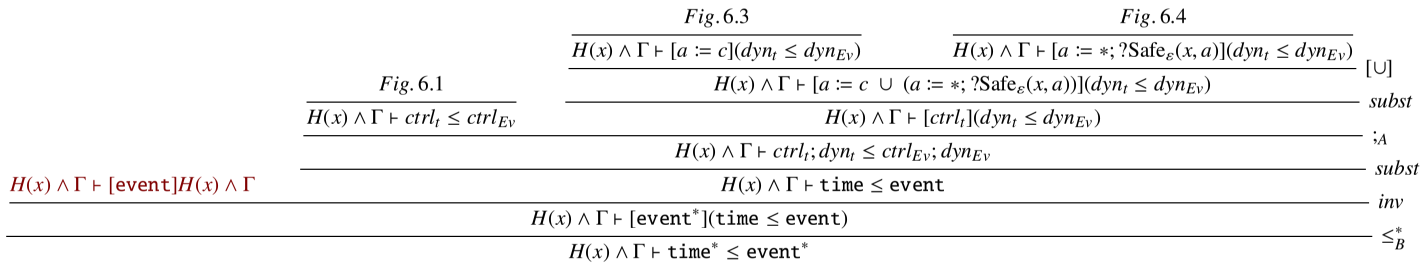

Figure 6.5: Proof in dRℒ that a time-triggered system refines a generic template for an event-triggered system.

6.4 Example: Proof of Local Lane Control Using Refinement

In Model 1 from Chapter 4 we present the local lane control `llc`, a time-triggered model for adaptive cruise control with two cars on a straight lane. We also provide a proof that this model always is safe, i.e. that the follow car never collides with the lead car. However, proving this safety property using the $d\mathcal{L}$ proof calculus in KeYmaera required enormous effort – 656 interactive proving steps. Meanwhile, the proof of safety for the event-triggered model for the same system requires just 4 interactive steps (see Table 6.1). Now that we have a template for an event-triggered system that provably refines a time-triggered system (Section 6.3), we can complete the proof of safety for `llc` with significantly less effort.

Model 8 Event-triggered local lane car control model

$$\texttt{llc}_{Ev} \equiv (ctrl_{Ev}; dyn_{Ev})^* \tag{6.8}$$

$$ctrl_{Ev} \equiv a_l := *; ?(-B \leq a_l \leq A); \tag{6.9}$$

$$(a_f := -B \cup (a := *; ?\text{Safe}(x))) \tag{6.10}$$

$$\text{Safe}(x) \equiv (-B \leq a_f \leq A) \wedge (x_f < x_l) \wedge \left(x_f + \frac{v_f^2}{2B} < x_l + \frac{v_l^2}{2B}\right) \tag{6.11}$$

$$dyn_{Ev} \equiv t := 0; x_0 := x; \tag{6.12}$$

$$((x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l, t' = 1 \;\&\; H(x) \wedge E(x)) \tag{6.13}$$

$$\cup (x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l, t' = 1 \;\&\; {\sim}H(x) \wedge E(x))) \tag{6.14}$$

$$E(x) \equiv v_f \geq 0 \wedge v_l \geq 0 \tag{6.15}$$

$$H(x) \equiv (x_f \leq x_l) \wedge \left(x_f + \frac{v_f^2}{2B} \leq x_l + \frac{v_l^2}{2B}\right) \tag{6.16}$$

$${\sim}H(x) \equiv (x_f \geq x_l) \vee \left(x_f + \frac{v_f^2}{2B} \geq x_l + \frac{v_l^2}{2B}\right) \tag{6.17}$$

Model 8 is an event-triggered model of a local lane controller. We can see that the discrete controller, in (6.11) of this model is checking only that the system is currently in a controllable state. In other words, that the follow car is behind the lead car, and that if both cars were to apply full braking force, the follow car would not overtake the lead car.

The continuous dynamics in lines (6.12 - 6.17) capture both the physical dynamics of the car, as well as the continuous sensors. The continuous sensor must be able to capture the exact moment when the system evolved out of $H(x)$, also known as capturing a "zero crossing" event. In order to ensure that this event is captured, we include the closed evolution domain $H(x)$ to our continuous dynamics in (6.13). This means that exactly when the boundary of the event trigger $H(x)$ is crossed, the discrete controller will have a chance to execute. Whatever control choice is made, we still want the system to be able to evolve beyond the boundaries of $H(x)$, so we include a nondeterministic choice with the same differential equations, but an evolution domain of $\sim H(x)$ which is the topological closure of the inverse of $H(x)$. This is modeled in (6.14).

We model braking as a deceleration, but with the important distinction that when a car brakes

to a stop it does not start moving backward. This behavior is captured by the evolution domain $E(x)$, which is seen in both (6.13) and (6.14)

Model 9 Time-triggered local lane car control model from Model 1

$$\texttt{llc}_t \equiv (ctrl_t; dyn_t)^* \tag{6.18}$$

$$ctrl_t \equiv a_l := *; ?(-B \leq a_l \leq A); \tag{6.19}$$

$$(a_f := -B \cup (a := *; ?\text{Safe}_\varepsilon(x, a))) \tag{6.20}$$

$$\text{Safe}_\varepsilon(x, a) \equiv (-B \leq a_f \leq A) \wedge x_f + \frac{v_f^2}{2B} + \left(\frac{A}{B} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon v_f\right) < x_\ell + \frac{v_\ell^2}{2B} \tag{6.21}$$

$$dyn_{Ev} \equiv t := 0; x_0 := x; \tag{6.22}$$

$$(x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l, t' = 1 \;\&\; t \leq \varepsilon \wedge E(x)) \tag{6.23}$$

$$E(x) \equiv v_f \geq 0 \wedge v_l \geq 0 \tag{6.24}$$

$$\tag{6.25}$$

Model 9 models a time-triggered local lane controller and is a slight variation on Model 1. We make these variations so that it fits the generic model of a time-triggered system presented in Model 7, and therefore automatically inherit the proof of refinement presented in Section 6.3.

We add the ghost variable x_0 in (6.22) to keep track of the initial value of x before each continuous evolution. This is really a proof artifact, as the proof that time-triggered refines event-triggered relies on having access to the value of x_0.

Next, we assume that when a car applies the brakes, it does so with exact braking power $-B$, rather than with some power in the range of $[-B, -b]$. This is because our proof of refinement in Section 6.3 uses a deterministic control choice for the evasive maneuver branch. We leave it as future work to extend this proof to allow nondeterministic controllers in both the normal and evasive modes.

Finally, the controller in Model 1 had an additional control branch which allowed the car to remain stopped once it braked to a stop. We remove this branch because the proof of refinement in Section 6.3 only allows two control modes: normal and evade. Of course, we can argue informally that a stopped car does not pose a risk of colliding with a car in front of it. But we notice that with the more efficient version of the local lane control presented in Model 5, which allows more aggressive choices of acceleration, this branch is no longer needed, as it is covered by the normal operating mode. While each of these changes still constitute reasonable representations of the underlying system, they do lessen the impact of direct comparisons of proof statistics between the two models.

The remainder of this section will prove that Model 9 refines Model 8. In this proof, it will be useful to give concrete definitions to the variables originally introduced in the generic proof that time-triggered systems refine event-triggered systems. We provide definitions here for the restrictions on constants, I, as well as the solutions to the differential equation for position and velocity, given acceleration choice a_i, at time t ($S_{x_i,a_i}(t)$ and $S_{v_i,a_i}(t)$ respectively):

$$\Gamma \equiv A > 0 \wedge B > 0 \wedge \varepsilon > 0 \tag{6.26}$$

$$S_{x_i,a_i}(t) \equiv \frac{1}{2}a_i t^2 + v_i t + x_i \tag{6.27}$$

$$S_{v_i,a_i}(t) \equiv at + v_i \tag{6.28}$$

$\texttt{llc}_t$ Goal 1 - Discrete Controllers Satisfy Refinement:

$$H(x) \wedge \Gamma \wedge \text{Safe}_\varepsilon(x, \overline{a}) \vdash \text{Safe}(x)$$

This property in first order logic over the reals is easy for KeYmaera, finishing almost instantaneously with no user interaction. Notice that Safe(x) is the topological closure of our invariant, $H(x)$, which makes the bulk of this proof trivial.
Link to proof file: `http://www.cs.cmu.edu/~sloos/llct/llct_goal1.key.proof`

$\texttt{llc}_t$ Goal 2 - Evade Mode:

$$H(\overline{x_0}) \wedge \Gamma \wedge 0 \leq t \leq \varepsilon \wedge E(\overline{x}) \wedge \overline{x} = S_{\overline{x_0},c}(t) \vdash H(\overline{x})$$

This goal requires that for any evolution where the car is braking, if the invariant was initially satisfied (i.e. $H(\overline{x_0})$), then the invariant must hold for as long as the car continues braking.
Link to proof file: `http://www.cs.cmu.edu/~sloos/llct/llct_goal2.key.proof`

$\texttt{llc}_t$ Goal 3 - Normal Mode:

$$H(\overline{x_0}) \wedge \Gamma \wedge \text{Safe}_\varepsilon(\overline{x_0}, \overline{a}) \wedge 0 \leq t \leq \varepsilon \wedge E(\overline{x}) \wedge \overline{x} = S_{\overline{x_0},\overline{a}}(t) \vdash H(\overline{x})$$

This goal is where 95% of the interactive steps in the dR$\mathcal{L}$ proof of safety for $\texttt{llc}_t$ occur. Many of the interactive steps are very similar to the interactive steps required for the d$\mathcal{L}$ proof of `llc`. However, in the d$\mathcal{L}$ proof, these steps had to be repeated on multiple branches, particularly if good branch management is not used early on in the proof. Good branch management for a proof as complicated as this one requires more foresight than is possible to expect from even the most experienced users. Using dR$\mathcal{L}$, we have much better control over proof branching and can eliminate repetitive applications of proof rules. The majority of the proving for this goal focuses on reducing the computation time needed to solve the problem by hiding and renaming unnecessary or complicated sub formulas.
Link to proof file: `http://www.cs.cmu.edu/~sloos/llct/llct_goal3.key.proof`

$\texttt{llc}_t$ Goal 4 - Event-Triggered is Safe:

$$H(x) \wedge \Gamma \vdash [\texttt{event}](H(x) \wedge \Gamma)$$

Proving safety for event-triggered systems tends to be much easier than proving safety for time-triggered systems. We see an example of this in Goal 4, which takes only 4 interactive steps to verify. These user interactions are simply to instantiate a $\forall_l$ rule in a useful way that is not yet automatically discovered by the theorem prover.

Link to proof file: `http://www.cs.cmu.edu/~sloos/llct/llct_goal4.key.proof` (Note that this proof file actually provides a proof of safety for the full event-triggered model, $H(x) \wedge I \vdash [\texttt{event}^*](x_f \leq x_l)$. However, the proof relies on using $H(x) \wedge \Gamma$ as an invariant, and therefore $H(x) \wedge I \vdash [\texttt{event}](H(x) \wedge \Gamma)$ is also proved in this file.)

Each of these four goals is a $\mathsf{d}\mathcal{L}$ property, and three of the four are properties in $\text{FOL}_{\mathbb{R}}$. This makes them prime candidates for the theorem prover KeYmaera, and indeed, all four proofs are closed using KeYmaera. The proof statistics are presented in Table 6.1.

The four goals proved in this section are automatically derived from the four goals introduced in Section 6.3. We can now revisit the role that each of these goals plays in the context of a more concrete application, rather than in the context of generic event- and time-triggered systems. The time-triggered controller must make certain that the follow car is safe no matter what the lead car does for up to ε time. This must be a more conservative controller that the event-triggered controller, which required only that the car was safe at present, and did not need to look ahead into the future, since it could rely on instantaneous reaction times via continuous sensing. The purpose of Goal 1 is to formally show that this refinement relationship holds, i.e. that the time-triggered controller is indeed more conservative than the event-triggered controller. In the evade mode, the time-triggered system is braking. We must prove that the level of braking is sufficient to ensure safety for at least ε time, which is accomplished in Goal 2. The normal operating mode is similar to the evade mode, except that we now also have the formula for the time-triggered controller. We check that this choice of acceleration is safe for at least ε time in Goal 3.

But notice that each of these goals mentions time only up to ε. Real safety requires verification not just for some small window into the future, but for all time, which does not show up in these goals. Goal 4 proves the invariant holds for an event-triggered controller, and therefore the event-triggered system is guaranteed safe for any arbitrary amount of time into the future. This, combined with Goals 1-3 shows that the time-triggered system is a refinement of the event-triggered system, and therefore that the time-triggered system inherits the proof of safety for any duration from the event-triggered system.

6.5 Conclusion

In this chapter we present the first formal proof that a generic time-triggered system refines its event-triggered counterpart. This refinement relation is an important one, as it ties together two architecture types that are often considered fundamentally different modeling choices. Event-triggered systems are generally considered easier to verify, while time-triggered systems give a more faithful representation of real-world systems with discrete sensors. By establishing this relationship between the two, we can get the best of both worlds. We first prove properties about event-triggered systems which are significantly easier to verify. By proving a few simple side

[1]This number only reflects the interactive steps done through the theorem prover KeYmaera. It does *not* include any statistics for the reuse of the on-paper proof from Section 6.3. Those proof steps are included in the line $\texttt{time}^* \leq \texttt{event}^*$ and we do not include these steps in the final tally as this proof may be reused and can therefore be considered a lemma.

[2]This computation was run on a different machine, and is therefore not comparable to the other values in this column.

conditions, the resulting proofs extend easily in dR$\mathcal{L}$ to verify the time-triggered systems, which give a more realistic model of discrete sensors.

The total number of interactive steps required to prove safety for this adapted time-triggered model using dR$\mathcal{L}$ is 133, an 80% reduction when compared to the version proved using d$\mathcal{L}$, which took 656 interactive steps. Computation time also decreased by 74%. While there are some differences in the structure of the compared models (stated above) that complicate direct comparisons, these are significant improvements over the original proof in d$\mathcal{L}$, and should be considered strong evidence that dR$\mathcal{L}$ can significantly reduce the number of user interactions and computation required for proving. Reducing computational requirements and user interactions is a crucial step to enabling formal verification of more complex hybrid systems.

Proof statistics for local lane controller, with and without refinement			
	Interactive Steps	Computation Time (seconds)	Proof Nodes
llc from Chapter 4 (using d$\mathcal{L}$)	**656**	**329.8**[2]	**924**
time* ≤ event* (reusable proof)	50	-	-
$\texttt{llc}_t$ Goal 1 - Discrete Controllers	0	0.6	16
$\texttt{llc}_t$ Goal 2 - Evade Mode	0	2.7	30
$\texttt{llc}_t$ Goal 3 - Normal Mode	79	8.4	126
$\texttt{llc}_t$ Goal 4 - Event Triggered (proof of safety for $\texttt{llc}_{Ev}$)	4	73.3	140
Total – $\texttt{llc}_t$ (using dR$\mathcal{L}$)	**$\sum$ 83**[1]	**$\sum$ 85.0**	**$\sum$ 312**

Table 6.1: This table compares proof statistics from KeYmaera. The first row (llc from Chapter 4) gives proof statistics for the KeYmaera proof of safety for Model 1. The second row (time* ≤ event*) is a count of each proof step taken in the proof presented in Section 6.3. Goals 1-4 are proofs of the corresponding open goals left by the proof that time* refines event* from Section 6.3 for the specific implementations of $\texttt{llc}_{Ev}$ and $\texttt{llc}_t$. The last row (Total) simply sums the first four rows to get the proof statistic totals for $\texttt{llc}_t$. All models and proof files for $\texttt{llc}_t$ can be found online at http://www.cs.cmu.edu/~sloos/llct. Models and proof files for llc from Chapter 4 can be found online at http://www.ls.cs.cmu.edu/dccs/.

Chapter 7

Distributed Aircraft Control System

As airspace becomes ever more crowded, air traffic management must reduce both space and time between aircraft to increase throughput, making on-board collision avoidance systems ever more important. These safety-critical systems must be extremely reliable, and as such, many resources are invested into ensuring that the protocols they implement are accurate. Still, it is challenging to guarantee that such a controller works properly under every circumstance. In tough scenarios where a large number of aircraft must execute a collision avoidance maneuver, a human pilot under stress is not necessarily able to understand the complexity of the distributed system and may not take the right course, especially if actions must be taken quickly. We consider a class of distributed collision avoidance controllers designed to work even in environments with arbitrarily many aircraft or UAVs. We prove that the controllers never allow the aircraft to get too close to one another, even when new planes approach an in-progress avoidance maneuver that the new plane may not be aware of. Because these safety guarantees always hold, the aircraft are protected against unexpected emergent behavior which simulation and testing may miss. This is an important step in formally verified, flyable, and distributed air traffic control.

7.1 Introduction

Verification of air traffic control is particularly challenging because it lies in the intersection of many fields which already give tough verification problems when examined independently. It is a distributed system, with a large number of aircraft interacting over an unbounded time horizon. Each aircraft has nonlinear continuous dynamics combined with complex discrete controllers. And finally, every protocol must be flyable (i.e. not cause the aircraft to enter a stall, bank too sharply, or require it to turn on sharp corners).

In this chapter, we investigate the safety of collision-avoidance controllers for aircraft systems. We want to prove safety not just for a single aircraft or a pair of aircraft, but for all aircraft operating simultaneously in the sky. Because this system is composed of multiple independent computational agents that interact with the physical world, it is called a *distributed hybrid system*. It is this combination of continuous flight dynamics, discrete flight control decisions, and distributed communication that causes verification of aircraft control protocols to be extremely challenging.

Aircraft control systems are safety-critical, so they must be designed with a high assurance of correctness. When the costs of failure are high, system designers must be able to guarantee ahead of time that their systems work as intended. Many methods, such as testing and simulation, are used in combination to improve reliability. While testing and simulation may reveal software bugs and increase safety assurance, they are not able to prove safety guarantees over the continuous and infinite state-spaces characteristic of hybrid systems like flight control, where the aircraft move continuously through space and time. The complexity of curved flight dynamics has been difficult for many analysis techniques [48–55], which often resort to unflyable approximations of flight trajectories that require aircraft to turn on corners. However, the formal verification techniques described in this chapter are able to provide guarantees for flyable maneuvers over the entirety of this continuous state-space and therefore over all evolutions of all aircraft movement.

These strong guarantees are especially important in a distributed system with a large number of interacting participants. As in [48, 51, 54, 56, 57], many previous approaches to aircraft control have looked into a relatively small number of agents. But with thousands of aircraft flying through commercial airspace daily, this system is already far too complex for humans to predict every scenario by looking at interactions of only a few aircraft. And this challenge increases when we examine controllers for Unmanned Aerial Vehicles (UAVs), which are becoming increasingly autonomous and fly even closer together, with less direct supervision by humans. As a result, we must provide a good argument for why a controller will always take the right action, even in extremely crowded airspace.

In this chapter we specify and verify two control policies for planar aircraft avoidance maneuvers using automated theorem prover KeYmaeraD to produce a proof of safety for each of them. We design these policies such that all aircraft adhere to a simple and easy-to-implement separation principle: associated with each aircraft is a disc, within which the aircraft must remain. In this way, the problem reduces to proving that i) sufficient separation is maintained between pairs of discs, and ii) individual aircraft always remain inside their associated disc. We model 2D flight dynamics since they are the relevant dynamics for planar maneuvers, but investigating 3D maneuvers and dynamics may make interesting future work.

The complexities which arise from the curved flight trajectories of an arbitrary number of aircraft interacting in a distributed manner, along with the tight coupling of discrete control and continuous dynamics presently make KeYmaeraD the only verification tool capable of proving safety for this system. Our contributions are:

- We provide the *first formally* verified distributed system of aircraft with *curved flight* dynamics.
- Our controller requires only *flyable* aircraft trajectories with no corners or instantaneous changes of ground speed.
- We prove our controller is safe for an *arbitrarily large number of aircraft*. This guarantee is necessary for high-traffic applications such as crowded commercial airspace, unmanned aerial vehicle maneuvers, and robotic swarms.
- Other aircraft may enter an avoidance maneuver already *in progress* and safety for all aircraft is guaranteed still.
- We use *Arithmetic coding* to reduce proof complexity and branching.

- We prove that even when the interactions of many aircraft cause unexpected *emergent behaviors*, all resulting control choices are still safe.
- We present *hierarchical and compositional* techniques to reduce a very complex system into smaller, provable pieces.

Proof Calculus and Prover Just as with the distributed car control problem from Chapter 4, we use *quantified hybrid programs* (QHPs) to model distributed aircraft systems and *quantified differential dynamic logic* (Qd$\mathcal{L}$) to verify them. See Section 4.3 for details on QHPs and Qd$\mathcal{L}$.

KeYmearaD [89] is a theorem prover which mechanizes the use of the Qd$\mathcal{L}$ proof calculus. It has previously been used to verify a simple car control system [89]. KeYmaeraD further implements quantified differential invariants [90]. KeYmaeraD constructs proofs by following a user-created *tactic script*. When KeYmaeraD runs a tactic script, it applies proof rules to the sequent based on tactics specified in the script which will ultimately reduce the sequent to several problems in first-order real arithmetic. These simpler problems are then sent to a backend decision procedure in Mathematica.

7.2 Related Work

Many methods for ensuring correctness have been researched, each having different strengths in dealing with the various challenges posed by air traffic control. Pallottino et al. [60] proposed a distributed collision avoidance policy that is closely related to the systems we examine here. They provide a thorough empirical description of the system's behavior, emphasizing simulation and physical experiment. They formulate a liveness property and give probabilistic evidence for it using Monte Carlo methods. They also provide an informal proof of safety that is similar in high-level ideas to our proofs, but does not consider a model for flight dynamics. However, since we provide *formal* proofs of safety based directly on the control protocols and working with a continuous model of flight dynamics, we provide a higher degree of assurance and a clearer avenue to safely extend the systems.

Verification methods for systems with an arbitrary number of agents behaving under distributed control fall primarily into one of two categories: theorem proving and parameterized verification. Johnson and Mitra [55] use parameterized verification to guarantee that a distributed air traffic landing protocol (SATS) is collision free. Using backward reachability, they prove safety in the SATS landing maneuver given a bound on the number of aircraft that can be engaged in the landing maneuver. The protocol divides the airspace into regions and models the aircraft flight trajectory within each region by a clock. We consider the complementary problem of free flight instead of airport landing traffic, and we show that in free space, *arbitrarily* many aircraft can join our maneuver, and we model aircraft movement using *flyable*, curved flight dynamics.

Other provably safe systems with a specific (usually small) number of agents are presented in [48, 51, 54, 56]. The work by Umeno and Lynch [51, 54] is complementary to ours; however while they consider real-time properties of airport protocols using Timed I/O Automata, we prove local properties of the actual hybrid system flight dynamics. Duperret et al. [56] verify a roundabout maneuver with three vehicles. Each vehicle is constrained to a specific, pre-defined

path, so physical dynamics are simplified to one dimension. Tomlin et al. [48] analyze competitive aircraft maneuvers game theoretically using numerical approximations of partial differential equations. As a solution, they propose roundabout maneuvers and give bounded-time verification results for up to four aircraft using straight-line approximations of flight dynamics.

Flyability is identified as a major challenge in Košecká et al. [58], where planning based on superposition of potential fields is used to resolve air traffic conflicts. This planning does not guarantee flyability but, rather, defaults to classical vertical altitude changes whenever a nonflyable path is detected. The resulting maneuver has not yet been verified. The planning approach has been pursued by Bicchi and Pallottino [59] with numerical simulations.

Numerical simulation algorithms approximating discrete-time Markov Chain approximations of aircraft behavior have been proposed by Hu et al. [49]. They approximate bounded-time probabilistic reachable sets for one initial state. We consider hybrid systems combining discrete control choices and continuous dynamics instead of uncontrolled, probabilistic continuous dynamics. Hwang et al. [53] have presented a straight-line aircraft conflict avoidance maneuver involving optimization over complicated trigonometric computations, and validate it using random numerical simulation and informal arguments. The work of Dowek et al. [50] and Galdino et al. [52] shares many goals with ours. They consider unflyable, straight-line maneuvers and formalize geometrical proofs in PVS.

Our approach has a different focus from complementary work:

- Our maneuver directly involves curved flight unlike [48–55]. This makes our maneuver more realistic since it is *flyable*, but much more difficult to analyze.
- Unlike [49, 53, 58], we do not give results for a finite (sometimes small) number of initial flight positions (as in simulation). Instead, we verify uncountably many initial states and give unbounded-time horizon verification results.
- Unlike [48, 49, 53, 58, 59, 91], we use symbolic computation so that numerical and floating point errors can not violate soundness.
- Unlike [49–55, 59, 92], we analyze hybrid system dynamics directly, not approximations like clocks.
- Unlike [48, 49, 53, 58–60, 92] we produce *formal*, deductive proofs.
- In [50–54, 60], it is not proved that the hybrid dynamics and flight equations follow the geometrical thoughts. In contrast, our approach directly works for the hybrid flight dynamics.
- Unlike [48–50, 52, 53, 56–59], we verify the case of *arbitrarily many* aircraft, which is crucial for dense airspace.
- Unlike [59, 91], we do not guarantee optimality of the resulting maneuver.

7.3 Big Disc

In this chapter, we present two classes of aircraft controllers, each of which maintains a guaranteed minimum distance p between all aircraft. We then prove that both of these controller classes are safe, (i.e. that the minimum distance p between aircraft is never violated). Each aircraft has a

disc-shaped zone large enough to fly a circle within and which no other aircraft will be allowed to enter.

In the first controller class, presented in this section, each aircraft maintains a larger buffer disc with the aircraft at its center. This disc allows pilots some freedom during an avoidance maneuver, including the choice of circling direction. We imagine this controller will be useful when passenger comfort is a factor, as in commercial airlines. The second class of controllers (Section 7.4) uses smaller buffer discs centered to the left or right of the aircraft. The smaller discs allow the aircraft to fly closer together, but there may be little choice in how a maneuver is executed. This is well suited to UAVs which may fly very close together and are concerned only with flyability, not passenger comfort. Additionally, since many UAVs may be monitored and managed remotely by a small group of people, it may be more desirable to have a specific collision avoidance maneuver with little freedom and high predictability. Because the first controller class requires a larger disc than the second, we call it *Big Disc*, and appropriately we name the second class *Small Discs*.

We use two levels of abstraction to analyze the controllers. At the higher abstraction level we model the buffer discs, which can freeze instantaneously when they get within p distance of each other. At the lower level, we model the movement of aircraft within their discs, ensuring they always stay within the buffer zone while following flyable trajectories. In the proof, these two levels of abstraction are joined so that safety is assured for the system as a whole.

We model airspace as $\mathbb{R}^2$ and aircraft as points moving in this space. Each aircraft i steers by adjusting its angular velocity $\omega(i)$. When $\omega(i)$ is zero, the plane flies in a straight line. As angular velocity increases, the plane flies in a tighter and tighter circle, so we put an upper bound on the angular velocity $\Omega(i)$ based on the smallest circle that aircraft i can fly while maintaining constant linear speed $v(i)$. We keep linear speed $v(i)$ constant for each aircraft. We can determine the radius of each aircraft's smallest flyable circle by the equation $minr(i) = v(i)/\Omega(i)$. This model is known as the *Dubins vehicle* [93] and has been used previously for aircraft verification [48]. We allow an arbitrarily large number of aircraft to be present in airspace, so long as there is enough space to pack their discs. To our knowledge, no other method has been able to verify a protocol or controller safe for an arbitrary number of aircraft using a continuous model of their flight dynamics. This is not surprising, since safety must be guaranteed even for unpredictable emergent behaviors and in crowded, worst-case scenarios. Models written as QHPs inherently have a compositional and hierarchical structure which makes them easier to decompose into smaller, provable pieces by using sound proof rules. We also use nondeterminism in the model of the controller, which means that our proof is robust to variations in implementation on individual aircraft.

During normal free flight (i.e. whenever the aircraft is not engaged in a collision avoidance maneuver), the buffer zone for an aircraft i is a disc of radius $2minr(i)$ centered around the aircraft, which is at planar position $x(i) = (x_1(i), x_2(i))$. So long as the aircraft does not enter a collision avoidance maneuver, its buffer disc remains centered on the aircraft. However, should aircraft i come too close to another plane, it will enter collision avoidance mode and begin circling at radius $minr(i)$ to either the left or the right. The disc allows just enough room for this maneuver; however, the disc is big in the sense that it allows a considerable amount of freedom once the aircraft has gone halfway around this initial circle. The beginning of one possible trajectory of a collision avoidance maneuver is illustrated in Figure 7.1. The current direction of flight of

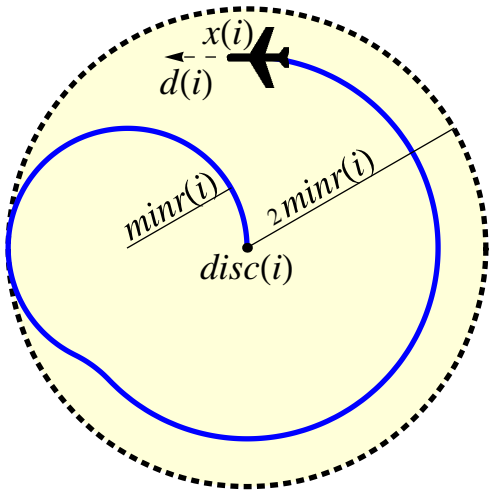

Figure 7.1: A possible collision avoidance trajectory of `BigDisc`.

aircraft i is given by the indexed variable $d(i)$ as a 2D unit vector. The variable $disc(i)$ stores the position of the center of i's buffer disc. The aircraft need not always turn at the maximum angular velocity $\Omega(i)$; we require only that the aircraft remain within the disc by circling in its original direction. (Note: while it is possible for the aircraft to change its circling direction while staying within the disc by flying a figure eight or an 'S' shape, we disallow this behavior since it would increase the complexity of the controller.)

Formal Model

The Big Disc policy is presented formally in Model 10 and we describe it in this section. The variable $ca(i)$, indicates whether aircraft i is in a collision avoidance maneuver. If $ca(i) = 0$ then i is in free flight; if $ca(i) = 1$ then i is in an avoidance maneuver and is circling within its disc. Each aircraft has the ability to enter collision avoidance independently and asynchronously. This simplifies collision avoidance maneuvers with more than two aircraft and improves reliability, since no perfect synchronization is required, which would be difficult to implement in a distributed system.

The variable $side(i)$ indicates i's circling direction when it enters an avoidance maneuver. If $side(i) = 1$, i will circle counter-clockwise; if $side(i) = -1$, i circles clockwise. We use $\|y\|$ to denote the Euclidean norm and we use $y^\perp$ to denote the vector obtained by rotating y ninety degrees counter-clockwise, $y^\perp := (-y_2, y_1)$.

The quantified hybrid program `BigDisc` is a loop, which is represented in (7.1) by *, the nondeterministic repetition operator. Each iteration is either a control action as represented by `Control` or an evolution of physics as represented by `Plant`. This loop may repeat arbitrarily many times. In the `Control` branch, the program nondeterministically selects an aircraft k ($k := *_{\mathbb{A}}$ assigns an arbitrary aircraft into k) and then allows k to perform some action. The allowed actions depend on whether k is in a collision avoidance maneuver. If it is (case `CA`), then k may either adjust its angular velocity in the `Steer` branch, or exit the maneuver with the `Exit` branch. The new angular velocity in the `Steer` branch is arbitrary ($\omega(k) := *_{\mathbb{R}}$, where $*_{\mathbb{R}}$ is an arbitrary real number) but bounded by $-\Omega(k)$ and $\Omega(k)$ due to the subsequent test. The

aircraft may only `Exit` the collision avoidance circling maneuver when $x(k) = disc(k)$, i.e., the aircraft must return to the center of the disc before exiting the maneuver. If k is not in a collision avoidance maneuver (case `NotCA`), then it may once again `Steer`, or it may switch its circling direction with the `Flip` branch, or it may enter collision avoidance with the `Enter` branch. In the `Enter` branch, the aircraft sets its angular velocity so that it will circle with radius $minr(k)$, thereby entering a collision avoidance. It also sets the $ca(k)$ flag to indicate internally that it has entered this maneuver.

Model 10 Big Disc

$$\texttt{BigDisc} \equiv (\texttt{Control} \cup \texttt{Plant})^* \tag{7.1}$$
$$\texttt{Control} \equiv k := *_{\mathbb{A}};\ (\texttt{CA} \cup \texttt{NotCA}) \tag{7.2}$$
$$\texttt{CA} \equiv ?(ca(k) = 1);\ (\texttt{Steer} \cup \texttt{Exit}) \tag{7.3}$$
$$\texttt{NotCA} \equiv ?(ca(k) = 0);\ (\texttt{Steer} \cup \texttt{Flip} \cup \texttt{Enter}) \tag{7.4}$$
$$\texttt{Steer} \equiv \omega(k) := *_{\mathbb{R}};\ ?(-\Omega(k) \leq \omega(k) \leq \Omega(k)) \tag{7.5}$$
$$\texttt{Exit} \equiv ?(disc(k) = x(k));\ ca(k) := 0 \tag{7.6}$$
$$\texttt{Enter} \equiv \omega(k) := side(k) \cdot \Omega(k);\ ca(k) := 1 \tag{7.7}$$
$$\texttt{Flip} \equiv side(k) := -side(k) \tag{7.8}$$
$$\texttt{Plant} \equiv \forall i : \mathbb{A} \Big(x(i)' = v(i) \cdot d(i),\ d(i)' = \omega(i) \cdot d(i)^{\perp}, \tag{7.9}$$
$$disc(i)' = (1 - ca(i)) \cdot v(i) \cdot d(i)\ \&\ \texttt{EvDom}\Big) \tag{7.10}$$
$$\texttt{EvDom} \equiv \forall j : \mathbb{A} \tag{7.11}$$
$$((j \neq i \wedge (ca(i) = 0 \vee ca(j) = 0)) \rightarrow \mathsf{Sep}(i, j) \tag{7.12}$$
$$\wedge \|disc(i) - (x(i) + minr(i) \cdot side(i) \cdot d(i)^{\perp})\|$$
$$\leq minr(i)) \tag{7.13}$$
$$\mathsf{Sep}(i, j) \equiv \|disc(i) - disc(j)\| \geq 2minr(i) + 2minr(j) + p \tag{7.14}$$

The other branch in `BigDisc`'s main loop is `Plant`. The position of the aircraft, $x(i)$, changes according to its direction, $d(i)$, which in turn changes according to its angular velocity, $\omega(i)$. This makes $\omega(i)$ our primary control variable. These physical dynamics are modeled by the differential equation in (7.9). The center of the disc, $disc(i)$, is stationary during a collision avoidance maneuver, but otherwise it is equal to the aircraft position. This case distinction is achieved by using arithmetic coding, whereby we multiply by $(1 - ca(i))$, in (7.10). This reduces a branching of the system which would be incurred if we were to use traditional if-else coding style, and thereby reduces the complexity of the safety proof. When the aircraft is in free flight, $ca(i) = 0$, which causes $disc(i)'$ to equal $x(i)'$. But, when the aircraft is in a collision avoidance maneuver, $ca(i) = 1$, causing $disc(i)' = 0$, so the disc is stationary. The evolution domain, `EvDom`, has two purposes. First, it monitors the disc positions of other aircraft ((7.12)). Recall that in order for the system to be considered safe, no aircraft can pass closer than distance p to another. So, if the aircraft's disc comes within p of another disc (as quantified in (7.14)), it forces both

aircraft to enter collision avoidance. Second, while the aircraft has a great amount of freedom in how it maneuvers during collision avoidance, it must always be able to flyably remain within its buffer disc. We use the inequality in (7.13) to quantify this condition. It states that if the aircraft turns in a tight circle with radius $minr(i)$, the origin of that tight circle is no more than $minr(i)$ away from the point $disc(i)$.

Our model allows for a huge amount of nondeterminism, both in the discrete dynamics of the controller (e.g. which aircraft are controlled ($k := *_{\mathbb{A}}$, (7.2)), how the aircraft steer ($\omega(k) := *$, (7.5)), and whether to enter a collision avoidance maneuver), and in the continuous dynamics of the plant (e.g. how long to wait between control choices). This nondeterminism is a beneficial property of our collision avoidance protocol, since it allows for each aircraft to implement slightly different control algorithms without violating the proof of safety for the entire system. As a result, we have verified a class of controllers, rather than one specific implementation.

Theorem Statement

In order to guarantee safety, we must prove that for all pairs of distinct aircraft i, j, the distance between i and j is greater than or equal to p. We can express this condition formally as

$$\mathsf{Safe} \equiv \forall\, i, j : \mathbb{A}\ \ (i \neq j \rightarrow \|x(i) - x(j)\| \geq p).$$

We must show that Safe holds during all executions of `BigDisc`. The Qd$\mathcal{L}$ formula expressing this property is [`BigDisc`]Safe. We must also ensure that the aircraft begin in a controllable state. This means each aircraft must have a buffer disc (`InitA`), which is empty (`InitB`), and within which it may flyably maneuver (`InitC`). For aircraft i that are not in an avoidance maneuver, we ensure that i's disc is at the same point as i. Since $ca(i) = 1$ for aircraft in a maneuver and $ca(i) = 0$ otherwise, we may write this property as

$$\texttt{InitA} \equiv \forall i : \mathbb{A}\ (1 - ca(i)) \cdot disc(i) = (1 - ca(i)) \cdot x(i).$$

By again using this arithmetic coding style rather than an if-else statement, we eliminate a significant branching factor in the resulting proof. We then show that the discs are empty by ensuring sufficient separation between the discs as defined in Model 10 (7.14).

$$\texttt{InitB} \equiv \forall i, j : \mathbb{A}\ \ (i \neq j \rightarrow \mathsf{Sep}(i, j))$$

Finally, we ensure that aircraft in collision avoidance maneuvers are able to flyably remain within their discs. We do this by proving that the following formula is an invariant of our system. It states that if i tightly circles in its current circling direction, the center of this tight circle will be within distance $minr(i)$ of $disc(i)$.

$$\texttt{InitC} \equiv \forall i : \mathbb{A}\ \ \|disc(i) - (x(i) + minr(i) \cdot side(i) \cdot d(i)^{\perp})\| \leq minr(i)$$

Note that these initial conditions all hold trivially if the aircraft begin far enough apart that none is in a collision avoidance maneuver.

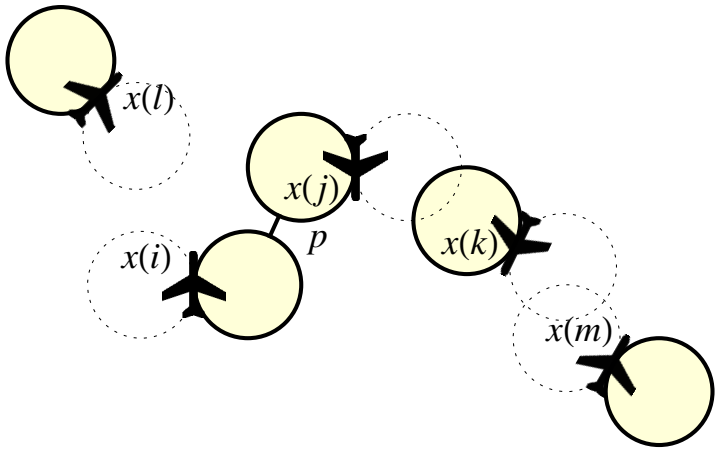

Figure 7.2: One possible scenario in the Small Discs policy

Theorem 2 (Safety of `BigDisc`)**.** *If the aircraft are initially in a controllable state, then no two aircraft will come closer than distance p while all aircraft follow* `Control`*; therefore safety of the* `BigDisc` *controller is expressed by the provable* QdℒL *formula:*

$$(\texttt{InitA} \wedge \texttt{InitB} \wedge \texttt{InitC}) \rightarrow [\texttt{BigDisc}]\mathsf{Safe}$$

We proved Theorem 2 for all parameter values by showing that `InitA`, `B` and `C` are maintained as invariants. This proof was generated using KeYmaeraD from a 330 line user-generated tactic script. The tactic file and the KeYmaeraD theorem prover are available online [6]. A discussion of the critical techniques needed to complete this proof is presented in [94, Appendix A.1].

7.4 Small Discs

One drawback of the Big Disc policy is that it may trigger collision avoidance maneuvers that are not strictly necessary; the buffer zones are larger than the required circling space of the aircraft. Our second policy, *Small Discs*, aims to decrease the size of the disc. The only way to do this is to abandon the assumption that the disc must be centered on the aircraft during free flight. Instead, the buffer zone, a disc of radius $minr(i)$, is centered at a point with distance $minr(i)$ away from $x(i)$, in a direction perpendicular to i's motion either to the left or the right. Thus the aircraft is always on the edge of its disc, and during collision avoidance, the aircraft follows the circumference of its disc. As with `BigDisc`, an aircraft may flip its circling direction during free flight. This now makes the disc jump to the other side of the aircraft, and before an aircraft can flip its circling direction, it must check that it may do so safely.

Fig. 7.2 illustrates a situation where flipping the disc to the opposite side prevents an unnecessary collision avoidance maneuver. Each aircraft has an *active disc* (solid discs in Fig. 7.2) that it will use for collision avoidance if needed. We also illustrate the *inactive disc* (dotted discs) as an alternative choice for the disc if the aircraft decides to flip its circling direction. In Fig. 7.2, the active discs of aircraft i and aircraft j are on a collision course. If nothing is done, at the

latest when the edges of the discs are separated by distance p, both aircraft will enter collision avoidance by circling to the right around the circumference of their respective discs. Notice that in this case, the aircraft may pass as close as the minimum separation distance p when aircraft i is at the top of its disc while aircraft j is at the bottom. However, since aircraft i has free space to its left, it may flip its circling disc to the opposite side and no collision avoidance maneuvers are necessary. Aircraft j is unable to make such a flip, since aircraft k's disc occupies the necessary space. Only collisions of active discs are a problem. The fact that the inactive discs of k and m overlap is immaterial, because if collision avoidance is necessary, every aircraft will follow the circumference of its *active* disc. This also illustrates that aircraft must synchronize disc flipping. If, to enable j to flip its disc, k flips its disc, but, at the same time, and unaware of this, m flips its disc, then k and m would have incompatible collision avoidance discs. Since purely discrete standard solutions exist for ensuring consistency in such discrete mode changes, our model simply uses sequentialized flipping decisions.

Formal Model

The Small Discs policy is presented formally as `SmallDiscs` in Model 11. The overall structure is similar to that of `BigDisc`. One notable difference is that `SmallDiscs` no longer uses the state variable $disc(i)$. During free flight, the center of an aircraft's disc moves with dynamics that are not easy to express in terms of other variables; it is certainly not as easy as setting $disc(i)' = x(i)'$, as we did in `BigDisc`. The point $disc(i)$ moves faster or slower than $x(i)$, depending on whether the aircraft is veering away from its active disc, or towards it. As a result, $disc(i)$ has very involved continuous dynamics. Fortunately, however, the position of aircraft i's disc can be simply expressed in terms of other state variables. By using differential-algebraic equations as in [62], we equate

$$disc(i) = x(i) + minr(i) \cdot side(i) \cdot d(i)^{\perp}. \tag{7.15}$$

In order to simplify the mathematics of the system, we directly reduce the system to ordinary differential equations, which also makes the connection to `BigDisc` more apparent. Thus, instead of using (7.15) as part of a differential-algebraic equation [62], we consider (7.15) as a definition and statically replace all occurrences of $disc(i)$ by the right-hand side of (7.15). The subsequent model should be read with this in mind. The other major change in Model 11 is the separation condition Sep and the newly introduced separation condition $\mathsf{FlipSep}$ for flipping the disc.

Theorem Statement

Here again we want to prove that under safe initial conditions, the `SmallDiscs` controller is always Safe, where Safe is exactly as we defined it for `BigDisc`. We need to modify the initial conditions for `SmallDiscs`. There are two properties that we want to hold: first, the discs are separated (`InitD`), and second, when an aircraft is engaged in collision avoidance it is flying along the circumference of its active disc (`InitE`). `InitD` is similar to `InitB` for `BigDisc`, but it uses our new definition of Sep for `SmallDiscs`:

$$\texttt{InitD} \equiv \forall i, j : \mathbb{A} \ \ (i \neq j \rightarrow \mathsf{Sep}(i, j)).$$

$$\texttt{SmallDiscs} \equiv (\texttt{Control} \cup \texttt{Plant})^* \tag{7.16}$$
$$\texttt{Control} \equiv k := *_{\mathbb{A}};\ (\texttt{CA} \cup \texttt{NotCA}) \tag{7.17}$$
$$\texttt{CA} \equiv ?(ca(k) = 1);\ (\texttt{Exit} \cup \texttt{Skip}) \tag{7.18}$$
$$\texttt{NotCA} \equiv ?(ca(k) = 0);\ (\texttt{Steer} \cup \texttt{Flip} \cup \texttt{Enter}) \tag{7.19}$$
$$\texttt{Skip} \equiv ?true \tag{7.20}$$
$$\texttt{Steer} \equiv \omega(k) := *_{\mathbb{R}};\ ?(-\Omega(k) \leq \omega(k) \leq \Omega(k)) \tag{7.21}$$
$$\texttt{Exit} \equiv ca(k) := 0 \tag{7.22}$$
$$\texttt{Enter} \equiv (\omega(k) := side(k) \cdot \Omega(k));\ ca(k) := 1 \tag{7.23}$$
$$\texttt{Flip} \equiv ?(\forall j : \mathbb{A}\ (j \neq k \rightarrow \ \mathsf{FlipSep}(j,k)));\ side(k) := -side(k) \tag{7.24, 7.25}$$
$$\mathsf{FlipSep}(i,j) \equiv \|(x(i) + minr(i) \cdot side(i) \cdot d(i)^{\perp}) - (x(j) - minr(j) \cdot side(j) \cdot d(j)^{\perp})\| \geq minr(i) + minr(j) + p \tag{7.26–7.28}$$
$$\texttt{Plant} \equiv \forall i : \mathbb{A}\ \Big(x(i)' = v(i) \cdot d(i),\ d(i)' = \omega(i) d(i)^{\perp} \ \&\ \forall j : \mathbb{A}\ ((j \neq i \wedge (ca(i) = 0 \vee ca(j) = 0)) \rightarrow \mathsf{Sep}(i,j))\Big) \tag{7.29–7.31}$$
$$\mathsf{Sep}(i,j) \equiv \|(x(i) + minr(i) \cdot side(i) \cdot d(i)^{\perp}) - (x(j) + minr(j) \cdot side(j) \cdot d(j)^{\perp})\| \geq minr(i) + minr(j) + p \tag{7.32–7.34}$$

If i is in a collision avoidance maneuver, then i is turning at maximal angular velocity. This implication is expressed with arithmetic coding by multiplying both sides with the indicator $ca(i)$:

$$\texttt{InitE} \equiv \forall i : \mathbb{A}\ \ (\omega(i) \cdot ca(i) = \Omega(i) \cdot side(i) \cdot ca(i)).$$

Theorem 3 (Safety of `SmallDiscs`)**.** *If the aircraft are initially in a controllable state (i.e. where* `InitD` *and* `InitE` *hold), then no aircraft will come closer than distance p to any other aircraft so long as each aircraft follows* `Control`*; therefore safety of the* `SmallDiscs` *controller is expressed by the provable* Qd$\mathcal{L}$ *formula:*

$$(\texttt{InitD} \wedge \texttt{InitE}) \rightarrow [\texttt{SmallDiscs}]\mathsf{SafeSafe}$$

We proved Theorem 3 in KeYmaeraD by showing `InitD` and `InitE` are invariant. The accompanying tactic script is 309 lines in length and is available online [6].

Chapter 8

Applications of dR$\mathcal{L}$

8.1 MPC Design and Verification

A challenge we have seen using d$\mathcal{L}$ is when we have tried to verify a system that is very far removed in its design from the safety properties that we are trying to check. One such example, mentioned in Section 1.4, is the adaptive cruise controller designed to optimize fuel efficiency. The controller that optimizes fuel consumption (which would be included in the definition of function f) would obfuscate the properties of function f that ensure safe following.

$$[a_f := f(x_f, v_f, x_l, v_l); a_l := *; x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l](x_f < x_l) \tag{8.1}$$

On the other extreme, we could write a hybrid program with a controller that is nearly vacuously true. One that is, by perfect, implicit design, easy to verify satisfies the safety conditions.

$$\begin{aligned}&\Big[a_f := *;\\ &?([a_l := *; x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l](x_f < x_l));\\ &a_l := *;\\ &x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l\Big](x_f < x_l)\end{aligned} \tag{8.2}$$

This implicit controller is perfect by design; we call it MPC, since it is reminiscent of a model predictive controller. MPC properties like (8.2) are easy to verify even without refinement due to their somewhat circular construction. However, implicit constructions are not exactly useful when it comes time to implement the model.

By using the refinement relation in dR$\mathcal{L}$, it may be possible to relate this easy-to-verify MPC property to our original property (8.1). If we can prove the following refinement, then property (8.1) must also hold:

$$(a_f := f(x_f, v_f, x_l, v_l)) \leq (a_f := *; ?([a_l := *; x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l](x_f < x_l))) \tag{8.3}$$

Previously, when proving properties about hybrid programs with implicit controllers in d$\mathcal{L}$, we had to demonstrate as a side condition that there is some value which can in fact satisfy the

implicit design, otherwise the proof would actually be vacuous. Now, the refinement relation in (8.3) cannot hold in the case where the implicit hybrid program satisfies the safety property vacuously, so we have a guarantee that the side condition was correctly and carefully handled.

8.2 Safety Envelope Verification

Related to MPC, but perhaps not taken to quite an extreme is the concept of verifying safety envelopes within which the controller is guaranteed to be safe. This brings us a step closer to something that could be implementable as it doesn't include box modality properties in the controller, but is still easier to prove than the explicit property. This would allow us to put an intermediate step between the explicit controller, which is easy-to-implement, and the MPC controller, which is easy to verify.

Consider the following car control example. In (8.4), we have a specific controller, which would be extremely challenging to verify directly, particularly if $f(x_f, v_f, x_l, v_l)$ is not closely related to safety. At the other extreme, in (8.6) we have an MPC version of the car control example. This property is likely to be extremely easy to verify, since the safety property is essentially baked into the controller, as discussed in Section 8.1.

However, proving refinement directly between these two models may be challenging, as they are structurally and conceptually very different. If we can automatically generate a hybrid program that sits between these two, we may still have a hope of establishing the refinement relation. The intermediate step could be in the form of a control envelope, shown in (8.5).

$$(a_f := f(x_f, v_f, x_l, v_l)) \tag{8.4}$$

$$\leq a_f := *; ?(x_f + \frac{v_f^2}{-2a_f} < x_l \wedge a_f < 0) \tag{8.5}$$

$$\leq (a_f := *; ?([a_l := *; x_f' = v_f, v_f' = a_f, x_l' = v_l, v_l' = a_l](x_f < x_l))) \tag{8.6}$$

8.3 Refinement and Hierarchical Proofs

While exploring the use of refinement in the distributed car control and aircraft control case studies may reveal some specific examples where hierarchical proof structures can be retroactively injected using a refinement relation, ideally the structures would automatically arise during the proof search. Exploring new heuristics that leverage the refinement relation in a way that enforces hierarchical proof structures may lead to more efficient proof search algorithms. While designing and testing such proof search heuristics are likely out of scope for this thesis, they represent an interesting and possibly very fruitful extension of this research.

8.4 Proof Structure of Distributed Aircraft Control

The primary insight in Sections 7.3 and 7.4 that puts verification of the collision avoidance protocols in scope for QdL and KeYmaeraD is the decomposition of disc requirements and aircraft

requirements in the first step of the proof. In the left branch of the proof, we show that the small discs protocol ensures that each aircraft always remains inside its associated disc. The right branch proves that the small discs protocol ensures all discs stay safely separated.

However, this decomposition is limited to breaking apart the safe separation requirement. Could we extend this same concept to break apart the model itself into simpler models that would be easier to verify? Fig. 8.1 gives a pictorial representation of what such a proof might look like.

This style of proving is entirely out of scope for QdL, which has no formal mechanism for relating hybrid programs. In this thesis, we introduce a refinement relation over hybrid programs in dRL, which is a step in the right direction. However, in order to achieve the transformation on models indicated in green in Fig. 8.1, variables for continuous dynamics of aircraft and the distributed dynamics of discs would need to be soundly removed from the hybrid program. We expect this to be possible, since these variables no longer exist in the safety properties that are being verified in the two (independent) branches of the proof. However, in dRL, the refinement relation is completely oblivious to any safety or liveness properties. We can only hope to prove the Small Discs protocol in this style after adding a notion of refinement that can project onto a set of variables.

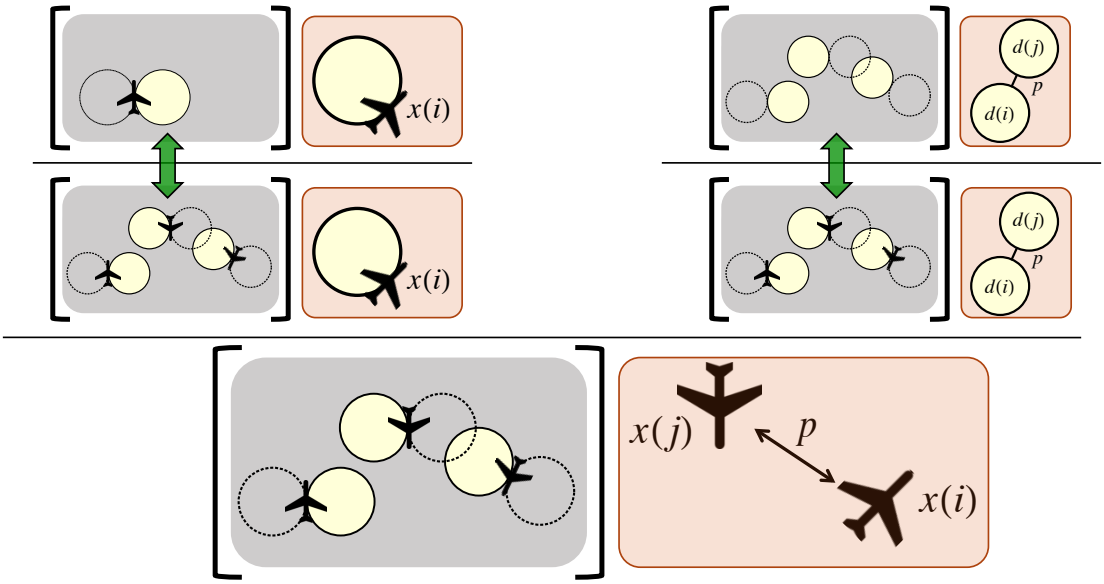

Figure 8.1: Using refinement, we may be able to reduce the complexity of the branches for the proofs presented in Section 7.4.

8.5 From Verification to Synthesis

Perhaps the most desirable way to close the gap between a verified model of a system and its implementation would be to automatically synthesize the implementation. This is only useful if each step in the synthesis process maintains the verification guarantees and the result of the

synthesis adheres to the expectations and requirements of system designers. This challenge is amplified when the original model has many nondeterministic components.

Chapter 9

Conclusion

This thesis presents differential refinement logic (dR$\mathcal{L}$), a specification and verification logic that allows the direct comparison of hybrid programs. We present a proof calculus for dR$\mathcal{L}$ and prove it sound. The rules in the proof calculus can be partitioned into three primary types: 1) structural proof rules, which leverage structural similarities between hybrid programs, 2) proof rules for handling the differential equations, and 3) rules based on the axioms of Kleene algebra with tests.

In this thesis we consider three cutting-edge case studies in the automotive and aerospace domains. Each case study has been formally verified using differential dynamic logic (d$\mathcal{L}$) and its associated proof calculus. However, these d$\mathcal{L}$ proofs often required ad hoc and tricky proof techniques. With dR$\mathcal{L}$, we formalize and simplify many of the tricks required to verify challenging hybrid programs. We then examine how both the models and verification process can be improved using *refinement* in dR$\mathcal{L}$.

While this thesis introduces the foundations of the dR$\mathcal{L}$ logic and provides solid evidence that dR$\mathcal{L}$ can simplify many challenges of verification for hybrid systems, there are still many questions to be explored, and in particular with a view to automation: How can an automated or semi-automated proof search take advantage of the added proof structure that dR$\mathcal{L}$ provides? Can refinement aid automatic synthesis for verified implementation of hybrid programs? These and many other questions remain open goals for future work.

In conclusion, dR$\mathcal{L}$ can improve the feasibility of theorem proving for hybrid systems by making it easier to break systems into smaller subsystems, abstract implementation-specific design details, leverage an iterative approach to system design, and maintain a modular proof structure.

Bibliography

[1] Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In Olivetti, N., ed.: TABLEAUX. Volume 4548 of LNCS., Springer (2007) 216–232 1.1, 3.3

[2] Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems. In Armando, A., Baumgartner, P., Dowek, G., eds.: IJCAR. Volume 5195 of LNCS., Springer (2008) 171–178 5.3

[3] Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010) 2, 3.1, 3.1, 3.2

[4] Platzer, A.: Logics of dynamical systems. In: LICS, IEEE (2012) 13–24 1.1, 2, 3.1, 3.2, 3.2

[5] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In Butler, M., Schulte, W., eds.: FM. Volume 6664 of LNCS., Springer (2011) 42–56 1.1, 1.1, 5.2, 5.3

[6] Loos, S.M., Renshaw, D., Platzer, A.: Formal verification of distributed aircraft controllers. In: Proceedings of the 16th international conference on Hybrid systems: computation and control. HSCC '13, New York, NY, USA, ACM (2013) 125–130 Electronic proofs can be downloaded online: `www.ls.cs.cmu.edu/discworld`. 1.1, 1.2, 1.1, 2.5, 7.3, 7.4

[7] Loos, S., Platzer, A.: Safe intersections: At the crossing of hybrid systems and verification. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). (2011) 1181–1186 1.1

[8] Rajhans, A., Bhave, A., Loos, S., Krogh, B., Platzer, A., Garlan, D.: Using parameters in architectural views to support heterogeneous design and verification. In: 2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC). (2011) 2705–2710 1.1, 1.4.3

[9] Arechiga, N., Loos, S., Platzer, A., Krogh, B.: Using theorem provers to guarantee closed-loop system properties. In: American Control Conference (ACC), 2012. (2012) 3573–3580 1.1, 1.1, 1.4.3

[10] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In Butler, M., Schulte, W., eds.: FM 2011: Formal Methods, 17th International Symposium on Formal Methods, Limerick, Ireland. Volume 6664 of LNCS., Springer (2011) 42–56 1.4.1, 2.4

[11] Loos, S.M., Witmer, D., Steenkiste, P., Platzer, A.: Efficiency analysis of formally verified

adaptive cruise controllers. In Hegyi, A., Schutter, B.D., eds.: Intelligent Transportation Systems (ITSC), 16th International IEEE Conference on, October 6-9, The Hague, Netherlands, Proceedings. (2013) 1565–1570 1.4.4

[12] Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. In Bonakdarpour, B., Smolka, S.A., eds.: Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Volume 8734 of LNCS., Springer (2014) 199–214 2.1, 3.3

[13] Mitsch, S., Quesel, J.D., Platzer, A.: Refactoring, refinement, and reasoning: A logical characterization for hybrid systems. In Jones, C.B., Pihlajasaari, P., Sun, J., eds.: FM. (2014) 2.1

[14] Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems (TOPLAS) **19** (1997) 427–443 2.2, 3.1, 3.4

[15] Kozen, D.: NetKAT – A formal system for the verification of networks. In: Programming Languages and Systems. Springer (2014) 1–18 2.2

[16] Foster, N., Kozen, D., Milano, M., Silva, A., Thompson, L.: A coalgebraic decision procedure for NetKAT. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM (2015) 343–355 2.2

[17] Spivey, J.M., Abrial, J.: The Z notation. Prentice Hall Hemel Hempstead (1992) 2.2

[18] Woodcock, J., Davies, J.: Using Z: specification, refinement, and proof. Prentice-Hall, Inc. (1996) 2.2

[19] Cavalcanti, A., Woodcock, J.: Zrc–a refinement calculus for z. Formal Aspects of Computing **10** (1999) 267–289 2.2

[20] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. Journal of the ACM (JACM) **50** (2003) 752–794 2.3

[21] Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010) 2.3

[22] Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in event-b. International journal on software tools for technology transfer **12** (2010) 447–466 2.3

[23] Hoare, C.A.R.: Communicating sequential processes. Volume 178. Prentice-hall Englewood Cliffs (1985) 2.3

[24] Banach, R., Zhu, H., Su, W., Huang, R.: Continuous kaos, asm, and formal control system design across the continuous/discrete modeling interface: a simple train stopping application. Formal Aspects of Computing **26** (2014) 319–366 2.3

[25] Butler, M., Abrial, J.R., Banach, R.: Modelling and refining hybrid systems in event-b and rodin. (2015) 2.3

[26] Girard, A., Julius, A.A., Pappas, G.J.: Approximate simulation relations for hybrid systems. Discrete Event Dynamic Systems **18** (2008) 163–179 2.3

[27] Girard, A., Pappas, G.J.: Approximate bisimulation: A bridge between computer science and control theory. European Journal of Control **17** (2011) 568–578 2.3

[28] Chang, J., Cohen, D., Blincoe, L., Subramanian, R., Lombardo, L.: CICAS-V research on comprehensive costs of intersection crashes. Technical Report 07-0016, NHTSA (2007) 2.4, 4.1

[29] Damm, W., Hungar, H., Olderog, E.R.: Verification of cooperating traffic agents. International Journal of Control **79** (2006) 395–421 4.2

[30] Dao, T.S., Clark, C.M., Huissoon, J.P.: Distributed platoon assignment and lane selection for traffic flow optimization. In: IEEE IV'08. (2008) 739–744 2.4, 4.2

[31] Dao, T.S., Clark, C.M., Huissoon, J.P.: Optimized lane assignment using inter-vehicle communication. In: IEEE IV'07. (2007) 1217–1222 2.4, 4.2

[32] Hall, R., Chin, C., Gadgil, N.: The automated highway system / street interface: Final report. PATH Research Report UCB-ITS-PRR-2003-06, UC Berkeley (2003) 2.4, 4.2

[33] Hall, R., Chin, C.: Vehicle sorting for platoon formation: Impacts on highway entry and troughput. PATH Research Report UCB-ITS-PRR-2002-07, UC Berkeley (2002) 2.4, 4.2

[34] Hsu, A., Eskafi, F., Sachs, S., Varaiya, P.: Design of platoon maneuver protocols for IVHS. PATH Research Report UCB-ITS-PRR-91-6, UC Berkeley (1991) 2.4, 4.2

[35] Ioannou, P.A.: Automated Highway Systems. Springer (1997) 2.4, 4.2

[36] Jula, H., Kosmatopoulos, E.B., Ioannou, P.A.: Collision avoidance analysis for lane changing and merging. PATH Research Report UCB-ITS-PRR-99-13, UC Berkeley (1999) 2.4, 4.2

[37] Horowitz, R., Tan, C.W., Sun, X.: An efficient lane change maneuver for platoons of vehicles in an automated highway system. PATH Research Report UCB-ITS-PRR-2004-16, UC Berkeley (2004) 2.4, 4.2

[38] Shladover, S.E.: Effects of traffic density on communication requirements for Cooperative Intersection Collision Avoidance Systems (CICAS). PATH Working Paper UCB-ITS-PWP-2005-1, UC Berkeley (2004) 2.4, 4.2

[39] Stursberg, O., Fehnker, A., Han, Z., Krogh, B.H.: Verification of a cruise control system using counterexample-guided search. Control Engineering Practice **38** (2004) 1269–1278 2.4, 4.2

[40] Varaiya, P.: Smart cars on smart roads: problems of control. IEEE Trans. Automat. Control **38** (1993) 195–207 2.4, 4.2

[41] Wongpiromsarn, T., Mitra, S., Murray, R.M., Lamperski, A.G.: Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle. In Majumdar, R., Tabuada, P., eds.: HSCC. Volume 5469 of LNCS., Springer (2009) 396–410 4.2

[42] Chee, W., Tomizuka, M.: Vehicle lane change maneuver in automated highway systems. PATH Research Report UCB-ITS-PRR-94-22, UC Berkeley (1994) 2.4, 4.2

[43] Johansson, R., Rantzer, A., eds.: Nonlinear and Hybrid Systems in Automotive Control. Society of Automotive Engineers Inc. (2003)

[44] Althoff, M., Althoff, D., Wollherr, D., Buss, M.: Safety verification of autonomous vehicles for coordinated evasive maneuvers. In: IEEE IV'10. (2010) 1078 – 1083 2.4, 4.2

[45] Berardi, L., Santis, E., Benedetto, M., Pola, G.: Approximations of maximal controlled safe sets for hybrid systems. In Johansson, R., Rantzer, A., eds.: Nonlinear and Hybrid Systems in Automotive Control, Springer (2003) 335–350 2.4, 4.1

[46] Alur, R.: Formal verification of hybrid systems. In: 2011 Proceedings of the International Conference on Embedded Software (EMSOFT). (2011) 273–278 2.4

[47] Clark, M., Koutsoukos, X., Kumar, R., Lee, I., Pappas, G., Pike, L., Porter, J., Sokolsky, O.: A study on run time assurance for complex cyber physical systems. (2013) 2.4

[48] Tomlin, C., Pappas, G.J., Sastry, S.: Conflict resolution for air traffic management. IEEE T. Automat. Contr. **43** (1998) 509–521 2.5, 7.1, 7.2, 7.3

[49] Hu, J., Prandini, M., Sastry, S.: Probabilistic safety analysis in three-dimensional aircraft flight. In: CDC. (2003) 2.5, 7.2

[50] Dowek, G., Muñoz, C., Carreño, V.A.: Provably safe coordinated strategy for distributed conflict resolution. In: AIAA-2005-6047. (2005) 2.5, 7.2

[51] Umeno, S., Lynch, N.A.: Proving safety properties of an aircraft landing protocol using I/O automata and the PVS theorem prover. In Misra, J., Nipkow, T., Sekerinski, E., eds.: FM. Volume 4085 of LNCS., Springer (2006) 64–80 2.5, 7.1, 7.2

[52] Galdino, A.L., Muñoz, C., Ayala-Rincón, M.: Formal verification of an optimal air traffic conflict resolution and recovery algorithm. In Leivant, D., de Queiroz, R., eds.: WoLLIC. Volume 4576 of LNCS., Springer (2007) 177–188 2.5, 7.2

[53] Hwang, I., Kim, J., Tomlin, C.: Protocol-based conflict resolution for air traffic control. Air Traffic Control Quarterly **15** (2007) 1–34 2.5, 7.2

[54] Umeno, S., Lynch, N.A.: Safety verification of an aircraft landing protocol: A refinement approach. In Bemporad, A., Bicchi, A., Buttazzo, G., eds.: HSCC. Volume 4416 of LNCS., Springer (2007) 557–572 2.5, 7.1, 7.2

[55] Johnson, T., Mitra, S.: Parameterized verification of distributed cyber-physical systems:an aircraft landing protocol case study. In: ACM/IEEE ICCPS. (2012) 2.5, 7.1, 7.2

[56] Duperret, J.M., Hafner, M.R., Del Vecchio, D.: Formal design of a provably safe roundabout system. (In: IEEE/RSJ IROS) 2006–2011 2.5, 7.1, 7.2

[57] Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: FM. Volume 5850 of LNCS., Springer (2009) 547–562 2.5, 7.1

[58] Košecká, J., Tomlin, C., Pappas, G., Sastry, S.: 2-1/2D conflict resolution maneuvers for ATMS. In: CDC. Volume 3., Tampa, FL, USA (1998) 2650–2655 2.5, 7.2

[59] Bicchi, A., Pallottino, L.: On optimal cooperative conflict resolution for air traffic management systems. IEEE Trans. ITS **1** (2000) 221–231 2.5, 7.2

[60] Pallottino, L., Scordio, V., Frazzoli, E., Bicchi, A.: Decentralized cooperative policy for conflict resolution in multi-vehicle systems. IEEE Trans. on Robotics **23** (2007) 2.5, 7.2

[61] Platzer, A.: Logics of dynamical systems. In: LICS, IEEE (2012) 13–24 3.3, 5.3

[62] Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. **20** (2010) 309–352 3.4, 7.4, 7.4

[63] Kozen, D.: The design and analysis of algorithms. Springer (1992) 1

[64] Platzer, A.: Differential game logic. ACM Trans. Comput. Log. **17** (2015) 1:1–1:51 4

[65] Platzer, A.: A uniform substitution calculus for differential dynamic logic. In Felty, A.P., Middeldorp, A., eds.: CADE. Volume 9195 of LNCS., Springer (2015) 467–481 3.6.1, 3.6.2

[66] Walter, W. In: Ordinary Differential Equations. Springer (1998) 105–157 3.6.1

[67] Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In Dawar, A., Veith, H., eds.: CSL. Volume 6247 of LNCS., Springer (2010) 469–483 4.1, 4.2, 4.3, 4.5.2, 4.7.1

[68] Lygeros, J., Lynch, N.: Strings of vehicles: Modeling safety conditions. In Henzinger, T., Sastry, S., eds.: HSCC. Volume 1386 of LNCS., Springer (1998) 273–288 4.2

[69] Dolginova, E., Lynch, N.: Safety verification for automated platoon maneuvers: A case study. In Maler, O., ed.: HART, Springer (1997) 154–170 4.2

[70] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified (2011) Electronic proof and demo: `http://www.ls.cs.cmu.edu/dccs/`. 4.5.2, 4.6.1

[71] Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. Technical Report CMU-CS-11-107, Carnegie Mellon University (2011)

[72] Germann, S.: Modellbildung und Modellgestützte Regelung der Fahrzeuglängsdynamik. In: Fortschrittsberichte VDI, Reihe 12, Nr. 309, VDI Verlag (1997) 4.6.2

[73] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., Qadeer, S., eds.: CAV. Volume 6806 of LNCS., Springer (2011) 585–591 5.2

[74] Frehse, G., Guernic, C.L., Donz, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In Gopalakrishnan, G., Qadeer, S., eds.: CAV. (2011) 379–395 5.2

[75] Hartenstein, H., Laberteaux, K.: A tutorial survey on vehicular ad hoc networks. Communications Magazine, IEEE **46** (2008) 164–171 5.2

[76] Jiang, D., Taliwal, V., Meier, A., Holfelder, W., Herrtwich, R.: Design of 5.9 GHz DSRC-based vehicular safety communication. Wireless Communications, IEEE **13** (2006) 36–43 5.2, 5.5

[77] Ahmed-Zaid, F., Bai, F., Bai, S., Basnayake, C., Bellur, B., Brovold, S., Brown, G., Caminiti, L., Cunningham, D., Elzein, H., et al.: Vehicle safety communications–applications (VSC-A) final report: Appendix volume 1 system design and objective test. Technical report (2011) 5.2

[78] Sepulcre, M., Gozalvez, J.: On the importance of application requirements in coopera-

tive vehicular communications. In: Wireless On-Demand Network Systems and Services (WONS). (2011) 124–131 5.2, 5.5

[79] Meireles, R., Boban, M., Steenkiste, P., Tonguz, O., Barros, J.: Experimental study on the impact of vehicular obstructions in VANETs. In: Vehicular Networking Conference (VNC), IEEE. (2010) 338–345 5.2

[80] Stanica, R., Chaput, E., Beylot, A.L.: Simulation of vehicular ad-hoc networks: Challenges, review of tools and recommendations. Computer Networks **55** (2011) 3179 – 3188 5.2

[81] Dhoutaut, D., Régis, A., Spies, F.: Impact of radio propagation models in vehicular ad hoc networks simulations. In: Proceedings of the 3rd international workshop on Vehicular ad hoc networks. VANET '06, New York, NY, USA, ACM (2006) 40–49 5.2

[82] Moser, S., Kargl, F., Keller, A.: Interactive realistic simulation of wireless networks. In: Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on. (2007) 161 –166 5.2

[83] Killat, M., Hartenstein, H.: An empirical model for probability of packet reception in vehicular ad hoc networks. EURASIP Journal on Wireless Communications and Networking **2009** (2009) 721301 5.2, 5.5

[84] Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. **41** (2008) 143–189 5.3

[85] Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010) 5.3

[86] Kopetz, H.: Event-triggered versus time-triggered real-time systems. In: Operating Systems of the 90s and Beyond, Springer (1991) 86–101 6

[87] Scheler, F., Schröder-Preikschat, W.: Time-triggered vs. event-triggered: A matter of configuration? In: MMB Workshop Proceedings GI/ITG Workshop on Non-Functional Properties of Embedded Systems, VDE (2006) 1–6 6

[88] Kouskoulas, Y., Renshaw, D.W., Platzer, A., Kazanzides, P.: Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In Belta, C., Ivancic, F., eds.: Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013, ACM (2013) 263–272 6.3

[89] Renshaw, D.W., Loos, S.M., Platzer, A.: Distributed theorem proving for distributed hybrid systems. In Qin, S., Qiu, Z., eds.: ICFEM. Volume 6991 of LNCS., Springer (2011) 356–371 7.1

[90] Platzer, A.: Quantified differential invariants. In Frazzoli, E., Grosu, R., eds.: HSCC, ACM (2011) 63–72 7.1

[91] Hu, J., Prandini, M., Sastry, S.: Optimal coordinated motions of multiple agents moving on a plane. SIAM Journal on Control and Optimization **42** (2003) 637–668 7.2

[92] Massink, M., Francesco, N.D.: Modelling free flight with collision avoidance. In Andler, S.F., Offutt, J., eds.: ICECCS, Los Alamitos, IEEE (2001) 270–280 7.2

[93] Dubins, L.E.: On curves of minimal length with a constraint on average curvature, and with

prescribed initial and terminal positions and tangents. Am J Math **79** (1957) pp. 497–516 7.3

[94] David Renshaw, Sarah M. Loos, A.P.: Mechanized safety proofs for disc-constrained aircraft. Technical Report CMU-CS-12-132, Carnegie Mellon (2012) 7.3