Thesis Proposal
Practical End-to-End Verification of Cyber-Physical Systems

Brandon Bohrer

July 26, 2019

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
André Platzer, Chair
Stefan Mitsch
Frank Pfenning
Tobias Nipkow, TU Munich

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

Keywords: hybrid systems, theorem proving, end-to-end verification, hybrid games, hybrid logic

Abstract

Cyber-physical systems (CPSs) combining discrete control and continuous physical dynamics are pervasive in modern society: examples include driver assistance in cars, industrial robotics, airborne collision avoidance systems, the electrical grid, and medical devices. Many of these systems are safety-critical or even life-critical because they operate in close proximity to humans and in some cases perform life-sustaining functions. Formal safety verification of these systems is a key tool for attaining the strongest possible guarantees that they meet their safety objectives. Therefore, as the importance of CPS in society grows, so does the importance of their formal verification. *Hybrid systems* models, in particular, have succeeded in providing a common formalism for the discrete and continuous aspects of a CPS. Of the available approaches to verification, hybrid systems theorem-proving in differential dynamic logic (dL) and its generalization dGL is notable for its strong logical foundations and successful application in a number of case studies using the theorem provers KeYmaera and KeYmaera X.

Hybrid systems theorem proving has been a boon to CPS verification: hybrid systems are abstract enough that verification is tractable, but detailed enough to capture realistic control logic and physics. However, hybrid-system verification results are leaky: because we are truly interested in the safety of a system *implementation*, we lose safety guarantees when implementations are not faithful to our model. Further still, a safety result in a formal logic such as dGL is lost if the software tool which checks the proof is *unsound*.

My thesis is that these leaks can be sealed by placing the entire verification toolchain, from checking of safety proofs to runtime execution, on a common foundation in constructive logic and programming languages. That is: *Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.* CdGL does so by connecting the synthesis of practical implementations down to bulletproof theoretical foundations. We show that a logical (corr. languages) approach provides a keystone against which high-level proofs can be rigorously connected to their implementations and foundations. The same approach even facilitates the proof author's connection to their proof, enabling proof languages beyond the state of the art.

The first step of the toolchain we propose is a proof language Kaisar for constructive proofs about hybrid systems (or games, since adversarial dynamics are supported). We then propose the CdGL, its constructive logical foundation. Unlike its classical predecessors dGL and dL, a CdGL proof contains, in the general case, enough information to synthesis both monitors and controllers, which respectively check that the external world meets modeling assumptions and compute model-compliant (thus safe) control choices for the CPS implementation. The push-button synthesis process itself is proven correct down to machine-code level, with an end-to-end formal guarantee in a theorem prover that the implementation code is safe. To connect proofs to their foundations, we also contribute a formalization of the KeYmaera X prover core's soundness in Isabelle/HOL, so that proofs can be automatically cross-checked in Isabelle/HOL, in the common special case of hybrid systems proofs. We evaluate our toolchain on ground robotics models, culminat-

ing in a model capable of general 2D driving with acceleration on arc-shaped paths. Modeling and implementation mutually inform each other: if the implementation is unsafe or physical assumptions are unrealistic, monitors ensure we notice during testing. More subtly, but just as important: our correct-by-construction monitors ensure our model was flexible enough for practical use, as an overly conservative model would lead to monitors which always report unsafety. Not only does our end-to-end approach provide formal guarantees of runtime safety while presenting a clean linguistic interface to proof, but in validating environmental assumptions, it forces users to be truly honest as well. if the model is overly restrictive, correct-by-construction monitors ensure we notice during testing

Chapter 1

Introduction

Cyber-physical systems (CPSs) combining discrete control and continuous physical dynamics are pervasive in modern society: examples include driver assistance in cars, industrial robotics, airborne collision avoidance systems, the electrical grid, and medical devices. Many of these systems are safety-critical or even life-critical because they operate in close proximity to humans and in some cases perform life-sustaining functions. Formal safety verification of these systems is a key tool for attaining the strongest possible guarantees that they meet their safety objectives. Therefore, as the importance of CPS in society grows, so does the importance of their formal verification. *Hybrid systems* models, in particular, have succeeded in providing a common formalism for the discrete and continuous aspects of a CPS. Of the available approaches to verification, hybrid systems theorem-proving in differential dynamic logic (dL) (Platzer, 2018a, 2008, 2017a, 2011) is notable for its strong logical foundations and successful application in a number of case studies (Jeannin et al., 2015; Loos et al., 2011; Mitsch et al., 2013; Platzer & Quesel, 2008b) using the theorem provers KeYmaera (Platzer & Quesel, 2008a) and KeYmaera X (Fulton et al., 2015). Other notable approaches include model-checking of automata-theoretic (Henzinger, 1996) models, control synthesis, and runtime monitoring.

While approaches differ, the fundamental motivation for CPS motivation is always the same: to promote the safety and correctness of real systems. The end-to-end philosophy recognizes that hybrid systems are only an abstraction of reality, so achieving safety in practice requires bridging the abstraction gaps between models and implementations. In this thesis, we extend the dL tradition to an end-to-end verification technique. We use VeriPhy as a name both for the resulting software toolchain and the general approach. The VeriPhy approach says that to achieve end-to-end guarantees, every stage of verification should be founded in logics and programming languages with well-defined formal semantics. By verifying systems *constructively*, we enable automated synthesis of correct implementations in the general case. Because the source, implementation, and proof languages all have formal semantics, we can formally show that implementations are refinements of source models and thus are safe.

The Cast. End-to-end verification is hard in large part because it must balance competing theoretical and practical needs. Throughout this thesis, we argue that the VeriPhy meets these needs better than existing approaches by providing a dialogue between three characters: the Logician, the Engineer, and the Logic-User. We now introduce each character.

The Logician follows in the formalist tradition of David Hilbert: to know something is to have a proof of it. Today's formalist goes further and says to have a proof is to have a machine-checkable derivation in a sound, formal proof calculus, and to have a sound proof calculus is to have a formal semantics against which the rules have a formal proof of soundness. The Logician holds mathematics to the highest standard possible, and they do so with good reason. CPS are often life-critical, and if all the Logician's paranoia can prevent defects in CPS, it has been worthwhile. Not only have flaws been found in designs of unverified CPS, but bugs in informal proofs are commonplace. Even bugs in proof calculi, while less common, have been uncovered through careful verification (Bohrer et al., 2017). Considering the existence of such bugs, the Logician is justified in their paranoia.

The Engineer, however, might see these efforts as misdirected. The Engineer is the one tasked with designing, building, and delivering a production CPS under time and budget constraints. The Engineer will gladly use formal modeling and verification, but only if it can show concrete safety benefits on realistic systems in a short timeframe. Techniques that appeal to the Logician might appall the Engineer because they are time-consuming or only guarantee safety of an ideal model. The Engineer would rather fix one bug in the implementation than ten bugs in an ideal model, since fixing a bug in the model is not guaranteed to improve the quality of shipped code.

The Logic-User is oft-forgotten, and sits between the Logician and Engineer on the spectrum from theory to practice. The Logic-User (sometimes called the Proof Engineer), is the person tasking with employing verification tools at scale. Unlike the Logician, the Logic-User does not obsess with the soundness of proof rules, because they trust that the Logician has implemented the verification tool correctly. The Logic-User believes in the value of a verified model, but sympathizes with the Engineer's plea that only a nuanced model could hope to capture the difficulties faced in practice. Verification takes time, but the Logic-User expects their time to spent wisely: time should not be wasted on verification tasks that could easily be automated away, and no unnecessary barriers to learning the tool should be erected.

As shown in Table 1.1, no prior approach satisfies all characters. General-purpose interactive theorem provers (GPITP) such as Isabelle/HOL or Coq satisfy the Logician's desire for solid logical foundations, but the Logic-User will be dissatisfied with the laborious proofs required when specialized support for CPS is absent, and the Engineer will be dissatisfied that practical artifacts like controllers and runtime monitors must still be implemented manually. The Engineer can synthesize artifacts automatically if hybrid automata are used as the foundation, but the Logician will be disappointed with the paper-only foundations and the Logic-User disappointed by error-prone modeling constructs. Of the available choices, the Logic-User prefers domain-specific logics like dL: a program-like syntax helps avoid modeling mistakes, while proof is not as laborious as with GPITP. This comes at cost to the Engineer because state-of-the-art synthesis tools for dL (Mitsch & Platzer, 2016) only synthesize monitors, and sometimes fail to synthesize monitors for complicated models in practice. Prior to this thesis, the Logician also paid the price that dL's foundations were developed only informally on paper.

Nor is the Logician truly satisfied with any approach: while GPITP's have rigorous foundations and support extraction of controllers or monitors, the extraction facilities are not provably correct, nor can the Logician convince themself that the modeled the system correctly, as the models of CPS in GPITP are ad-hoc by comparison to automata and dL's hybrid programs.

The goal of this thesis is to satisfy the needs of all three characters at once, a goal we call

Approach	Logician	(Connection)	Engineer	Logic-User
GPITP	formal	⇢	manual effort	labor-intensive
Automata	paper	⇢	monitors, controls	error-prone
dL before	paper	⇢	some monitors	less error-prone, less labor-intensive
dL after	formal	→	some monitors	less error-prone, less labor-intensive
CdGL	formal	→	monitors, controls	least error-prone, less labor-intensive

Table 1.1: Comparison of Verification Approaches

"Practical, End-to-End Verification of CPS". The terms "practical" and "end-to-end" are notoriously vague; to avoid ambiguity, this thesis defines verification to be end-to-end if it connects the formal end to the implementation end, meaning the real implementation of the CPS obeys some precise formal guarantee in terms of the system model, which itself rests on a bulletproof mathematical foundation. We say that verification is practical if it also meets the needs of the Logic-User, by simplifying models, minimizing the risk of serious modeling errors, and simplifying proofs.

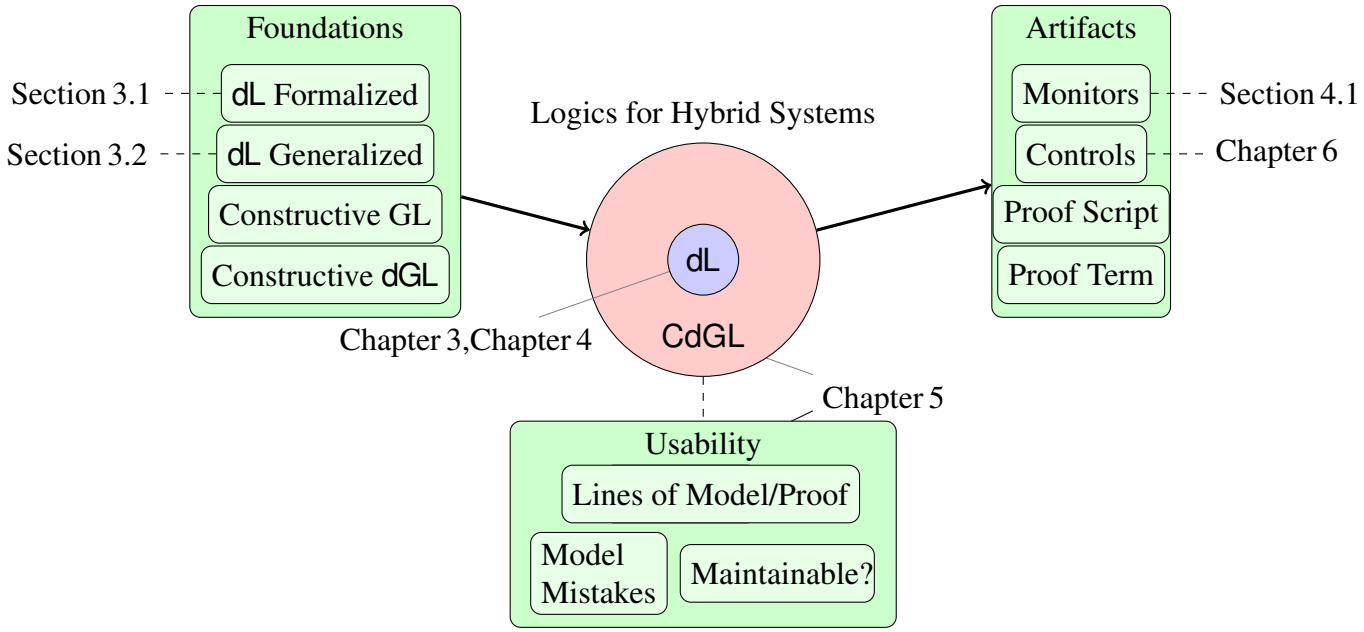


Figure 1.1: Goals of the thesis

Fig. 1.1 lays out the relationships between the chapters of the thesis. The already-completed works in this thesis connect implementation artifacts to foundations by way of dL: the monitor synthesis tool VeriPhy (Section 4.1) connects an implementation (for example, of a robot: Section 4.2) to a formal model in dL. Chapter 3 then satisfies the logician by putting dL (as implemented in KeYmaera X) on firm footing in Isabelle/HOL (Section 3.1), and also generalizing the foundations of dL to survive future development (Section 3.2). The proposed work extends our logical foundation by developing CdGL (Constructive dGL) and Kaisar, a principled language for CdGL proofs. We show that this foundation promotes simpler models (Chapter 5) and

supports synthesis of both controllers and monitors in the general case. These additions meet the practical needs of the Engineer and Logic-User while extending the VeriPhy pipeline on one end to high-level user proofs scripts and rounds out the other end by providing verified controllers, resulting in a stronger safety guarantee than prior versions of VeriPhy.

At last, we arrive at a concise thesis statement for the thesis as a whole:

> Constructive Differential Game Logic (CdGL) enables practical, end-to-end verification of cyber-physical systems.

In Section 1.1, we will compare the CdGL +VeriPhy approach in detail with other works that follow an end-to-end philosophy. Our take on end-to-end verification is that results implementation-level guarantees must come via formal guarantees connecting implementation to models, else the Logician will not accept the system as verified. For this verification to be practical, results should be transported to implementation level automatically, with artifacts that the Engineer can use. The Logic-User must be able to perform their proofs in a high-level, productive language. Viewed through this lens, Table 1.1 shows that the VeriPhy approach, with its strong foundation in logic, marries practical end-to-end verification foundation with rigorous foundations on a level that prior approaches have not.

1.1 Related Work

Here, we discuss works that are broadly related to the thesis as a whole. Works which are relevant only to a given chapter are discussed in that chapter.

1.1.1 Verification of Theorem Provers

The contributions of the thesis include a formalization of dL in Isabelle/HOL (Section 3.1) which has been used to generate a verified proof-term checker for dL (Section 4.1). We compare these contributions to other works on theorem prover verification.

Barras and Werner (Barras & Werner, 1997) verified a typechecker for a fragment of Coq in Coq. Harrison (Harrison, 2006b) verified (1) a weaker version of HOL Light's kernel in HOL Light and (2) HOL Light in a stronger variant of HOL Light. Myreen et al. have extended this work, verifying HOL Light in HOL4 (Myreen et al., 2013; Kumar et al., 2016a) and using their verified compiler CakeML (Kumar et al., 2014) to ensure these guarantees apply at the machine-code level. Myreen and Davis proved the soundness of the ACL2-like theorem prover Mitawa in HOL4 (Myreen & Davis, 2014). Anand, Bickford, and Rahli (Anand & Rahli, 2014a; Rahli & Bickford, 2016) proved the relative consistency of Nuprl's type theory (Constable et al., 1986; Allen et al., 2006) in Coq with the goal of generating a verified prover core. We share a common goal: formally verifying that theorem provers are correct. However, the foundations of dL differ greatly from those of the other provers (dL intimately deals with programs with differential equations), leading to substantially different proofs (in our case, intricate proofs about real analysis and substitution).

Our verified checker supports a significant fragment of the KeYmaera X core, including enough to recheck proofs of safety for monitor-based synthesized controllers (≈100K steps). While the Isabelle/HOL formalization is based directly on the calculus supported in KeYmaera X, another contribution of the thesis (Section 3.2) is to develop a foundation for dL which is more amenable to the practical extensions used in KeYmaera X in practice.

1.1.2 Verification of CPS

Hybrid systems verification is a broad and actively studied field. We discuss some of the major lines of research here.

Reachability Analysis

Model-checking for hybrid systems by reachability analysis is thoroughly studied. Well-known model-checkers include SpaceEx (Frehse et al., 2011), Flow* (Chen et al., 2013), and C2E2 (Duggirala et al., 2015). Compared to deductive verification in dL, the strength of model-checkers is that support a higher degree of automation. Their downsides typically include a significantly larger trusted base (≈100,000 SLOC for SpaceEx) and reduced expressivity. Typical restrictions include restriction to linear differential equations, bounded-time safety, or bounded state spaces. In a notable exception regarding large trusted cores, (Immler, 2015) verifies a set-based reachability analysis for ODEs in Isabelle using his differential equations library (Immler & Traut, 2016). While it is trustworthy, it only computes numerical solutions to differential equations, and so cannot handle full hybrid systems nor the symbolic reasoning necessary to show correctness in the general case. Reachability analysis has been applied to ground robotics (Chen et al., 2015),

Differential Dynamic Logics

This thesis is part of a long line of work using differential dynamic logic (dL) (Platzer, 2008, 2012b, 2017a) and logics descended from it. Compared to reachability analysis approaches, deductive verification in dL is symbolic and expressive: all polynomial differential equations are supported, and discrete tests and assignments are also polynomial. Guarantees are typically unbounded-time, and the plant is modeled faithfully as a continuous differential equation, not a discrete difference equation. The tradeoff is that verification is undecidable, and serious case studies usually require significant manual interaction from the user. The KeYmaera X (Fulton et al., 2015) theorem prover implements dL and provides significant automation to minimize the number of interactions required. dL and KeYmaera X have been applied in a number of case studies (Platzer, 2016).

Our case study (Section 4.2) generalizes a dL proofs of 1D straight-line motion (with direct velocity control) (Bohrer et al., 2018), of 2D obstacle avoidance, and 1D liveness (Mitsch et al., 2013, 2017). A paper proof of liveness using dL rules assumed perfect sensing and constant speed (Martin et al., 2017). Their controllers, like ours, are closely related to the classic DYNAMIC-WINDOW (Fox et al., 1997) control algorithm. In contrast to the prior dL effort, we offer stronger results, including 2D liveness, waypoint-following, and end-to-end correctness.

Synthesis for Planning and Control

The completed works on robotics verification (Section 4.2) and end-to-end verification (Section 4.1) do not explicitly verify the planner and controller, though they could in principle be combined with a verified planner (Rizaldi et al., 2018). An alternative approach is to synthesize the controller and/or planner, and we will explore controller synthesis in the proposed work (Chapter 6) Others have explored synthesis of plans and controls, but none of them address correctness of low-level feedback control, and all trust the correctness of their kinematic models:

- The tools LTLMoP (Finucane et al., 2010) and TuLiP (Filippidis et al., 2016) can synthesize robot controls that satisfy a high-level temporal logic specification. However, they depend on accuracy of their kinematic models, which assume discrete-space and discrete-time, and the guarantees are not end-to-end.
- Bisimulation (Alur et al., 2000) is used to synthesize plans based on hybrid dynamics (Bhatia et al., 2010; Fainekos et al., 2009), but assumes model compliance and cannot ensure low-level controller correctness.
- Controllers have been synthesized: *i)* from temporal logic specifications for linear systems (Kloetzer & Belta, 2008), *ii)* for adaptive cruise control (Nilsson et al., 2016), tested in simulation and on hardware, and *iii)* from safety proofs (Taly & Tiwari, 2010) for switched systems using templates, but those works assume model compliance and cannot ensure low-level controller correctness.

Online Verification

The approaches above are *offline*: i.e., they are used before-the-fact to ascertain the correctness of a hybrid model of a CPS. In contrast, *online verification* (or runtime verification) uses checks at runtime to enforce correctness. Online verification excels at treating systems that are not fully understood before runtime (such as sensors and actuators) and systems which we simply wish to not try to fully understand (such as complex untrusted control systems). However, the guarantees of online verification are weaker and often lack a clear foundation.

- The basis of online verification is the SIMPLEX (Krogh, 1998) method, which uses a trusted monitor to decide between an untrusted controller and trusted fallback.
- Runtime reachability analysis has been used for car control (Althoff & Dolan, 2014), but relies on correctness of the model and of the analysis implementation. Since reachability tools rely on large, complicated codebases and models are challenging to get right, these assumptions present a correctness gap which stands in the way of an end-to-end argument.
- ModelPlex is a tool for synthesizing monitors (for both the controller and plant) from proven-safe dL models in KeYmaera X. However, it simply synthesizes a dL formula describing the monitored conditions, not an executable program. Its present implementation only supports special cases of dL and often fails mysteriously when applied outside this fragment.
- High-Assurance SPIRAL (Franchetti et al., 2017) is a compilation toolchain for ModelPlex-synthesized monitors for dL, but does not provide formal end-to-end guarantees, and has not been used to develop an end-to-end system capable of free-range 2D driving.

- A contribution of this thesis is the VeriPhy (Bohrer et al., 2018) toolchain for dL. VeriPhy extends SIMPLEX by ensuring the monitor is correct-by-construction (via ModelPlex), formally proving the safety of the resulting system, and automatically maintaining safety down to machine code implementation.

End-to-End Approaches

End-to-end verification refers to any verification approach which attempts to verify CPS as they exist in reality while maintaining a rigorous foundation. Notably, none of these approaches show correctness unconditionally: all have a trusted base and make simplifying assumptions about reality.

- High-Assurance SPIRAL (Franchetti et al., 2017) generates executable x64 machine code based on dL model. However, its claim to "end-to-end" status is weaker than that for our tool VeriPhy, because High-Assurance SPIRAL relies heavily on esoteric compiler optimizations which are difficult to verify correct. Moreover, VeriPhy adopts a deeper understanding of hybrid systems: several stages of VeriPhy exploit language-level semantic proofs that link the initial hybrid system model to a low-level executable semantics. High-Assurance SPIRAL uses only syntactic transformations and does not have a semantic understanding of hybrid system, which precludes the clear connection between the system model and compiled code as provided by VeriPhy.
- Anand and Knepper (Anand & Knepper, 2015) developed the framework ROSCoq based on the Logic of Events for reasoning about distributed CPS in Coq with constructive reals and generating verified controllers. Their use of constructive reals is prescient, and inspires the design choices in Chapter 6. However, they provide limited support for reasoning about derivatives, requiring extensive manual proofs by the user. They do not synthesize and automatically verify monitors, nor is their machine code verified.
- VeriDrone by Ricketts et al. (Ricketts et al., 2015) is a framework for verifying hybrid systems in Coq that relies on a discrete-time temporal logic called RTLA, inspired by TLA (Lamport, 1992). They prove an analog of DI, but not the other ODE axioms, thus other ODE proofs, as well as arithmetic proofs, are likely laborious. They do not automatically verify monitors, nor is their machine code verified. Their use of discrete-time temporal logic also raises foundational concerns, since real time is best understood as continuous.

Chapter 2

Background: Differential Game Logic dGL

Differential dynamic logic (dL) (Platzer, 2018a, 2008, 2017a, 2011) is a dynamic logic for the formal verification of hybrid systems, which combine discrete transitions with differential equations to model CPSs. *Differential game logic* (dGL) (Platzer, 2015, 2017b, 2018b) extends the programs α of dL with a turn-taking operator α^d to enable representing and verifying hybrid *games* combining discrete, continuous, and adversarial dynamics. While the already-completed thesis chapters (Chapter 3, Section 4.1) employ dL, we present full dGL since i) the proposed works (Chapter 5,Chapter 6) will require an understanding of full dGL, and ii) an introductory understanding of dGL will also provide an introductory understanding of dL. While the syntax of dGL and dL are nearly identical, their semantic differences are deep: dL is a regular modal logic whose denotational semantics can be and are given in the forward-chaining, relational Kripke style, while dGL is a subregular logic with a backward-chaining winning-region semantics. While many basic axioms are the same in dL and dGL, some are necessarily different because dGL is subregular: for example, Kripke's axiom K fails in dGL and is in practice subsumed by a monotonicity rule.

2.1 Syntax

We introduce the syntax and informal meaning of dGL here, then introduce the winning-region semantics formally in Section 2.2. The syntax of dGL consists of three syntactic classes: terms, formulas, and hybrid games.

Definition 1 (Terms of dGL)**.** Terms θ, η of dGL are defined recursively according to the following grammar:

$$\theta, \eta ::= q \mid x \mid \theta + \eta \mid \theta \cdot \eta \mid (\theta)'$$

Where $q \in \mathbb{Q}$ is a rational constant, $x \in \mathcal{V}$ is a program variable and $\mathcal{V}$ is the (at most countable) set of all base variable names. For every base variable $x \in \mathcal{V}$, there is a differential variable $x' \in \mathcal{V}'$ standing for the differential of x. Terms $\theta + \eta$ and $\theta \cdot \eta$ are the sum and product of θ and η, and $(\theta)'$ is the *differential* of term θ. It is worth noting that the dGL term language is restrictive, and for good reason: all terms are polynomials, so all terms are defined in every state and even C^∞-smooth. Because dGL terms are well-behaved, the theory of dGL is simplified as a result. In Section 3.2, we show how to remove these simplifying assumptions in the context of

dL, enabling many more term constructs that are essential in practical proving, the most common constructs including quotients, roots, trigonometric functions, and conditionals.

Definition 2 (Formulas of dGL)**.** Formulas are defined by the following grammar:

$$\phi, \psi ::= \theta \sim \eta \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \phi \to \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

where $\sim$ stands for any comparison operator $\sim \in \{\leq, <, =, \neq, >, \geq\}$. We write conjunctions $\phi \wedge \psi$, negations $\phi \vee \psi$, disjunctions $\phi \vee \psi$, implications $\phi \to \psi$, biimplications $\phi \leftrightarrow \psi$, and quantifiers over real numbers $\exists x\, \phi$ and $\forall x\, \phi$. The diamond modality $\langle\alpha\rangle\phi$ says that the player who is actively making decisions (typically named Angel) has a strategy for the game α which ensures postcondition ϕ. The box modality $[\alpha]\phi$ says that the player who is not currently making decisions (typically named Demon) has a strategy for the game α which ensures postcondition ϕ. Because dGL is a classical logic, it is worth noting that the formula syntax is not minimal: many constructs such as implication, equivalence, disjunction, universal quantifiers, and the box modality are all definable classically. This will not be the case in the proposed logic CdGL because these dualities do not hold constructively.

Definition 3 (Hybrid games)**.** Games are defined by the following grammar:

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

where $x := \theta$ stores the current value of term θ in program variable x and test $?\phi$ makes Angel lose if formula ϕ does not hold in the current state. In nondeterministic assignment $x := *$, Angel chooses a value $r \in \mathbb{R}$ to assign to x. Program $x' = \theta \,\&\, \phi$ evolves x continuously according to the differential equation (ODE) $x' = \theta$ for a duration $d \geq 0$ of Angel's choosing, but Angel must ensure that ϕ is true throughout the evolution of $x' = \theta$, else they lose. In game $\alpha \cup \beta$, Angel chooses whether to play game α or game β. In game $\alpha;\ \beta$ the players must first play α and then play β; this case is largely responsible for the need to employ backward-chaining semantics in dGL, as it is most natural to first ask from what region X β is winnable, then ask from what region does Angel have a strategy to reach X when playing α. In the iterated game α^*, Angel decides at the end of each loop iteration whether to continue playing α another time. All plays must be finite (Angel must stop eventually), but Angel need not decide a-priori *when* to stop, let alone announce that decision to Demon beforehand. If Angel cannot reach the goal region without repeating α infinitely, then Angel loses. To play the dual game α^d, Angel and Demon switch roles then play game α in their new roles. That is, the player previously known as Demon makes the decisions in α, at least until another dual operator is encountered. We parenthesize hybrid games $\{\alpha\}$ for clarity and disambiguation as needed.

We now give an example hybrid game and example safety and liveness properties. The game in Example 1 is a generalization of a standard hybrid systems model to a hybrid game.

Example 1 (Acceleration-Controlled 1D Driving).

$$\alpha_\times \equiv \tag{2.1}$$

$$v := 0;\ x := 0; (obs := *; T := *; A := *; B := *; ?obs > 0 \wedge T > 0 \wedge A > 0 \wedge B > 0)^d; \tag{2.2}$$

$$\big\{a := *;\ ? - B \leq a \wedge a \leq A; \tag{2.3}$$

$$t := 0;\ \{x' = v, v' = a, t' = 1 \,\&\, t \leq T \wedge v \geq 0\}^d\big\}^\times \tag{2.4}$$

The first line (2.2) is an initialization phase: we (Angel) call the current position $x = 0$ and are initially stopped ($v = 0$). The opponent then picks how far away our obstacle (*obs*) is and how long Angel will have to wait between control decisions (T). They also pick Angel's maximum braking rate B and maximum acceleration rate B. One could argue on principle that neither player really picks the timestep T, in practice this is determined by the speed of the robot's processors, sensors, and actuators, as well as the speed of its control program. However, what's important is that robot's controller (the Angel player) is *not* the one who picks, so a conservative model should assume the worst, which is that their opponent (the environment) gets to pick. The same goes for the rates A and B. It would also be eminently unreasonable if the obstacle already collided with the robot at the start of the game, and would be nonsensical if the control cycle lasted zero time or less, or if the robot could not accelerate or brake, thus the environment is responsible for picking positive obs, T, A, and B, else they lose by default and Angel wins immediately. The remainder of the game is a *demonic loop* $\alpha^\times$ where the Demon player controls the duration, which can be derived $\alpha^\times \equiv \alpha^{d*d}$. Angel's turn consists of making a control decision by setting the acceleration a. Angel has near complete freedom to set the acceleration: they may set any value ($a := *$) so long as it is within the physical limits of the car ($? - B \leq a \wedge a \leq A$). It is Angel's responsibility to do so, and they lose the game if the acceleration is out of bounds. Next, the car moves according to ideal Newtonian physics: the velocity v continuously changes according to acceleration a while position changes continuously according to the velocity. Demon (the environment) controls the duration of the ODE, but is constrained to the timestep $t \leq T$: i.e., Angel's control action must be safe even if Demon chooses not to use the full time budget, but need not be safe past time T : in that case Demon is responsible for breaking the rules, so Angel wins. The equation $t' = 1$ simply says that t represents the time elapsed in the current ODE run.

We write α_* for $\alpha_\times$ with an Angelic loop.

We give examples of safety and liveness theorems:

Example 2 (1D Game Safety and Liveness).

$$\textit{safe} \equiv \langle \alpha_\times \rangle x \leq obs$$
$$\textit{live} \equiv \langle \alpha_* \rangle (x \geq obs \vee t \leq \frac{T}{2})$$

The safety theorem *safe* says that Angel (the robot) has a strategy to ensure $x \leq obs$ (i.e. a strategy not to hit the obstacle) regardless how long Demon runs the loop and the ODE. The liveness theorem *live* says that Angel (the robot) has a strategy to reach the obstacle, *so long as* Demon always runs the ODE long enough ($t \geq \frac{T}{2}$). The requirement on running the ODE long enough is to rule out *Zeno* behaviors by Demon: without this restriction, Demon can run the ODE for shorter and shorter durations, with infinitely many iterations in finite time, which is not physically realizable but would falsify the liveness theorem.

It is worth noting that prior case studies (Jeannin et al., 2015; Loos et al., 2011; Mitsch et al., 2013; Platzer & Quesel, 2008b) used vanilla dL for hybrid *systems* in KeYmaera X (Section 4.1,Section 4.2). In Chapter 5 of this thesis, we will advocate for making (constructive) games proofs the standard. We foreshadow the motivations of Chapter 5 here.

Consider a hybrid *system* version of Example 1:

Example 3 (Acceleration-Controlled 1D Driving System).

$$\text{Pre} \equiv v = 0 \land x = 0 \land obs > 0 \land T > 0 \land A > 0 \land B > 0 \tag{2.5}$$

$$\alpha_{\text{Sys}} \equiv \tag{2.6}$$

$$\big\{\big\{\ a := *;\ ?-B \leq a \land a \leq A;\ ?\frac{(v+aT)^2}{2B} \leq obs - \left(x + vT + a\frac{T^2}{2}\right) \tag{2.7}$$

$$\cup\ ?v = 0;\ a := 0 \tag{2.8}$$

$$\cup\ ?v \geq 0;\ a := -B\big\};\ \tag{2.9}$$

$$\{x' = v, v' = a, t' = 1\ \&\ t \leq T \land v \geq 0\}\big\}^* \tag{2.10}$$

Note α_{Sys} of Example 3 is much more complex and explicit than Example 1. This is because the *strategy* by which acceleration is chosen must be explicitly given in the model as three cases: (2.7) allows any physically-possible choice that is safe for time T, (2.8) lets a robot stay stopped, and (2.9) allows braking.

Why are Example 1 and game models in general simpler? A typical game model has shape $(\mathsf{ctrl}^d;\ \mathsf{plant})^*$, where the turn-taking (or dual) operator α^d has the affect of alternating between the modalities $\langle\alpha\rangle\phi$ and $[\alpha]\phi$. Most often, we wish to show that there *exists* a control choice where *all* behaviors of the environment are safe and live, and so on for all iterations of the system. The ability to express this pattern is exactly what distinguishes dGL from dL. In vanilla dL, safety is usually expressed in the form $\phi \to [(\mathsf{ctrl};\ \mathsf{plant})^*]\psi$ and liveness is approximated as $\phi \to \langle(\mathsf{ctrl};\ \mathsf{plant})^*\rangle\psi$ or $\phi \to [(\mathsf{ctrl};\ \mathsf{plant})^*]\langle(\mathsf{ctrl};\ \mathsf{plant})^*\rangle\psi$. dGL models are simpler because in typical dL usage, *all* cases of a controller must be safe, whereas a safe choice need only *exist* in a game. As a result, operational information (what control decision is made in which circumstances) is present in dL models, whereas it need only appear in *proofs* in dGL.

As discussed further in Chapter 5, relegating control choices to the proof has advantages for synthesis in CdGL as well as simplicity advantages. That being said, the maturity of KeYmaera X and the many available case studies show the utility of the thesis chapters which target vanilla dL as well. The connections between vanilla dL and CdGL will be further discussed in Chapter 5.

2.2 Semantics

We give the denotational semantics for classical hybrid game logic dGL, which is divided into a semantics for terms, formulas, and games. Throughout the semantics, states are written $\omega, \nu, \mu : \mathcal{S}$ where the set of states $\mathcal{S}$ is in bijection to $\mathbb{R}^n$ for $n = |\mathcal{V} \cup \mathcal{V}'|$ where $\mathcal{V}$ is the set of base variables and $\mathcal{V}'$ the set of differential variables.

Definition 4 (Classical term semantics)**.** The semantics of a term $[\![\theta]\!]\omega : \mathbb{R}$ is the value of term θ

in state ω, and is defined inductively on θ:

$$\begin{aligned}
[\![q]\!]\omega &= q \\
[\![x]\!]\omega &= \omega(x) \\
[\![\theta+\eta]\!]\omega &= [\![\theta]\!]\omega + [\![\eta]\!]\omega \\
[\![\theta\cdot\eta]\!]\omega &= [\![\theta]\!]\omega \cdot [\![\eta]\!]\omega \\
[\![(\theta)']\!]\omega &= \sum_{x\in\mathcal{V}} \frac{\partial [\![\theta]\!]\omega}{\partial x} \cdot \omega(x')
\end{aligned}$$

That is, literals denote themselves, variables x take their meaning from the state ω, sums and products denote the sum and product of the denotation of their operands respectively, and the differential $(\theta)'$ denotes the sum of every partial derivative of the denotation of θ, each scaled by the value of the corresponding differential variable x'. Because every term mentions at most a finite number of variables and the partial with respect to an unmentioned variable is uniformly zero, this sum always has finite support. An advantage of this semantics is that it is defined in every state while agreeing with the intuitive meaning of "time derivative of $[\![\theta]\!]\omega$" whenever a differential appears in the postcondition of an ODE.

Definition 5 (Classical formula semantics)**.** The semantics of a formula ϕ is given by the relation $[\![\phi]\!]\omega$, which is defined inductively on ϕ

$$\begin{aligned}
[\![\theta\sim\eta]\!] &= \{\omega \mid [\![\theta]\!]\omega \sim [\![\eta]\!]\omega\} && \text{for } \sim\in\{\leq,<,=,\neq,>,\geq\} \\
[\![\neg\phi]\!] &= [\![\phi]\!]^C \\
[\![\phi\wedge\psi]\!] &= [\![\phi]\!]\omega \cap [\![\psi]\!]\omega \\
[\![\phi\to\psi]\!] &= \{\omega \mid \omega\in[\![\phi]\!]\omega \text{ implies } \omega\in[\![\psi]\!]\} \\
[\![\phi\leftrightarrow\psi]\!] &= \{\omega \mid \omega\in[\![\phi]\!]\omega \text{ iff } \omega\in[\![\psi]\!]\} \\
[\![\exists x\,\phi]\!] &= \{\omega \mid \omega^r_x\in[\![\phi]\!]\} && \text{for some } x\in\mathbb{R} \\
[\![\forall x\,\phi]\!] &= \{\omega \mid \omega^r_x\in[\![\phi]\!]\} && \text{for all } x\in\mathbb{R} \\
[\![\langle\alpha\rangle\phi]\!] &= \varsigma_\alpha([\![\phi]\!]) \\
[\![[\alpha]\phi]\!] &= \delta_\alpha([\![\phi]\!])
\end{aligned}$$

where $\varsigma_\alpha(X)$ and $\delta_\alpha(X)$ are the regions from which Angel (or Demon, respectively) has a strategy to reach a region X.

Definition 6 (Classical game semantics)**.** For any game α and goal region $X\subseteq\mathcal{S}$, we inductively define the *winning region* $\varsigma_\alpha(X)$ for Angel, i.e., the region from which they have a strategy to

enter region X after playing game α:

$$\begin{aligned}
\varsigma_{x:=\theta}(X) &= \{\omega \in \mathcal{S} \mid \omega_x^{[\![\theta]\!]\omega}\} \\
\varsigma_{x:=*}(X) &= \{\omega \in \mathcal{S} \mid \omega_r^{[\![\theta]\!]\omega} \text{ for some } r \in \mathbb{R}\} \\
\varsigma_{x'=\theta \,\&\, \psi}(X) &= \{\varphi(0) \in \mathcal{S} \mid \varphi(r) \in X \text{ for some } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0,r] \to \mathcal{S} \\
&\qquad \text{such that } \varphi(\zeta) \in [\![\psi]\!] \text{ and } \frac{\partial \varphi(t)(x)}{\partial t}(\zeta) = [\![\theta]\!]\varphi(\zeta) \text{ for all } 0 \leq \zeta \leq r\} \\
\varsigma_{?\phi}(X) &= [\![\phi]\!] \cap X \\
\varsigma_{\alpha \cup \beta}(X) &= \varsigma_\alpha(X) \cup \varsigma_\beta(X) \\
\varsigma_{\alpha;\beta}(X) &= \varsigma_\alpha(\varsigma_\beta(X)) \\
\varsigma_{\alpha^*}(X) &= \bigcap \{Z \subseteq \mathcal{S} \mid X \cup \varsigma_\alpha(Z) \subseteq Z\} \\
\varsigma_{\alpha^d}(X) &= (\varsigma_\alpha(X^C))^C
\end{aligned}$$

Likewise, we define the winning regions $\delta_\alpha(X)$ for *Demon*, which are dual to those for Angel:

$$\begin{aligned}
\delta_{x:=\theta}(X) &= \{\omega \in \mathcal{S} \mid \omega_x^{[\![\theta]\!]\omega}\} \\
\delta_{x:=*}(X) &= \{\omega \in \mathcal{S} \mid \omega_r^{[\![\theta]\!]\omega} \text{ for all } r \in \mathbb{R}\} \\
\delta_{x'=\theta \,\&\, \psi}(X) &= \{\varphi(0) \in \mathcal{S} \mid \varphi(r) \in X \text{ for all } r \in \mathbb{R}_{\geq 0} \text{ and (differentiable) } \varphi : [0,r] \to \mathcal{S} \\
&\qquad \text{such that } \varphi(\zeta) \in [\![\psi]\!] \text{ and } \frac{\partial \varphi(t)(x)}{\partial t}(\zeta) = [\![\theta]\!]\varphi(\zeta) \text{ for all } 0 \leq \zeta \leq r\} \\
\delta_{?\phi}(X) &= ([\![\phi]\!])^C \cup X \\
\delta_{\alpha \cup \beta}(X) &= \delta_\alpha(X) \cap \delta_\beta(X) \\
\delta_{\alpha;\beta}(X) &= \delta_\alpha(\delta_\beta(X)) \\
\delta_{\alpha^*}(X) &= \bigcup \{Z \subseteq \mathcal{S} \mid Z \subseteq X \cap \delta_\alpha(Z)\} \\
\delta_{\alpha^d}(X) &= (\delta_\alpha(X^C))^C
\end{aligned}$$

2.3 Proof Calculus

We recite the proof calculus for dGL in Fig. 2.1. The proof calculus is given as a Hilbert system: axioms are repeatedly applied to recursively decompose hybrid games. Axiom $[\cdot]$ says that the modalities $\langle\alpha\rangle\phi$ and $[\alpha]\phi$ are dual to each other for arbitrary games α, whereas the axiom $\langle^d\rangle$ says game α^d is dual to α. In practical proofs it is often easier to say that the α^d operation switches players and then plays α, e.g. the derived axiom:

$$\langle\alpha^d\rangle\phi \leftrightarrow [\alpha]\phi$$

Axiom $\langle:=\rangle$ says assignments can be eliminated by substituting in the postcondition, while $\langle:*\rangle$ says random assignments can be converted to quantifiers. Axiom $\langle'\rangle$ says that if there exists a function $y(t)$ which solves an ODE, then the ODE can be eliminated by substituting in $y(t)$. This axiom gives the case where there is no domain constraint because ODEs with domain constraints

$$([\cdot]) \quad \neg\langle\alpha\rangle\neg\phi \leftrightarrow [\alpha]\phi$$

$$(\langle:=\rangle) \quad \phi(\theta) \leftrightarrow \langle x := \theta\rangle\phi(x)$$

$$(\langle:*\rangle) \quad \exists x\, \phi(x) \leftrightarrow \langle x := \theta\rangle\phi(x)$$

$$(\langle'\rangle) \quad \exists t{\geq}0\, \langle x := y(t)\rangle\phi \leftrightarrow \langle x' = \theta\rangle\phi$$

$$(\langle?\rangle) \quad (\phi \wedge \psi) \leftrightarrow \langle ?\psi\rangle\phi$$

$$(\langle\cup\rangle) \quad \langle\alpha\rangle\phi \vee \langle\beta\rangle\phi \leftrightarrow \langle\alpha \cup \beta\rangle\phi$$

$$(\langle;\rangle) \quad \langle\alpha\rangle\langle\beta\rangle\phi \leftrightarrow \langle\alpha;\beta\rangle\phi$$

$$(\langle*\rangle) \quad \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi \rightarrow \langle\alpha^*\rangle\phi$$

$$(\langle^d\rangle) \quad \neg\langle\alpha\rangle\neg\phi \leftrightarrow \langle\alpha^d\rangle\phi$$

$$\text{(M)} \quad \frac{\phi \rightarrow \psi}{\langle\alpha\rangle\phi \rightarrow \langle\alpha\rangle\psi}$$

$$\text{(FP)} \quad \frac{\phi \vee \langle\alpha\rangle\psi \rightarrow \psi}{\langle\alpha^*\rangle\phi \rightarrow \psi}$$

Figure 2.1: Selected axioms for dGL

are derivable from those without them. The practicality of this axiom depends on the complexity of $y(t)$. Especially when $y(t)$ is not a polynomial, it is preferable to reason with *differential invariants*. Axiom $\langle ? \rangle$ says a test passes when the test condition holds. Axiom $\langle \cup \rangle$ says Angel chooses which game to play. Axiom $\langle ; \rangle$ says game $\alpha; \beta$ is played by playing α first, then β in the resulting state. Axiom $\langle * \rangle$ says a repetition game can be played by choosing between playing zero rounds or playing some strategy for the first round followed by some strategy for the repetition. Axiom M says hybrid games satisfy monotonicity. Axiom FP says repetition α^* is a fixpoint of α.

Chapter 3

Completed Work: Logical Foundations

One of the key components of *foundational* end-to-end synthesis and verification is a bulletproof theoretical foundation. Logics for verification of hybrid systems and hybrid games are the main formal tool with which we wish to develop correct CPS's, and as such it is doubly important that the logics themselves are developed with the absolute highest degree of certainty in their correctness. In this chapter, we discuss two completed works by the author. These works not only attest that an adequate theoretical foundation has been laid for the remaining chapters of the thesis, but also that the author is adequately experienced with developing logical foundations for the proposed works to succeed.

3.1 Formalization of dL in Isabelle/HOL

When we perform proofs in dL (or likewise in dGL), we wish to know that theorem we believe we have proved is indeed true. This property is called soundness, and paper proofs of soundness are provided in the papers which developed a sequent calculus (Platzer, 2008) and uniform substitution (Platzer, 2017a) calculus for dL, of which the latter is the basis for the implementation of dL in the theorem prover KeYmaera X (Fulton et al., 2015). However, proofs on paper are notoriously susceptible both to 1. errors in human logical reasoning and 2. even more often, errors by omission: paper proofs often focus on the simplest cases of the proof, providing no evidence that the implementation will continue to be correct in the most difficult cases, exactly when it is most in need of formal assurance. Errors of both kinds can be most robustly addressed by formalizing the entire development of the proof system in a general-purpose proof assistant such as Isabelle (*Isabelle*, n.d.) or Coq (*Coq Proof Assistant*, n.d.), starting from syntax and semantics up to axioms and ultimately a soundness theorem for an entire proof-checking procedure. Errors of the first kind are the easiest to address: assuming we have not accidentally exploited a bug in the host prover, an erroneous logical step in the mechanized proof will simply be rejected by the prover. To totally eliminate errors of the second kind requires a more ambitious effort: once we have formalized dL in another prover, how do we know that we have formalized every important feature and that we have not forgotten one? After all, if we forget to include a certain feature in our paper proof, we might very easily forget it in a mechanized proof as well, and because this is not a soundness error, we will not catch it simply by attempting to prove soundness in the prover.

Errors of the second kind can be addressed by completing the circle, so to speak: modern theorem provers contain code generation facilities which can be used to synthesize a proof checker from an appropriately-written formal development. If that proof checker is willing to accept the dL proofs that are constructed in practice, then there is little question to the exhaustiveness of the development, because no matter whether the paper proof and mechanized proof forgot to account for some feature from the implementation, the implementation itself certainly cannot forget. While the work involved in bringing a mechanization up to speed with the practical implementation can be significant, there is also a second payoff: the resulting proof checker is guaranteed sound by construction, and can be used as a backup for the standard KeYmaera X prover core, effectively removing the core from the trusted computing base.

The author has carried out all of the above developments for the dL uniform substitution calculus (Platzer, 2017a) using Isabelle (*Isabelle*, n.d.) as a host prover. The initial formalization is documented in (Bohrer et al., 2017) and follow-up work in (Bohrer et al., 2018). In all, the formalization includes the syntax, static and dynamic semantics, proof calculus, and soundness theorem for dL. The full formalization is ≈25,000 lines of Isabelle proof scripts. In addition, a proof checker was synthesized with the Isabelle code generation mechanism and evaluated by instrumenting KeYmaera X to output complete proof terms as it performs a proof. In (Bohrer et al., 2018) this proof checker was evaluated on a proof that a velocity-controlled car driving in a straight line is safe under the supervision of a controller generated by VeriPhy. The proof term in question is over 100,000 proof steps and is extracted directly from the actual KeYmaera X proof, attesting to the exhaustiveness of the formalization.

These results contribute to the overall goals of the thesis in several ways. Firstly, the proof checker has been used to check the correctness of dL proofs that end-to-end verified systems as produced throughout this thesis are actually safe, which in effect has bridged the high-level, practical foundation given by dL with the lower-level foundations of Isabelle. Secondly, the work requires intimate knowledge of the foundations of dL, including edge cases which were not addressed in prior works such as systems of (multiple) differential equations and a bound renaming rule, which was in fact unsound in KeYmaera X until the bug was discovered by doing the mechanized proof. Additional features include operations like division, absolute value, and min and max, which are not covered in previous presentations. This has prepared the author for the foundational work that remains in the proposed thesis chapters.

3.2 Definite Descriptions in dL

While Section 3.1 addresses the gap between theory presentations of dL and its usage in the theorem KeYmaera X, it does so in a somewhat brute force way: manually formalizing every extension that has been added to the KeYmaera X core over time. This is unmaintainable as the number of extensions grows, and moreover we would prefer a definition mechanism where new constructs can be defined without increasing the size of the core. That is to say, one of the lessons learned from the work described in Section 3.1 is that a more maintainable way to put a theorem prover on bulletproof foundations is to work from both ends at once, not just formalizing the proof calculus in great detail but also redesigning the proof calculus to be more general in the first place, so that what were once a grab bag of extensions are now elegantly described in the

core language.

Recent work by the author (Brandon Bohrer, 2019) does exactly that by introducing the logic dL$_\iota$ which extends dL with a definite description construct $\iota x\ \phi(x)$ meaning the unique x (if a unique one exists) such that $\phi(x)$ is true. The underlying reason for this extension is that an unending array of extensions to dL, both past and future, are simply new term constructs which do not fall within the restrictive polynomial language of traditional dL. Because the formula language of dL is relatively unrestricted, containing all of first-order real arithmetic and dynamic modalities, one can achieve the desired power in the term language through the definite description construct, which effectively imports the powers of the formula language into the term language.

This work is of relevance to the overall thesis in two ways. First, it justifies the choice to focus primarily on the core polynomial term language in the development of CdGL, as the foundations of additional terms (albeit a classical foundation) have now been studied in detail. Moreover, it again demonstrates the author's ability to work with the foundations of the logic: whereas the work in Section 3.1 focused on a painstakingly detailed reproduction of a proof calculus which had largely already been developed, this work required a wholesale overhaul of the semantics underlying dL, much in the way an extension from classical dGL to constructive CdGL will require another semantic overhaul.

It is perhaps surprising that dL$_\iota$ requires a new semantics in the first place. The simplicity of dL terms is exploited in several ways throughout the theory of dL, the main properties being that terms are defined everywhere and that they are everywhere infinitely differentiable and locally Lipschitz-continuous, meaning that all differential equations expressible in dL have guaranteed uniqueness and existence of solutions by the Picard-Lindelöf theorem. The simple addition of definite descriptions invalidates all of these assumptions. To deal with this generalization, the term semantics of dL$_\iota$ are made partial and the formula semantics are recast a 3-valued Łukasiewicz semantics. All axioms and proof rules of dL are then revised so that hold valid over the 3-valued semantics of dL$_\iota$ and are proven sound again with respect to the new semantics. The dL$_\iota$ calculus maintains as much generality as reasonably possible, so that for example it is possible to prove properties even about systems which do not satisfy uniqueness of solutions.

The theoretical developments for dL$_\iota$ culminate in soundness and equi-expressiveness proofs with respect to dL, which very much the same kind of results we aim to achieve for the CdGL theory.

3.2.1 Related Work

Formalizations of KeYmaera X (Bohrer et al., 2018), Coq (Barras, 2010), and NuPRL (Anand & Rahli, 2014b) all omit or simplify whichever practical features are most theoretically challenging for their specific logic: discontinuous and partial terms in KeYmaera X, termination-checking in Coq, or context management in NuPRL. When formalizations of theorem provers *do* succeed in reflecting the implementation (Kumar et al., 2016b), they owe a credit to the generality of the underlying theory: it is much more feasible to formalize a general base theory than to formalize ad-hoc extensions as they arise. Our calculus, as with modern implementations (Fulton et al., 2015) and machine-checked correctness proofs (Bohrer et al., 2017) for dL, is based on uniform substitution (Church, 1956, §35,§40): symbols ranging over predicates, programs, etc. are ex-

plicitly represented in the syntax. This improves the ease with which dL_ι can be implemented and its soundness proof checked mechanically in future work. We show that in contrast to dL, dL_ι can directly represent a hybrid system featuring such ODEs, which we base on Hubbard's leaky bucket (Hubbard & West, n.d., §4.2). Then the water drains out the cooler at a rate proportional to the square root of the current volume by Torricelli's Law (Driver, 1998), or rate 0 if the valve is closed. In contrast to the indexed variables of QdL (Platzer, 2012a), they are equipped with an induction operator, making it easier to write sophisticated computations. **??** is the *differential induction* (Platzer, 2010) Quantifier elimination rule **??** says first-order arithmetic, a fragment, is decidable (Tarski, 1998).

Chapter 4

Completed Work: End-to-End Robot Verification

This chapter describes works by the author and collaborators which introduce a general approach that enables end-to-end verification while maintaining solid logical foundations and then apply the approach in a case study on the verification of free driving for 2D Dubins-like ground robots. These works already constitute a major component of the goals of the thesis: what remains in the proposed work is to extend this beyond only verification and monitor synthesis to include controller synthesis while maintaining rigorous foundations. We begin by motivating end-to-end verification and defining it, then give our approach and case study results in greater detail.

4.1 VeriPhy Pipeline for End-to-End Verification

As discussed in the introduction (Chapter 1), end-to-end verification of CPS is in part motivated by the same factors as CPS in general: CPS such as self driving cars and robots are often safety critical because of their proximity to humans, and formal methods are a powerful tool for insuring that these critical systems are in fact safe enough to be trusted around humans. Because CPS are only growing more and more popular over time, any work that addresses their safety is poised to increase in impact, as well.

To fully motivate end-to-end verification, however, one must assess the key question of what it even *means* to verify the safety of a CPS. When we first learn how to prove theorems in math class, we may not always know how to do the proof, or we may not always know whether a proof has sufficient detail, but it is almost always clear *what* we are trying to do, because the theorems we prove come from a longstanding mathematical tradition with an established library of well-defined and well-understood mathematical terms: there is little debate today what it means to be a real number, nor what it means to be a group. As soon as we enter the realm of even discrete programs, this changes entirely. Not only might programs be written in an endless and ever changing array of programming languages, but it is no longer clear what we want to prove about a program. On one end of the spectrum we might simply prove that a program satisfies a simple property like termination or absence of segfaults and call it a day, while at the other end of the spectrum we might develop a full functional specification and prove that the entire specification

is met. In practice, we often want something in between, because the more expressive a program property is, typically the harder it is to prove. All of this is also compounded by the nature of software: while the real numbers change only occasionally (with the advent of fields like nonstandard analysis or constructive analysis ;-)), real software can change hundreds of times per day. Many interesting programs can only be proved with the intervention of humans, and it is a total nonstarter to ask a human to perform the verification of a complex program hundreds of times per day.

These issues only grow more pernicious in CPS, where physics enters the mix as well. It is equally true in CPS that code evolves and should not need to be constantly reverified by hand. It is even more true that it becomes difficult to nail down a proper specification: not only are there a variety of possible properties to prove, but because CPS's are fundamentally constrained by the laws of physics, the most general properties are often not provable because they are *simply false*: A car cannot avoid collision if it is an inch behind an intruder vehicle that slams on its breaks, nor can a car avoid collisions if its sensors exhibit an arbitrarily large error. Even once a specification has been identified, proving it is now more difficult because the verification of CPS spans several subspecialties of verification which require diverse reasoning principles. At the controls level (which is the focus of this thesis), CPS verification must reason seamlessly about both discrete control code and continuous dynamical systems (differential equations) that describe the kinematics of the system. Furthermore, these arguments are by far easiest to make at a high level of abstraction where controls are nondeterministic, variables are real-valued, and time evolves continuously (which also provides the strongest results), yet we expect results to hold of a low-level implementation where control is deterministic, variables have finite arithmetic precision, and a controller can only sense every so often. This is without even considering stochastic, distributed, and adversarial dynamics, let alone vulnerabilities in the network stack.

All of this goes to say, that in seeking end-to-end verification of CPS, in seeking the strongest possible results about CPS as they exist in actual implementation and not merely in mathematical models, *identifying the right question is half the battle*. As soon as we seek to nail down a definition of end-to-end verification, we will realize this is in fact a moving target. It is unfeasible to summarily verify every desirable property of every sensor, actuator, and controller, simply because a cornucopia of such components exist, all of which are in constant flux: If today we prove that today's controller is correct, Murphy's Law dictates that a superior controller is released for purchase tomorrow. With all the considerations in mind, it is less essential to ensure that every conceivable component of the CPS has been verified in full, but rather to devise and carry out an approach where:

1. Control software is free to evolve without reverification
2. Safety of the system as it is implemented can be tied formally to a rigorous theoretical foundation
3. When a human is required to perform verification, they are allowed to do so at a high level (hybrid systems) with their results automatically transported to implementation level
4. The architecture is open-ended enough that one could plausibly verify those components whose correctness is presently assumed.

While it is specifically because of requirement 2 that we call our work *end-to-end*, the author believes that all of the above are essential for verification to be applicable in realistic systems

while still claiming full generality and full rigor. Our approach, which we will refer to as the VeriPhy approach (as it is implemented in our tool VeriPhy), achieves all of the above, as we will now describe.

Requirement 1 effectively requires a significant *runtime verification* component, where the safety of control decisions is checked online, and potentially-unsafe decisions are replaced by provably safe ones. Requirements 2 and 3 entail an equally significant hybrid systems verification component. Requirement 4.1, arguably the easiest to satisfy, can be satisfied by providing a clean interface between those components which are proven correct and those which are trusted.

With all these requirements in mind, the VeriPhy approach starts with modeling and verifying a hybrid system in dL using KeYmaera X. The VeriPhy tool then synthesizes an *executable correct-by-construction monitor* as follows:

- The ModelPlex tool is invoked to synthesize a *control monitor formula* and *plant monitor formula*, which determine whether control decisions and physical evolutions respectively satisfy the assumptions made of them by the model
- The monitors are combined with a proven-safe *fallback controller* (which is provided in the input model) to form a *sandbox controller*. The sandbox controller applies control decisions provided from an external source (henceforth called the *external controller*) whenever those decisions satisfy the controller monitor, and otherwise invokes the fallback controller
- The sandbox controller is automatically proven safe in KeYmaera X, in part by reusing the contents of the hybrid system safety proof
- The sandbox safety proof is optionally rechecked in the verified proof checker (Section 3.1) to eliminate the KeYmaera X core from the trusted base
- The exact real arithmetic of hybrid systems is replaced with conservative fixed-point interval arithmetic, which is formally proven in Isabelle to be sound
- The resulting program is proven equivalent to a CakeML program (in HOL4)
- The equivalent CakeML program is compiled to machine code in a verified compiler
- The machine code is linked against the users' implementation of sensors and actuators

Each step is provably a refinement of the step before it, culminating that any execution of the machine-code running on the actual CPS is corresponds to some execution of the source hybrid system. Because all executions of the source hybrid system were proven safe already, then the execution of the actual CPS is also known to be safe. Notably, the sensors and actuator drivers, which are provided by the users of VeriPhy, are trusted. Yet requirement is satisfied: the sensors and actuators (correspond directly to the sensor and actuator variables in the hybrid system model) are isolated in a small foreign-function interface which has a simple specification in HOL4. It is not unrealistic to suggest that one day, we could verify sensors and actuators against these specifications in HOL4, at least given some weaker assumptions.

4.1.1 Related Work

Verified Compilation. We use the CakeML (Tan et al., 2016) verified ML compiler and its associated verification tools (Myreen & Owens, 2012; Guéneau et al., 2017). CakeML is higher-

level than other languages such as Clight with verified compilers such as floating-point CompCert (Boldo et al., 2013). This makes the verification of sandbox implementation in CakeML against hybrid systems semantics painless. We chose this over, e.g., translation validation with unverified compilers (Sewell et al., 2013) since translation validation can be brittle.

Lustre (Halbwachs et al., 1992) is a CPS-centric language with a verified compiler, Vélus (Bourke et al., 2017). Writing and compiling a Lustre controller provides no end-to-end guarantees because physical modeling and verification are left unanswered. It could be used as a code generation target, but this would be a detour because the Lustre language differs greatly from hybrid programs.

Machine Arithmetic Verification. Machine arithmetic correctness is a major VeriPhy component. We verify arithmetic soundness foundationally. This is an active research area with libraries available in HOL Light (Harrison, 2006a), Coq (Boldo & Melquiond, 2011; Daumas et al., 2001; Boldo et al., 2009; Melquiond, 2012), Isabelle/HOL (Yu, 2013), etc. Of these, only PFF in Coq (Daumas et al., 2001) provides the qualitative rounding correctness results we need, so we prove them in Isabelle/HOL using the seL4 (Klein et al., 2010) machine word library. We chose Isabelle/HOL and HOL4 over Coq because their combination of cutting-edge analysis libraries (Immler & Traut, 2016), mature formalization of dL (Bohrer et al., 2017), proof-producing code extraction (Myreen & Owens, 2012), and classical foundations positions them well for our end-to-end pipeline. Static analysis of arithmetic for hybrid systems has been studied (Martinez et al., 2010; Majumdar et al., 2012; Bouissou et al., 2009), but without foundational safety proofs for general dynamics.

4.2 Case Study: End-to-End Ground Robotics Verification

While the previous section describes VeriPhy in general (both as a methodology and as a tool), we certainly cannot claim that the VeriPhy tool enables end-to-end verification of realistic CPS implementations until we have actually applied VeriPhy to such a system. This section validates that claim by providing a full VeriPhy case study including dL models, proofs, a simulation which employs the sandbox controller generated by VeriPhy, and test environments for the simulation. For our case study, we chose 2D ground robots which follow Dubins-like arcs and lines and which enforce user-supplied speed limits as they do so. This choice is motivated by the fact that a huge variety of ground robots move by following such paths, yet the paths themselves are reasonably simple, and at the same time speed limits are incredibly useful for example in autonomous cars which intend to obey the law or in robots with high centers of mass which wish not to flip over when navigating curves.

4.2.1 Model

This section introduces our 2D robot model in dL. The Dubins dynamics are bloated by a tolerance, which accounts for imperfect actuation and for discrepancies between the Dubins dynamics and real dynamics of the implementation. That is, because realistic robots never follow a path

perfectly, we will bloat each arc to an annulus section which is more easily followed. Each waypoint is specified by coordinates (x,y), a curvature k, and a speed limit $[\mathsf{vl},\mathsf{vh}]$ for the robot's velocity v which is to be met by the time the waypoint is reached. The curvature parameter yields circular arcs (when $\mathsf{k}\neq 0$) and lines (when $\mathsf{k}=0$) as primitives.

Our hybrid program α is a time-triggered *control-plant* loop: $\alpha \equiv (\mathsf{ctrl};\mathsf{plant})^*$. We use *relative* coordinates: the robot's position is always the origin, from which perspective the waypoint "moves toward" the vehicle. This simplifies proofs (fewer variables) and implementation (real sensors and actuators are vehicle-centric). The plant is an ODE describing the robot kinematics:

$$\begin{aligned}\mathsf{plant} \equiv \{&\mathsf{x}' = -v\,\mathsf{k}\,\mathsf{y},\ \mathsf{y}' = v\,(\mathsf{k}\,\mathsf{x}-1),\ v' = a,\ \mathsf{t}' = 1\\ &\&\ \mathsf{t}\leq T\ \wedge\ v\geq 0\}\end{aligned}$$

Here, a is an input from the controller describing the acceleration with which the robot is to follow the arc of curvature k to waypoint (x,y). In the equations for x',y': *i)* The v factor models (x,y) moving at linear velocity v, *ii)* The $\mathsf{k},\mathsf{x},\mathsf{y}$ factors model circular motion with curvature k, with $\mathsf{k}>0$ corresponding to counter-clockwise rotation of the waypoint, $\mathsf{k}<0$ to clockwise rotation, and $\mathsf{k}=0$ to straight-line motion, *iii)* The additional -1 term in the y' equation shifts the center of rotation to $\left(\frac{1}{\mathsf{k}},0\right)$. The equations $v'=a$ and $\mathsf{t}'=1$ make acceleration the derivative of velocity and t stand for current time. The domain constraint $\mathsf{t}\leq T\wedge v\geq 0$ says that the duration of one control cycle shall never exceed a timestep parameter $T>0$ representing the maximum delay between control cycles and that the robot never drives in reverse.

The controller's task is to compute an acceleration a which slows down (or speeds up) soon enough that the speed limit $v\in[\mathsf{vl},\mathsf{vh}]$ is ensured *by the time* the robot reaches the goal. The controller is written:

$$\begin{aligned}\mathsf{Ann} &\equiv |\mathsf{k}|\varepsilon\leq 1\wedge\left|\frac{\mathsf{k}\,\left(\mathsf{x}^2+\mathsf{y}^2-\varepsilon^2\right)}{2}-\mathsf{x}\right|<\varepsilon\\ \mathsf{Feas} &\equiv \mathsf{Ann}\wedge \mathsf{y}{>}0\wedge 0{\leq}\mathsf{vl}{<}\mathsf{vh}\wedge AT{\leq}\mathsf{vh}-\mathsf{vl}\wedge BT{\leq}\mathsf{vh}-\mathsf{vl}\\ \mathsf{ctrl} &\equiv (\mathsf{x},\mathsf{y}):=*;\ [\mathsf{vl},\mathsf{vh}]:=*;\ \mathsf{k}:=*;\ ?\mathsf{Feas};\ \underbrace{a:=*;\ ?\mathsf{Go}}_{\mathsf{ctrl_a}}\end{aligned}$$

$$\begin{aligned}\mathsf{Go} \equiv{}& -B\leq a\leq A\wedge v+aT\geq 0\\ \wedge\Big(& v\leq \mathsf{vh}\wedge v+aT\leq \mathsf{vh}\ \vee\\ &(1+|\mathsf{k}|\varepsilon)^2\Big(vT+\frac{a}{2}T^2+\frac{(v{+}aT)^2-\mathsf{vh}^2}{2B}\Big)+\varepsilon{\leq}\|(\mathsf{x},\mathsf{y})\|_\infty\Big)\\ \wedge\Big(& \mathsf{vl}\leq v\wedge \mathsf{vl}\leq v+aT\ \vee\\ &(1+|\mathsf{k}|\varepsilon)^2\Big(vT+\frac{a}{2}T^2+\frac{\mathsf{vl}^2-(v{+}aT)^2}{2A}\Big)+\varepsilon{\leq}\|(\mathsf{x},\mathsf{y})\|_\infty\Big)\end{aligned}$$

where the plan assignment $(\mathsf{x},\mathsf{y}):=*$ chooses the next 2D waypoint, the assignment $[\mathsf{vl},\mathsf{vh}]:=*$ chooses the speed limit interval, and $\mathsf{k}:=*$ chooses any curvature. The *feasibility* test $?\mathsf{Feas}$ determines whether or not the chosen waypoint, speed limit, and curvature are *physically* attainable in the current state under the plant dynamics (e.g., it checks that there is enough remaining distance to get within the speed limit interval). We also simplify plans so all waypoints satisfy $\mathsf{y}>0$, by subdividing any violating paths automatically. This simplifies the feedback controller

and proofs. The abbreviation ctrl_a names just the control code responsible for deciding *how* the waypoint is followed rather than *which* waypoint is followed.

In Feas, formula Ann says we are within the *annulus section* (Fig. 4.1) ending at the waypoint (x, y) with the specified curvature k and width ϵ. A larger choice of ε yields more error tolerance in the sensed position and followed curvature at the cost of an enlarged goal region. Formula Ann also contains a simplifying assumption that the radius of the annulus is at least ϵ. Feas also says the speed limits are assumed distinct and large enough to not be crossed in one control cycle.

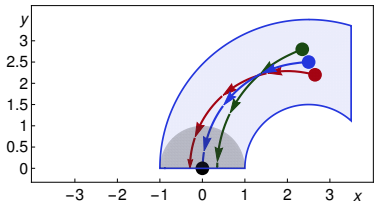


Figure 4.1: Annular section through the (blue) waypoint $(2.5, 2.5)$. Trajectories from the displaced green and red waypoints with slightly different curvatures remain within the annulus.

The *admissibility* test ?Go checks that the chosen a will take the robot to its goal with a safe speed limit, by *predicting future motion* of the robot. We illustrate this with the upper bound conditions. The bound will be satisfiable after one cycle if either the chosen acceleration a already maintains speed limit bounds ($v \leq \mathsf{vh} \wedge v+aT \leq \mathsf{vh}$) or when there is enough distance left to restore the limits before reaching the goal. For straight line motion ($\mathsf{k} = 0$), the required distance can simply be found by integrating acceleration and speed:

$$\underbrace{vT + \frac{a}{2}T^2}_{\text{distance in time } T} + \frac{(\overbrace{v+aT}^{\text{speed in time } T})^2 - \mathsf{vh}^2}{2B} + \varepsilon \leq \|(\mathsf{x}, \mathsf{y})\|_\infty$$

where $a \in [-B, A]$. The extra factor of $(1 + |k|\varepsilon)^2$ for curved motion accounts for the fact that an arc along the inner side of the annulus is shorter than one along the outside (Fig. 4.1).

4.2.2 Proofs

In contrast to prior works (Mitsch et al., 2017), this work contains both safety and liveness proofs for the full 2D dynamics.[1]. In many ways, this lays a foundation for the proposed work because a proof of winnability for a hybrid game will contain both liveness-like (strategy for our player) and safety-like components (strategy for the opponent). Taken at surface value, the safety theorem says that speed limits are obeyed whenever the robot reaches a waypoint

[1]In the interest of full disclosure, the modeling and proof sections were largely performed by collaborators in this work. The present author provided simulations, writing, integration with VeriPhy and advice on the modeling and proof tasks. In contrast, the modeling and proofs in the proposed work are to be performed by the present author

Theorem 1 (Safety)**.** *The following* dL *formula is valid:*

$$A{>}0 \land B{>}0 \land T{>}0 \land \varepsilon{>}0 \land J \rightarrow$$
$$[(\mathit{ctrl};\mathit{plant})^*]\big(\|(\mathsf{x},\mathsf{y})\| \leq \varepsilon \rightarrow v \in [\mathsf{vl},\mathsf{vh}]\big)$$

Taken more broadly, safety also includes the fact that the robot always remains within an ϵ annulus around the arc leading to the waypoint, which arises as part of the loop invariant used to prove Theorem 1. The first four assumptions ($A{>}0 \land \cdots \land \varepsilon{>}0$) are basic sign conditions on the symbolic constants used in the model. The final assumption, J, is the loop invariant. The full definition of J is deferred to Section 4.2. We write $\|(\mathsf{x},\mathsf{y})\|$ for the Euclidean norm $\sqrt{\mathsf{x}^2+\mathsf{y}^2}$ and consider the robot "close enough" to the waypoint when $\|(\mathsf{x},\mathsf{y})\| \leq \varepsilon$ for our chosen goal size ε. While this captures the desired notion of safety, it does not prove that the robot can actually reach the goal, which is a *liveness* property:

Theorem 2 (Liveness)**.** *The following* dL *formula is valid:*

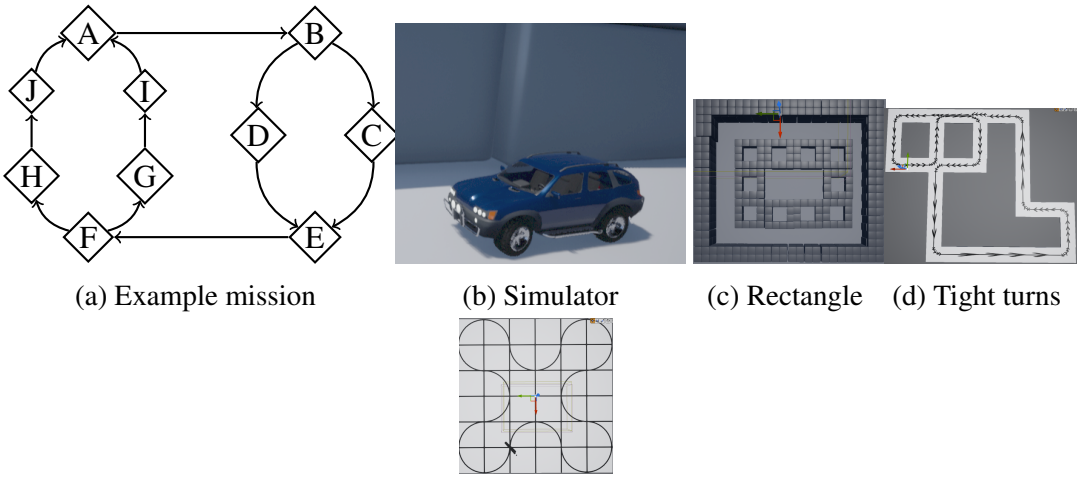
$$A{>}0 \land B{>}0 \land T{>}0 \land \varepsilon{>}0 \land J \rightarrow$$
$$[(\mathit{ctrl};\mathit{plant})^*]\Big(v{>}0 \land \mathsf{y}{>}0 \rightarrow$$
$$\langle(\mathit{ctrl}_a;\mathit{plant})^*\rangle\big(\|(\mathsf{x},\mathsf{y})\| \leq \varepsilon \ \land\ v \in [\mathsf{vl},\mathsf{vh}]\big)\Big)$$

This theorem has the same assumptions as Theorem 1. It says that no matter how long the robot has been running ($[(\mathsf{ctrl};\mathsf{plant})^*]$), then if some simplifying assumptions still hold ($v{>}0 \land \mathsf{y}{>}0$) the controller can be continually run ($\langle(\mathsf{ctrl}_\mathrm{a};\mathsf{plant})^*\rangle$) with admissible acceleration choices (ctrl_a) to reach the present goal ($\|(\mathsf{x},\mathsf{y})\| \leq \varepsilon$) within the desired speed limits ($v \in [\mathsf{vl},\mathsf{vh}]$). The simplifying assumptions $v{>}0 \land \mathsf{y}{>}0$ say the robot is still moving forward and the waypoint is still in the upper half-plane, i.e., we have not run *past* the waypoint.

The liveness theorem is invaluable because it validates the dL model: if it were not even possible at the level of the dL model to achieve liveness, then it would be absolutely impossible at the level of implementation to achieve liveness. Because it is undesirable to change the model drastically once implementation has started (i.e., delays in modeling and verification will trickle down to greater delays in implementation and testing), it is valuable to have this confidence in the model before implementation even begins. Moreover, the combination of safety and liveness gives us confidence that a game generalization of this proof will succeed, as the main components of a game proof are liveness and safety reasoning.

4.2.3 Simulations

In order to claim that VeriPhy has been evaluated on a realistic system implementation, it is essential that we choose a simulation platform that is reasonably faithful to the physics of actual autonomous cars. For this purpose we chose the AirSim (Shah et al., 2018) simulator, which is notable for combining top-notch visuals, reasonable physics, and an open-source code base that is particularly friendly to modification. While some aspects of the underlying physics engine are closed-source, many aspects are open to customization, including for example the details of wheels and suspensions, and the center of gravity. For this reason and because AirSim has been



(a) Example mission (b) Simulator (c) Rectangle (d) Tight turns

(e) Large clover

Figure 4.2: Implementation and environments built in AirSim

Table 4.1: Average speed, Monitor failure rates, plant violation rates, for AirSim and human driver in Rectangle, Turns, and Clover for Patrol missions

	Avg. Speed (m/s)					Ctrl Fail.					Plant		
World	BB	**PD1**	PD2	PD3	Human	BB	**PD1**	PD2	PD3	Human	BB	**PD1**	PD2
Rect	9.2	**6.24**	8.47	13.5	12.6	1.4%	**0%**	0%	0.17%	3.0%	9.8%	**1.8%**	1.2%
Turns	8.3	**4.54**	5.93	10.4	10.8	3.0%	**0%**	3.3%	0.75%	3.5%	11%	**0.2%**	0.4%
Clover	13.8	**13.9**	18.2	29.8	29.3	6.6%	**0.3%**	0%	0.32%	0.66%	28%	**1.7%**	44%

successfully applied in dozens of projects, we have good reason to believe its physics model is more faithful than any purpose-built one-off simulation would be.

The author implemented several bang-bang and PD (proportional-derivative) controllers in AirSim as well as several environments which provided a variety of driving conditions with turns of different radii. The environments and the evaluation results are summarized below:

Table 4.1 gives the results of several controllers (PD1 is the slowest PD controller, PD3 the fastest) as well as a human pilot (the author) on all environments. Overall, the slow PD controller had the best monitor failure rates of any controller. Qualitatively, the results tell us that while perfect failure 0% rates are not always attainable, low failure rates are achievable in practice, where low control failure rates mean that the fallback controller will rarely need to engage and low plant failure rates mean that because physical assumptions are rarely violated, the formal guarantees provided by VeriPhy are applicable almost all the time.

Aside from the concrete numbers provided by the simulations, perhaps the most important conclusion is that such a simulation could be developed and monitored in the first place. Throughout the development of this section, many iterations of the models, proofs, and simulations were developed: in order to make the proofs feasible, the models would be simplified, then when monitor failures were detected, the models would be relaxed to improve the failure rate. Low failure rates not only speak to the success of the implementation, but they say even more

about the success of the model: any time the model were too restrictive to account for the values given by the AirSim sensors and controllers, the monitor would have failed. The fact that they fail rarely attests that the model can account for the control decisions and physical dynamics that arise in practice. Because it was not until this chapter that a completely formal link between dL and CPS execution had been developed, this itself was long a point of curiosity.

4.2.4 Related Work

Related work in formal methods and robotics has applied synthesis and verification to safe control of robotic systems. The present work is unique among these in its use of a verified-safe sandbox controller to enforce compliance between the implementation and formal model for a realistic robot simulation with an expressive correctness guarantee.

Simulation

Simulation is an essential part of evaluating models and designs for any robotic system. Multiple simulation platforms are available, of which AirSim (Shah et al., 2018) is a recent platform for UAVs and autonomous cars. Other simulators would likely have worked as well, but we chose AirSim because it is configured with high-fidelity physical and visual models out of the box, reducing the chances of introducing modeling errors.

Chapter 5

Ongoing Work: CdGL: Constructive Differential Game Logic

Throughout this thesis, end-to-end verification is conceived as the problem of placing the correctness of realistic CPS on the most rigorous, formal logical foundation possible. The logic we use to verify a CPS model (thus far, dL) is then the centerpiece and our understanding of its foundations are of central importance. In Chapter 3, we started at the centerpiece and dug inward: we established a more rigorous foundation for dL as it is used in practice in the theorem prover by building a formally verified prover core and by providing a unified foundation for the grabbag of features that arise in practice. In Chapter 4, we started at the centerpiece and dug outward: model-driven monitor synthesis connects CPS implementations to their formal models; we showed that this connection can be made with full rigor for serious models and their implementations, placing our understanding of implementation-level safety on firm footing.

In the proposed work, the time has come to question the centerpiece itself. Vanilla dL has been used thus far in part because it comes with rich theoretical backing, a mature implementation, and prior cases studies for inspiration and comparison. However, as a logic, it hardly exists in isolation: A family of logics including QdL, SdL, and dGL have explored the addition of distributed, stochastic, and adversarial dynamics to hybrid systems, respectively. Further unexplored logical questions await: notably, all of the above logics are classical. When constructive logic is a key tool in software verification, we must earnestly ask what could be gained for end-to-end CPS verification by adopting constructive foundations. We must also ask which member of the dL family (which might not even be discovered yet) is best suited for end-to-end CPS verification.

This thesis views end-to-end verification as resolving the concerns of three entities: the logician concerned with rigorous foundations, the practitioner concerned with software artifacts resulting from verification, and the logic-user, concerned primarily with simplicity and readability of their models and the ease of their proofs. We have already satisfied the logician: we have made dL proofs in KeYmaera X indisputable. However, we argue that constructivity and games both have crucial advantages for the practitioner and logic-user. Thus, a constructive games logic (CdGL) is the optimal foundation to resolve the concerns of all three.

The completed sections of the thesis have revealed several weaknesses from the practitioner's perspective:

- The state-of-the-art supports only synthesis of control and plant monitors, not controllers nor plant simulations. The programmer might be content to write the controller themselves, but they are less content to prove for themselves that it always satisfies the monitor. For the programmer, the CPS has not been proven correct unless they know the monitor will be satisfied! Plant simulations are also valuable because they provide a complementary way to evaluate and understand a model, which is more familiar to most programmers.
- The state-of-the-art in synthesis lacks robustness, often restricting the user to special cases and failing to synthesize otherwise. Important control paradigms such as model-predictive control do not fall within the supported special cases, nor do any controllers containing loops. The ModelPlex synthesis approach started from the assumption that the model alone is sufficient, which was then falsified by the need to handle complex models, especially those whose plants are not nilpotent. The resulting approach scrapes invariant information from proofs in an ad-hoc fashion to aid synthesis. While the invariant information is essential, the ad-hoc nature stands in the way of robustness.
- The synthesized monitors are not tight. The arithmetic approximations used in Section 4.1 are conservative, and in practice can be prohibitively so, to the extent that significant cleverness is exerted in tweaking units of measure to avoid overflow errors. The implementation of arithmetic in synthesis is a fundamental question, but the present answer of integer interval arithmetic demands too much of the programmer.

All these concerns are addressed by the combination of games and constructivity:

- The main challenge in synthesizing a controller is deciding which control path to take (for example, acceleration vs. braking) when multiple options are plausible. All such decisions are captured by a proof of the shape $\langle \mathsf{ctrl} \rangle \phi$: in proving a diamond property, we have specified which path to take in order to achieve ϕ. Such synthesis is only possible in general when the proof is constructive: the state-of-the-art proof technique for such properties in fact applies the law of excluded middle to undecidable propositions, so constructivity is anything but a theoretical concern. In this chapter, we will incorporate an alternative proof method only case-analyzes decidable properties so that synthesis is possible. Games also play a part in controller synthesis: A typical game proof supports a diamond proof for control and a box proof for plant, making it simple to synthesize both a controller and monitor together.
- Robustness issues in the state-of-the-art are simply a side effect of the fact that there is no proof representation available that closely corresponds to intuitive notions of control and monitoring. While proof representations are available, including one developed by the author, these representations use a low-level classical Hilbert calculus with no obvious computational interpretation and no obvious correspondence to intuitive "human-level" proofs. Developing a proof-theoretic semantics with a computational interpretation of proof terms is par for the course in constructive logic. By developing such semantics for CdGL, we will lay the foundation for robust implementation of proof-driven synthesis.
- Conservativity is a consequence of the chosen arithmetic implementation: fixed-precision interval arithmetic. Better libraries might alleviate the practical concern, but constructive mathematics provides a more general answer: computable reals. Constructive proofs in

CdGL have the convenient property that whenever a proof of $\langle \mathsf{ctrl} \rangle \phi$ exists, a controller exists which uses only *total* computations on computable reals (e.g. the controller will never attempt to compare two reals for exact equality). Certainly interval arithmetic has its own advantages in performance, but with computable reals we resolve conservativity issues conclusively.

For the logic-user:

- Short, simple models are desirable. Longer models are more likely to contain mistakes, and modeling mistakes are the hardest kind to catch as we cannot simply "prove" their presence or absence. As Section 4.2.1 and Example 3 show, hybrid systems models are typically much more complicated than their hybrid game counterpart, because strategic choices must be modeled in the hybrid system which in games are deferred to the proof. Thus, games generally (dGL or CdGL) eliminate a prominent source of needless model complexity.
- Beyond reducing complexity, games models preclude common mistakes which have led even experienced researchers to write vacuous models in practice. Specifically, in a standard dL safety proof, safety is proven only for states from which at least one guard of the controller is satisfied. An intuitive notion of safety requires that at least one guard is satisfied in each state (the controller is *total*), which is often false when models are not written carefully. In games, we can (and by default do) require that controllers satisfy *co-safety*, which implies any faulty, non-total controller will be caught when at proof time since a co-safety proof will necessarily fail. Catching such mistakes is important because otherwise the logic-user, having proved a vacuous theorem, would assume their system is safe when in fact *nothing* of note has been proven.
- For many programmers, formal proofs are more intimidating than code and thus possess a steeper learning curve. While not a focus of the present thesis, a computational interpretation opens up the possibility of writing CdGL proofs by writing the code that implements a winning strategy. We suggest that such an approach would ease adoption of proofs for CPS.

Lastly, even though the logician's foundational concerns have been adequately addressed in the previous chapters, there are independent motivations for CdGL from a foundational, theoretical perspective:

- As discussed in related work, constructive program logics are a nearly-unexplored topic, and even constructive modal logic generally is a lightly explored field. CdGL is not only a generalization of dGL and dL, but also of (Harel's) First-order Dynamic Logic and (Parikh's) Game Logic, which itself generalizes (Pratt's) PDL and (Peleg's) Concurrent Dynamic Logic. In developing CdGL, we indirectly develop the first serious constructive treatment for all the above logics as well, specifically the first to feature a full proof-theoretic treatment based on proof terms and their normalization. It is also the first treatment of any of the above to support standard constructive properties such as the Existential Property
- The Curry-Howard correspondence, wherein propositions are interpreted types and proofs are interpreted as programs, has for many years been the backbone of cross-pollination be-

tween programming languages and logic research. CdGL gives the first understanding of Curry-Howard for game logics or any dynamic logic: Proofs of $\langle\alpha\rangle\phi$ and $[\alpha]\phi$ are understood computationally as winning strategies, for the P-player and N-player, respectively. In the special case, they are understood as *controlling* program α to achieve ϕ or *monitoring* an execution for compliance with α, which ensures ϕ as postcondition. This substantiates the claim that such proofs suffice for synthesis, and we posit that this correspondence is applicable to *any* nondeterministic program logic. [1]

5.1 Related Work

Several approaches have been taken to developing intuitionistic and constructive modal logics, including dynamic logics. What unequivocally sets us apart from these works is that we are the first to develop a proofs-as-programs interpretation, e.g. we develop a language of proof terms and study its operational semantics in full. Only a few prior works suggest an explicit representation of strategies, and they do not develop an operation semantics for strategies, let alone prove its properties.

5.1.1 Semantics for Constructive Modal Logics

We discuss several general approaches for semantics of constructive modal logics. The main approaches of interest to us are intuitionistic Kripke semantics and realizability semantics.

Intuitionistic Kripke semantics are perhaps the most widely-used approach, an overview is given in (Wijesekera, 1990). Like classical Kripke semantics, their intuitionistic counterparts are parameterized over worlds, except the worlds represent what is currently *known* about the world. Intuitionistic Kripke semantics are also equipped with a preorder $w_1 \geq w_2$ which holds when w_1 contains all the knowledge of w_2 and possibly more. This technique can be (and has been) applied to dynamic logics, with the twist that both the world per se and our knowledge about the world can change throughout executions.

Realizability semantics (van Oosten, 2002; Lipton, 1992) are particularly appealing to the computer scientist, because they give the semantics in terms of a *realizer*, a program which performs the proof. Such semantics are usually given abstractly, i.e., they do not specify a particular programming language, but the standard notion of computability is understood. Our realizability semantics are then more concrete than most, because they fall out of the operational semantics on strategy proof terms. A traditional abstract reachability semantics can then serve as a middle step connecting the operational semantics to the model-theoretic semantics.

Type theory has been used as a semantics of constructive ZF set theory (Aczel & Gambino, 2006). Topological models have been used, especially for modal logics of space (van Benthem & Bezhanishvili, 2007). While CdGL is not intended as a spatial logic, it does operate over a concrete topological space. A close relative of topological models are neighborhood models (van Benthem et al., 2017). Intuitionistic Kripke semantics for modal logic have been compared to a categorical semantics (Alechina et al., 2001). Sheaf-theoretic models of first-order modal logic

[1] One could also apply this interpretation to deterministic logics, but they lack our motivations of synthesis and monitoring, which is perhaps the reason this interpretation never developed sooner

have been given (Hilken & Rydeheard, n.d.), which provide a general treatment of the idea that an individual might change across states, as well as presheaf (Ghilardi, 1989) models. For (classical) PDL, stochastic Kripke semantics have been compared to coalgebra semantics (Doberkat, 2011).

5.1.2 Specific Constructive Modal Logics

We discuss a variety of constructive modal logics which have been developed, including treatments of PDL and Concurrent DL. Among intuitionistic dynamic logics, dynamic-epistemic logic (DEL) (Frittella et al., 2016) possesses a well-understood proof-theoretic semantics, but this semantics does not tell the whole story, since it neither exposes the underlying computational interpretation nor has its relation to a model-theoretic semantics been explored. While an argument can made in the case of DEL to take the proof-theoretic semantics as primary, model-theoretic semantics are an essential part of the story for CdGL, because the physical world and the real numbers which represent it are conceptually prior to proof. Proof-theoretic semantics for constructive, paraconsistent LTL have also been given (Kamide & Wansing, 2010).

Constructive tableaux have been developed for Concurrent Dynamic Logic (Wijesekera & Nerode, 2005), which is a special case of CdGL. However, only an intuitionistic Kripke semantics are given, and a proofs-as-programs interpretation is not developed.

Arguably, the closest paper in spirit is (Degen & Werner, 2006), because they pursue an intuitionistic PDL which satisfies an existence property (EP) for programs, which is an important step in our development of a proofs-as-programs interpretation. However, they stop short of full EP and show a special case, and question the usability of their own logic since their implication introduction rule has an nonsyntactic side condition. Moreover, they suggest that the first-order case (which we support) is beyond the methods of their day.

Intuitionistic Kripke semantics have also been given for a fragment of PDL (Celani, 2001), specifically multimodal System K with an iteration modality.

5.1.3 Game Logic

First off, all previous logical explorations of hybrid games (Quesel & Platzer, 2012; Platzer, 2015, 2017b, 2018b) have explored only the classical case. Explicit strategies for GL were proposed (Ghosh, 2008). The logic developed therein is suspect because "a game is winnable" and "a game is winnable by a given strategy" are defined as entirely different formulas, when they ought to be intimately connected.

Another explicit treatment of strategies is given in (Ramanujam & Simon, 2008), but their presentation is also questionable because arbitrary strategies are only allowed for the propositional games, which appears to exclude most interesting strategies for repeated games.

The prior works only treat Parikh's propositional GL, and moreover the connection between strategies as proof terms and constructivity was not made, so no proof theory was developed, in contrast to our work.

van Benthem has also written on strategies in logic (van Benthem, 2015), but his exploration is in the context of dynamic-epistemic (van Benthem et al., 2011; Van Benthem, 2001) and evidence logics (van Benthem & Pacuit, 2011), not GL per se.

5.1.4 Constructive Analysis

Constructive real analysis has been studied in depth, and is essential to the success of CdGL: computationally interpreting first-order real arithmetic and differential equation reasoning will require a solid foundation in constructive analysis.

The foundational work on constructive analysis is Bishop (Bishop, 1967), and a more recent in-depth treatment is given by Bridges (Bridges & Vita, 2007). A realizability interpretation of Bishop's analysis has been explored (Schwichtenberg, 2008). We are particularly interested in the constructive version of the Picard-Lindelöf theorem, which has been proven previously in Coq (Makarov & Spitters, 2013).

Constructive analysis typically deals with the true reals, defined constructively: i.e., there are uncountably many reals in constructive analysis. Closely related are the *computable reals*: the countable subset of the reals which have an actual implementation as a computer program. The computable reals are traditionally modeled by the Baire space (Scott, 1968). In the semantics of *hybrid games*, we simply employ the continuum as a given. What is essential, however, is to ensure that computable reals are an adequate model for *strategies*, i.e., no strategy should be allowed to exploit an operation that is not decidable on the computable reals. As we move toward implementation, it will be important to have access to an efficient implementation of computable reals. One such implementation is CORN (Krebbers & Spitters, 2011), which includes a constructive version of Rolle's Theorem/Mean Value Theorem, another key component for CdGL.

5.2 Syntax

The term language of CdGL is that of dL:

$$\theta ::= q \mid x \mid \theta + \eta \mid \theta \cdot \theta \mid (\theta)'$$

where $q \in \mathbb{Q}$ is a literal, x a variable, $\theta + \eta$ a sum, $\theta \cdot \eta$ a product and $(\theta)'$ a derivative.

The formulas of CdGL are:

$$\phi ::= \theta > \eta \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \to \psi \mid \bot \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid \forall x \phi \mid \exists x \phi$$

where $\theta > \eta$ is true when θ is greater than η, $\phi \wedge \psi$ is true when ϕ and ψ are both true, $\phi \vee \psi$ is true when either ϕ or ψ is true (and we know which one), $\phi \to \psi$ is true when ψ is true under the assumption that ϕ is true, and $\bot$ is not true. $\langle\alpha\rangle\phi$ is true when Angel has a strategy for α to reach the region where ϕ is true (and we know which strategy, and we know why ϕ is true in the end). Likewise, $[\alpha]\phi$ is true when Demon has a strategy for α to reach the region where ϕ is true. $\forall x\, \phi$ is true when ϕ is true under the assumption that x is a fresh (real-valued) variable. Likewise, $\exists x\, \phi$ is true when ϕ is true under some assignment of x (and we know which one).

Since CdGL is a constructive logic, classical duality axioms such as $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg psi)$ do not hold, and thus none of the above constructs are derived. Even in a constructive logic, however, the constructs $\neg\phi \equiv (\phi \to \bot), \phi \leftrightarrow \psi \equiv (\phi \to \psi \wedge \psi \to \phi)$, and $\theta \geq \eta \equiv \neg(\eta > \theta)$ are derived. The equivalence $\theta \geq \eta \equiv \neg(\eta > \theta)$ is standard in constructive analysis, but nonstrict inequalities will merit special attention since equality on reals is undecidable, and thus nonstrict

inequalities are as well. For example, the trichotomy axiom of classical analysis:

$$\theta < \eta \vee \theta = \eta \vee \theta > \eta$$

is false constructively, as it is undecidable which of three cases holds.

In CdGL, hybrid games follow the same syntax as in dGL, except that all formulas appearing in games are CdGL formulas.

5.3 Example System

As a case study for CdGL, we propose reproducing the 2D planar robot of Section 4.2 as a hybrid game, and providing a constructive proof that the robot can reach its waypoints while maintaining its speed limits. In order to assess the feasibility of this task early, without access to an implementation of CdGL, we propose performing the proof in classical dGL in KeYmaera X and inspecting by hand that the proof uses only constructive reasoning. We give the hybrid game model here. The plant is identical to that of Section 4.2: the robot is fixed at the origin facing toward the +y axis, while the waypoint moves relative to it:

$$\begin{aligned}\mathsf{plant} \equiv \{&\mathsf{x}' = -v\,\mathsf{k}\,\mathsf{y},\ \mathsf{y}' = v\,(\mathsf{k}\,\mathsf{x} - 1),\ v' = a,\ \mathsf{t}' = 1\\ &\&\ \mathsf{t} \leq T\ \wedge\ v \geq 0\}\end{aligned}$$

The conditions Ann and Feas, which say the waypoint and robot are on the specified annulus, and that the waypoint is achievable, respectively, are also as before:

$$\begin{aligned}\mathsf{Ann}\ &\equiv\ |\mathsf{k}|\varepsilon \leq 1 \wedge \left|\frac{\mathsf{k}\ (\mathsf{x}^2 + \mathsf{y}^2 - \varepsilon^2)}{2} - \mathsf{x}\right| < \varepsilon\\ \mathsf{Feas}\ &\equiv \mathsf{Ann} \wedge \mathsf{y}{>}0 \wedge 0{\leq}\mathsf{vl}{<}\mathsf{vh} \wedge AT{\leq}\mathsf{vh} - \mathsf{vl} \wedge BT{\leq}\mathsf{vh} - \mathsf{vl}\end{aligned}$$

The main change is in the controller. In the systems setting, the controller needed to express a condition Go which specified the allowable accelerations in each case. In the games setting, the acceleration is existentially quantified in the model, and choices of acceleration are merely part of the proof:

$$\mathsf{ctrl}\ \equiv\ ((\mathsf{x},\mathsf{y}) := *;\ [\mathsf{vl},\mathsf{vh}] := *;\ \mathsf{k} := *;\ ?\mathsf{Feas})^d;\ a := *$$

We propose to prove a winnability theorem saying the robot can always control its way to the goal region within its speed limits.

Theorem 3 (Angel Winnability)**.** *The following* dL *formula is valid:*

$$\begin{aligned}&A{>}0 \wedge B{>}0 \wedge T{>}0 \wedge \varepsilon{>}0 \wedge J \rightarrow\\ &\langle(\mathit{ctrl};\mathit{plant}^d)^*\rangle\big(\|(\mathsf{x},\mathsf{y})\| \leq \varepsilon \wedge v \in [\mathsf{vl},\mathsf{vh}]\big)\end{aligned}$$

where J is the loop invariant as defined in Section 4.2.

In this form, Theorem 3 corresponds to the liveness, Theorem 2. However, we propose also to generalize Theorem 3 slightly, thus proving Theorem 2 and Theorem 1 (safety) in one go.

Theorem 4 (Generalized Angel Winnability)**.** *The following* dL *formula is valid:*

$$A{>}0 \land B{>}0 \land T{>}0 \land \varepsilon{>}0 \land J \rightarrow$$
$$\langle (\mathit{ctrl}; \mathit{plant}^d; ?J)^* \rangle \big(\|(x, y)\| \leq \varepsilon \land v \in [\mathit{vl}, \mathit{vh}] \big)$$

where J is the loop invariant as defined in Section 4.2.

That is, after Demon chooses the duration of the plant, Angel must prove that the loop invariant J still holds. Recall that the invariant J includes as conjuncts the facts that the robot is on the annulus and has sufficient distance remaining to achieve its speed limit. To win the generalized game, Angel must be able to prove that it is on track to the waypoint (both in position and velocity) at *whatever* point Demon chooses, thus at every one of the uncountably many points in time. This is exactly what safety ought to constitute. Because we simply added a test to the previous model, whenever Angel wins the modified game they have still achieved liveness. It is simply harder now, as they must never violate safety in the process of achieving liveness.

The example demonstrates several advantages of hybrid games proving generally over hybrid systems. Not only is the model shorter, waypoint-following exhibits turn-taking behavior which is challenging to characterize in a hybrid system. The Angel player must follow an arbitrary waypoint, i.e., one chosen by Demon. In order to make victory for Angel even possible, the responsibility is Demon's to choose a waypoint that is feasible. Expressing this turn-taking in Theorem **??** is cumbersome, while in Theorem 4, it falls naturally out of the fact that acceleration is controlled by Angel while all else is up to Demon. Lastly, we have the advantage that safety and liveness need not be proved separately, but can be combined as a single winnability theorem. This obviously saves time and energy, but there is a conceptual advantage as well: in the systems setting, we must prove *all* options are safe and that *one* is live. In the games setting, we can simply prove that *one* strategy is safe-and-live, so for states our strategy will never enter, we need not even consider safety.

The winnability approach ought to apply to any *reach-avoid* problem for a control-plant loop, i.e., any theorem of form:

$$\text{pre} \rightarrow \langle (\mathsf{ctrl};\ (\mathsf{plant})^d;\ ?\text{safe})^* \rangle \text{live}$$

Within the hybrid systems literature, and certainly within the dL literature, most systems can be expressed as control-plant loops, and the most important properties are reach-avoid properties or special cases thereof. Thus we are justified to claim that a wide array of verification tasks would benefit from the simplicity provided by modeling and proving with games.

5.4 Semantics

Several styles of semantics are widely used for constructive logics, and each has its own advantages. We present several semantics here, since the remaining tasks (proof term semantics, theoretical results) will inform the choice between them.

5.4.1 Real Constructive Games, One Realizer

The semantics given here can be thought of as the "halfway-point" between fully Kripke-style or fully realizability-style semantics. Instead of giving explicit realizers for both players' strategies, we give a realizer just for the Angel player, and quantify over Demon's decisions locally. The benefits of this presentation are that it's clear Demon's strategy is being treated extensionally, i.e., Angel's strategy never gets to inspect Demon's strategy, and Angel's strategy works even if Demon employs noncomputable trickery. The local treatment may also simplify soundness proofs and certainly simplifies notation a bit.

$$\begin{aligned}
[\![\theta > \eta]\!] &= \{\omega \mid [\![\theta]\!]\omega > [\![\eta]\!]\omega\} \\
[\![\phi \wedge \psi]\!] &= [\![\phi]\!] \cap [\![\psi]\!] \\
[\![\phi \vee \psi]\!] &= \bigcup_a [\![\phi]\!] \cup_a [\![\psi]\!] \\
[\![\phi \to \psi]\!] &= \bigcup \{Z \subseteq \mathcal{S} \mid Z \subseteq [\![\phi]\!] \text{ constructively implies } Z \subseteq [\![\psi]\!]\} \\
[\![\forall x \phi]\!] &= \{\omega \mid \text{for all } r \in \mathbb{R}, \omega_x^r \in [\![\phi]\!]\} \\
[\![\exists x \phi]\!] &= \bigcup_{f:\mathcal{S}\rightharpoonup\mathbb{R}} \{\omega \mid \omega_x^{f(\omega)} \in [\![\phi]\!]\} \\
[\![\bot]\!] &= \emptyset \\
[\![\langle\alpha\rangle\phi]\!] &= \varsigma_\alpha([\![\phi]\!]) \\
[\![[\alpha]\phi]\!] &= \delta_\alpha([\![\phi]\!])
\end{aligned}$$

We elaborate on the formula semantics. The discriminated union $A \cup_a B$ exists for a given relation a (or we say a is suitable for $A \cup_a B$) iff and it only ever discriminates an element into a set where it is actually a member, i.e., $(x, 0) \in a \to x \in A$ and $(y, 1) \in a \to x \in B$. Then $A \cup_a B = \{x \mid \exists y\ (x, y) \in a, x \in A \cup B\}$. That is, not every member of $A \cup B$ must be sorted into one of the sets, but the discriminated union consists of those elements that can be so sorted. The meaning of $\phi \vee \psi$ is then the union of all such discriminated unions. This can be smaller than the classical set-theoretic union, e.g., $x < 0 \vee x \geq 0$ is classically a tautology but constructively holds everywhere but 0. This is notably more complicated a definition than even the typical constructive definition, and with good reason. Unlike many constructive logics, we assume that we are entitled to inspect the global state in a proof, just that we are not entitled to any magical, noncomputable inspections. The discriminator a allows us to capture the notion of which inspections are considered computable. The use of a is philosophically analogous to the use of a collection axiom in constructive ZF set theory to form proper functions from multi-valued functions.

The definition of $[\![\phi \to \psi]\!]$ also deserves special attention. The simpler alternative definition $[\![\phi \to \psi]\!] = [\![\psi]\!] \cup [\![\phi]\!]^{\complement}$ ought to work just fine, but I am hesitant to use it as it hides important subtleties. Firstly, there are at least three sometimes-different notions of complement $S^{\complement}$ in common use in constructive analysis: logical complement defined by contradiction $[\![x \in S \to \bot]\!]$, metric negation $\{x \mid \forall y \in S, \delta(x, y) > 0\}$, and plain old "complement" $\{x \mid \forall y \in S, x \neq y\}$. We risk causing confusion between the three, and also confusion over the fact that the union is

classical, not constructive. Lastly, union and complement do not correspond clearly to standard proof methods for semantically proving implications, while our awkward-looking definition is quite simple for use in semantic proofs. The definition of implication reads best in the epistemic understanding of intuitionistic logic: No matter how much I know about the current state, if it's enough to know ϕ then it has to be enough to also know ψ. As is typical in the semantics of constructive logic, the notion "constructively implies" is left vague-sounding. However, it really just comes down to the rule (as followed in all the other semantic cases) that the state may be inspected only with computable discriminators.

Now $\varsigma^a_\alpha(X)$ is the winning region from which Angel will reach goal region X by following strategy a, while $\delta^a_\alpha(X)$ is the winning region from which Demon will reach goal region X assuming *Angel* follows a.

$$
\begin{aligned}
\varsigma^a_{?\phi}(X) &= [\![\phi]\!] \cap X \\
\varsigma^a_{x:=\theta}(X) &= \{\omega \mid \omega^{[\![\theta]\!]\omega}_x \in X\} \\
\varsigma^a_{x:=*}(X) &= \{\omega \mid \omega^{a(\omega)}_x \in X\} \\
\varsigma^a_{\alpha;\beta}(X) &= \varsigma^{a_0}_\alpha(\varsigma^{a_1}_\beta(X)) \\
\varsigma^a_{\alpha\cup\beta}(X) &= \varsigma^{a_1}_\alpha(X) \cup_{a_0} \varsigma^{a_1}_\beta(X) \\
\varsigma^a_{\alpha^*}(X) &= \bigcap\{Z \subseteq S \mid X \cup_{a_0} \varsigma^{a_1}_\alpha(Z) \subseteq Z\} \\
\varsigma^a_{\alpha^d}(X) &= \delta^a_\alpha(X) \\
\delta^a_{?\phi}(X) &= [\![\phi]\!]^{\complement} \cup X \\
\delta^a_{x:=\theta}(X) &= \{\omega \mid \omega^{[\![\theta]\!]\omega}_x \in X\} \\
\delta^a_{x:=*}(X) &= \{\omega \mid \omega^r_x \in X\} && \text{for all } r \in \mathbb{R} \\
\delta^a_{\alpha;\beta}(X) &= \delta^{a_0}_\alpha(\delta^{a_1}_\beta(X)) \\
\delta^a_{\alpha\cup\beta}(X) &= \delta^{a_0}_\alpha(X) \cap \delta^{a_1}_\beta(X) \\
\delta^a_{\alpha^d}(X) &= \varsigma^a_\alpha(X) \\
\delta^a_{\alpha^*}(X) &= \bigcup\{Z \subseteq S \mid Z \subseteq X \cap \delta^a_\alpha(Z)\}
\end{aligned}
$$

where $A \cup_a B \equiv \{\omega \mid (a(\omega) = 0 \text{ and } \omega \in A) \text{ or } (a(\omega) = 1 \text{ and } \omega \in B)\}$ should be understood as a constructive union of $A \cup B$ where the realizer function a is used to decide whether element ω should seek membership in A vs B. In $\varsigma^a_{\alpha;\beta}(X)$ (and $\delta^a_{\alpha;\beta}(X)$), distinct strategies a_0 and a_1 may be used for α and β. In $\varsigma^a_{\alpha\cup\beta}(X)$, the realizer a_0 is used to decide whether to play α or β, then the remaining strategy a_1 is used to play the remaining game. It need not be different strategies for α vs. β since a_1 could just invoke a_0 to decide which branch had been taken. In $\varsigma^a_{\alpha^*}(X)$, the realizer a_0 is used to decide whether to terminate the loop, then a_1 is used to play the loop body. Not every syntactically valid strategy is sensible. We only consider a strategy to be well-formed if every $A \cup_a B$ satisfies the side conditions that $a(\omega) = 0$ implies $\omega \in A$ and $a(\omega) = 1$ for all ω. Because a must be computable, this will also require that A and B are either separated by some distance $\epsilon > 0$ or when they overlap, always overlap by some distance $\epsilon > 0$, i.e., they never meet exactly at a point. In the case $\delta^a_{\alpha^*}(X)$, a strategy is additionally only well-formed if it is "well-founded," meaning it will lead the loop to terminate. Specifically, a is well-founded (with

respect to X and α) if there is a computable termination metric $\mathcal{M}$ such that $a(\omega) = \mathcal{M} = 0$ on $\omega \in X$ and $\sup_{\omega \in Y} \mathcal{M}(\omega) < \sup \omega \in \delta^a_\alpha(Y)$ for all Y, where $<$ is the ordering on the metric. Given the nature of computable union, it is worth noting that this loop semantics does not always stop as early as possible: if we have just barely entered the edge of the goal region, the strategy might detect this as being outside the union, in which case we repeat the loop and then enter further into the goal region, which ought to terminate the loop, just one step too late.

Then the semantics for $[\alpha]\phi$ and $\langle\alpha\rangle\phi$ are defined as:

$$[\![[\alpha]\phi]\!] = \bigcup_a \delta^a_\alpha([\![\phi]\!])$$

$$[\![\langle\alpha\rangle\phi]\!] = \bigcup_a \varsigma^a_\alpha([\![\phi]\!])$$

The classical union over all a is justified by the fact that there are countably many strategies and it's decidable whether a given strategy wins a given game, so if a winning strategy does exist, Angel would eventually find it by an exhaustive search. We never test for the absence of such a strategy, so it is of no concern that a search would loop in that case.

In $\delta^a_{?\phi}(X)$, Angel can present us a proof of ϕ, and if so we win if we already in the goal region. The set complement in that case is classical: anywhere outside of $[\![\phi]\!]$, Angel would fail to present a proof. Unlike Angel, Demon is entitled to use noncomputable magic in its strategy, so $\delta^a_{\alpha\cup\beta}(X)$ and $\delta^a_{\alpha^*}(X)$ use classical unions, not constructive unions.

5.5 Proof Calculus

As usual when developing a new logic, we will develop a proof calculus for CdGL and prove that the calculus is sound. Because a proof calculus already exists for classical dGL, we focus here on the considerations that are unique to constructive logics.

First and foremost, to substantiate the notion that proofs in CdGL correspond to winning strategies for games (or for any formula), we propose to augment each proof rule with explicit *proof terms* M. This allows us to explore the proof-theoretic semantics is detail simply by developing a structural operational semantics for the proof terms. For example, when proof terms satisfy progress and preservation, then every provable formula has a canonical (and cut-free) proof. Furthermore, by showing that the structural operation semantics are a concrete implementation of the realizability semantics, we give a synthesis algorithm for computable strategies from proofs.

Any proof system can be augmented with proof terms, but some systems yield simpler interpretations of proof terms as programs than others. Several proof calculi for dL have been developed, including sequent calculi and Hilbert systems. Two systems of proof terms have been developed for classical dL, but one uses Hilbert calculus alone and the other mixes Hilbert calculus with propositional sequent calculus. Hilbert systems can be modeled computationally by combinatory logic and sequent calculus can be modeled with continuation-passing style proof terms, but both are typically more complex than the lambda-calculi which serve as computational models of natural deduction calculi. For this reason, we propose developing a natural deduction

calculus for CdGL, in order to simplify the proof term language. For practical use, a sequent calculus can be implemented on top of the natural deduction calculus.

In this natural deduction system with proof terms, the main proof judgement is $\Gamma \vdash M : \phi$, which says term M is a proof of ϕ in the context Γ. The assumptions in the context are named, so that they may be mentioned in proof terms.

$$\frac{\Gamma, \mathcal{M} = x, 0 < \mathcal{M} \vdash N : \langle\alpha\rangle\mathcal{M} < x}{\Gamma \vdash \text{while}(\mathcal{M} > 0)\{N\} : \langle\alpha^*\rangle\phi}$$

where $\mathcal{M}$ is a termination metric, x is fresh. The con rule corresponds to the metric on reals where $x < y \equiv \lfloor x \rfloor < \lfloor y \rfloor$. For games, we often want to support the following inductively defined metrics (in terms of metrics M_1, M_2, N_1, N_2):

- Lexicographic product metric (M, N) where $0_{(M,N)} = (0_M, 0_N)$ and $(M_1, N_1) < (M_2, N_2)$ iff (1) $M_1 < M_2$ or (2) $M_1 = M_2$ and $N_1 < N_2$.
- The sum metric $M \vee N$ where $0_{M \vee N} = 0_N$ and $\ell \cdot M < r \cdot N$ always and $\ell \cdot M_1 < \ell \cdot M_2$ iff $M_1 < M_2$ in metric M and $r \cdot N_1 < r \cdot N_2$ iff $N_1 < N_2$ in metric N.

The sum metric is the natural metric on disjoint sums. It's useful for loops that follow one strategy first, then another strategy, and can replaced by $\langle\alpha^*\rangle\phi \leftrightarrow \langle\alpha^*\rangle\langle\alpha^*\rangle\phi$.

5.6 Operational Semantics

The design of the operational semantics is intimately related to what normal forms we want. In the proposed calculus, normal forms are as follows:

Lemma 5 (Normal Form Characterization)**.** *If $\Gamma \vdash M : \phi$ and M is normal, then the final step of proof M is either:*

- *A case-redex that inspects the state, e.g. case on split ϵ*
- *Monotonicity, or*
- *An introduction rule for the outermost connective of ϕ. In the case that ϕ is a program modality, it is specifically an introduction rule for the outermost program construct.*

Perhaps it is surprising that monotonicity has to be a normal form. This is the case because the assignment rule cannot always be applied left-to-right, and monotonicity allows applying it right-to-left when needed. The alternative is to introduce a second equality rule which uses renaming:

$$\frac{\Gamma^y_x, x = \theta^y_x \vdash M : \phi}{\Gamma \vdash y \text{ was } x \text{ in } M : [\alpha]\phi}$$

If this rule is introduced, we need not evaluate right-to-left, and so the theorem becomes:

Lemma 6 (Alternative Normal Form Characterization)**.** *If $\Gamma \vdash M : \phi$ and M is normal, then the final step of proof M is either:*

- *A case-redex that inspects the state, e.g. case on split ϵ, or*
- *An introduction rule for the outermost connective of ϕ. In the case that ϕ is a program modality, it is specifically an introduction rule for the outermost program construct.*

This rule raises suspicions, because it requires renaming the whole context and possibly remembering the entire execution history. However, forall already requires this and forall seems to work, so I'd be okay with using this rule.

We give the reduction rules for proof terms:

The reduction rules for $M\circ_x N$ and the argument for their normalization are subtle. The termination argument is by lexicographic induction on the postcondition formula ϕ and on the derivation M.

The structural cases are "obvious" and are listed here, where M normal says a proof term M is already in normal form.

5.7 Theory Results

Once the calculus has been written and proved sound, we will prove standard theoretical results. We list the results here and remark how they inform the design of the proof term language.

Lemma 7 (Progress)**.** *If $\Gamma \vdash M : \phi$, then either M is normal or $M \mapsto N$ for some N.*

Lemma 8 (Preservation)**.** *If $\Gamma \vdash M : \phi$ and $M \mapsto^* N$, then $\Gamma \vdash N : \phi$*

Lemma 9 (Normal Form Existence)**.** *If $\Gamma \vdash M : \phi$, there exists normal N such that $\Gamma \vdash N : \phi$.*

Lemma 10 (Term Existence Property)**.** *If $\Gamma \vdash M : \exists x \phi(x)$, then there exists proof N and term θ such that $\Gamma \vdash N : \phi(\theta)$.*

Sketch. AWLOG M is normal by Lemma 9. Then the outermost step is existential introduction, which contains the witness θ. □

While the proof of Lemma 10 is simple, the fact that Lemma 10 is even true reveals the importance of choosing an expressive term language. In CdGL, as in traditional dGL, all terms are polynomials. However, there are many existential formulas which are only true of individuals that are non-polynomial, for example:

$$\exists t \langle x := 1;\ t' = -1, x' = x\rangle (t = 0 \land x = y)$$

is true for $t = \ln y$, which is not polynomial. If the existential introduction rule only allows polynomials, it will surely be incomplete in this case. A natural solution is to extend the term language with constructs for integration and algebraic numbers, which regain completeness for differential equations and real arithmetic, respectively. This solution is also natural because proof reductions for differential equations and quantifier elimination, respectively, naturally correspond to integrals and real-algebraic numbers. Because these constructs are of relatively little use in source-level hybrid game models, we intentionally do not present them when presenting the term language. However, to make Lemma 10 totally rigorous, we will give them a semantics as all other terms have.

Another standard property is the Disjunction Property (DP), which does not hold in its naive form in CdGL

Lemma 11 (Naive DP)**.** *It is not the case that when $\Gamma \vdash M : \phi \lor \psi$ there always exists an N such that $\Gamma \vdash M : \phi$ or $\Gamma \vdash M : \psi$.*

Proof. Consider $\phi \equiv x < 1$ and $\psi \equiv x > 0$. Then $\Gamma \vdash M : \phi \vee \psi$ where M is the approximate splitting rule. But clearly $x < 1$ and $x > 0$ are not true in the empty context, let alone provable. □

It is for very good reason that Naive DP fails: unlike in many constructive logics, the players in CdGL are both entitled to inspect the state of the world: Angel uses the inexact comparisons of the approximate splitting rule, and Demon is even entitled to arbitrary inspection. Naive DP is also stronger that what we really need to know, which is that when $\Gamma \vdash M : \phi \vee \psi$, then we can decide which of ϕ or ψ is true. We prove this Weak DP:

Lemma 12 (Weak DP)**.** *When $\Gamma \vdash M : \phi \vee \psi$ there exists $f : \mathcal{S} \rightharpoonup \mathbb{B}$ such that for every state ω where $\omega \in [\![\phi \vee \psi]\!]$, if $f(\omega) = 0$ then $\omega \in [\![\phi]\!]$, else $f(\omega) = 1$ and $\omega \in [\![\psi]\!]$*

Proof. AWLOG M is normal, then it ends in disjunction introduction. The introduction rule contains f as a witness. □

Because CdGL is a game logic, it is natural to desire a counterpart to EP and DP for the games connectives. That is, in order to establish Curry-Howard for the diamond and box modalities, we should construct the winning strategies for each player given only the proofs. There is no standard approach because no similar result exists in the literature. I propose one approach here.

Lemma 13 (Game EP for Angel)**.** *If $\Gamma \vdash M : \langle\alpha\rangle\phi$, then for each state ω there exists constructively an Angel strategy a such that $\varsigma^\alpha(a)[\![\phi]\!]$.*

Lemma 14 (Game EP for Demon)**.** *If $\Gamma \vdash M : [\alpha]\phi$, then for each state ω there exists constructively a strategy a, $\delta^\alpha(a)[\![\phi]\!]$.*

This definition of EP for games is somewhat unsatisfying in that it relates the proof-theoretic semantics only to the realizability semantics, and not back to itself. However, the representation of strategies in the language of hybrid games is a non-trivial question itself, so it is not clear that expressing the existence of strategies proof-theoretically is any better.

5.7.1 Arithmetic Proofs

CdGL accepts constructive analysis as its foundation instead of the classical analysis used in dGL. One challenge is ensuring that such proves are no more complex than strictly necessary, yet still sound by constructive principles.

In lieu of the law of excluded middle, the case analysis principle in constructive analysis is the Approximate Splitting Principle:

$$\frac{\Gamma \vdash \epsilon > 0}{\Gamma \vdash \theta > \eta \vee \theta < \eta + \epsilon}$$

That is, we can freely compare two terms, so long as we allow an overlap of some $\epsilon > 0$. We allow arbitrary proofs of $\epsilon > 0$ for generality, but in practice ϵ will usually a rational literal, for which the premiss is trivially decidable.

Most arithmetic proofs in classical dGL make heavy use of automated decision procedures (quantifier elimination, or QE) over real-closed fields. It is true that constructive analysis and classical analysis describe the exact same field, the reals, which is real-closed regardless of how

it is defined. However, the greater problem is that we might wish to use decision procedures to decide formulas involving disjunction $\phi \vee \psi$, and the constructive $\vee$ connective does not follow from its classical counterpart. For example, it is perfectly acceptable to conclude $x > 0 \vee x \leq 0$ classically by QE, but unacceptable to so constructively, as there is no decision procedure which decides between the two branches for any concrete x. This goes to show we must carefully re-evaluate which forms of QE reasoning are sound in CdGL; we can no longer accept the classical rule:

$$\frac{*}{\phi}\ \text{QE}$$

with the side condition that ϕ is a classical consequence of the theory of real-closed fields.

Instead, we introduce several QE rules. Firstly, it is still sound to apply general QE to a double-negated formula:

$$\frac{*}{\neg\neg\phi}\ \text{QE-DN}$$

with the side condition that ϕ is a classical consequence of the theory of real-closed fields. This is sound because $\neg\neg\phi$ holds constructively any time ϕ holds classically. Generally speaking, double negations can be used in a proof (and model) to indicate formulas which need not hold constructively. We postulate that in practical proving, most expensive arithmetic computations (such as postcondition reasoning and differential invariant reasoning) can be placed in a classical modality so that they are amenable to general QE.

Secondly, there are several fragments of constructive arithmetic for which QE can be made constructive. We call a formula ϕ *QE-positive* if all variables are universally quantified, and under the quantifiers there are only conjunctions of comparisons. Then the following rule is sound:

$$\frac{*}{\phi}\ \text{QE-Pos}$$

with the side condition that ϕ is QE-positive and classically valid. QE-positive formulas have trivial computational content, because they represent tautologies that cannot be inspected. When ϕ is QE-positive and valid, we know $\phi(x)$ and each of its conjuncts describe the entire state space, and so hold merely by virtue of x being a real number. A contentless proof term then suffices as a witness.

We note another decidable fragment, which we call *QE-negative*. We say a formula ϕ is *QE-negative* if all variables are existentially quantified, and under the quantifiers there are only disjunctions of comparisons.

$$\frac{*}{\phi}\ \text{QE-Neg}$$

with the side condition that ϕ is QE-negative and classically valid. Unlike the QE-positive fragment, the QE-negative fragment has interesting computational content which can be consumed: a proof of ϕ consists of a satisfying assignment and a proof that it indeed satisifies some given disjunct. Fortunately, this content can be constructed automatically. In the case of a unary disjunction, this is the instance-finding problem for first-order arithmetic. Implementations of this problem, while not simple, are widely available: the most obvious implementation is that which first solves the validity problem by constructing a cylindrical algebraic decomposition (CAD), then searches the CAD for the satisfying instance. With multiple disjunctions, simply search for instances of each disjunct in turn.

The decidability of the QE-positive and QE-negative fragments may turn out to be a mere academic amusement, but the ability to embed classical arithmetic ought to significantly aid practical proofs.

5.8 Soundness Theorem

Lemma 15 (Renaming)**.** *If* $\Gamma \vdash M : \phi$ *then* $[y/x]\Gamma \vdash [y/x]M : [y/x]\phi$.
Lemma 16 (Context substitution)**.** *If* $\Gamma, x : \psi \vdash M : \phi$ *and* $\Gamma \vdash N : \psi$ *then* $\Gamma \vdash [N/x]M : \phi$.
Lemma 17 (Variable substitution)**.** *If* $\Gamma \vdash M : \phi$ *and* $[\theta/x]\phi$ *is admissible then* $[\theta/x]\Gamma \vdash [\theta/x]M : [\theta/x]\phi$.
Theorem 18 (Soundness of Proof Calculus)**.** *If* $\cdot \vdash M : \phi$ *then* ϕ *is realizability-valid.*

The proof follows by structural induction on the derivation. Proofs can be assumed to be normal.

5.9 Relaxing (Constructive) Games to (Constructive) Systems

There is a folk theorem that given a game and a strategy for it, the strategy can be inlined into the game to produce a hybrid system which implements the strategy. Because CdGL represents strategies explicitly, it gives us a framework in which to make this a true theorem. Beyond our academic interest in making the folk theorem rigorous, this also provides an alternative formulation of EP for games: whenever a strategy is proved to exist, the strategy can be witnessed by inlining it into the hybrid game to produce a hybrid system.

We introduce some new concepts in order to state the theorem rigorously. We write $\mathbf{Pt}$ for the set of all proof terms and $\mathbf{Tr}(\alpha)$ for the set of all program traces for a program α. A trace is simply the sequence of all choices taken, e.g., which branch of a nondeterministic choice is executed or which term is substituted for a nondeterministic assignment. We then introduce the inlining function $\sigma_{[\alpha]}(M)$ and $\sigma_{\langle\alpha\rangle}(M)$ which inline Demon or Angel strategies M (respectively) into a game α. We write $\alpha \leq \beta$ to say α refines β. In the case of systems, α refines β when its state transitions are a strict subset of those for β. To generalize refinement to games, we observe that the dual operator reverses direction of refinement, i.e., $(\alpha^d \leq \beta) \equiv (\alpha \geq \beta)$. We say a formula ϕ is α-controllable in Γ if there exists a computable function $f : \mathcal{S} \rightharpoonup \mathbf{Pt}$ such that in every $\omega \in [\![\Gamma]\!]$, we have $\Gamma \vdash f(\omega) : \langle\alpha\rangle\phi$ and additionally α is decideable (i.e., deterministic, with all branches computable) within Γ. We say a formula ϕ is α-monitorable in Γ (for a system α) if there exists a computable function $f : \mathbf{Tr}(\alpha) \times \mathcal{S} \rightharpoonup \mathbf{Pt}$ where for $\omega \in [\![\Gamma]\!]$ and ν such that $(\omega, \nu) \in [\![\alpha]\!]$ witnessed by a trace $t \in \mathbf{Tr}(\alpha)$, then $\Gamma \vdash f(t, \omega) : [\alpha]\phi$. These notions of controllability and monitorability are non-trivial, but they capture the notions that (in addition to the initial state) an Angel proof can be used to compute a system under-approximation of the game where the same postcondition is achievable in a computable way, while a Demon proof can be used to compute a system over-approximation where the postcondition can be guaranteed given knowledge of which path was taken. Given these non-trivial definitions, the relaxation theorems are stated simply:

Theorem 19 (Angelic relaxation)**.** *If* $\Gamma \vdash M : \langle\alpha\rangle\phi$*, then let* $\beta = \sigma_{\langle\alpha\rangle}(M)$*. Then* $\beta \leq \alpha$ *and* ϕ *is* β*-controllable in* Γ.

Theorem 20 (Demonic relaxation)**.** *If* $\Gamma \vdash M : [\alpha]\phi$*, then let* $\beta = \sigma_{[\alpha]}(M)$*. Then* $\beta \geq \alpha$ *and* ϕ *is* β*-monitorable in* Γ*.*

5.10 Relating Classical and Constructive Games

Every constructive theorem holds classically:

Theorem 21 (Conservativity)**.** *If* $\Gamma \vdash M : \phi$ *in* CdGL *then* ϕ *is valid in* dGL*.*

dGL can then be encoded in CdGL via a typical Gödel-Gentzen style translation (let that translation be called T).

Theorem 22 (dGL Embedding)**.** *If* ϕ *is valid in* dGL*, then* $T(\phi)$ *is Kripke-valid in* CdGL*.*

This result is useful because it allows us to embed classical statements in a constructive proof, much as if classicality were a "modality". For many subgoals of a proof, constructivity is overkill: in particular, many arithmetic goals benefit from access to classical quantifier elimination, and their proofs never need be inspected.

Chapter 6

Proposed Work: ProofPlex, Proof-Directed Game Synthesis

In this chapter we propose a tool for the synthesis of controllers and monitors from (proven-safe and proven-live) computational hybrid games, which we name ProofPlex. We motivate ProofPlex by first considering the different artifacts one might wish to synthesize:

1. A *controller monitor*, which detects whether an untrusted external controller is compliant with the assumptions of a controller model
2. A *plant monitor*, which detects whether the environment is compliant with the physical assumptions of the plant model.
3. A *controller*, which actually picks a guaranteed-safe-and-live control decision to carry out
4. A *plant simulator*, which picks an adversarial behavior for the plant and evaluates the effect of the plant.

All of these are desirable under different circumstances: Synthesizing a controller is appealing because unlike an untrusted controller, it is guaranteed to make a safe decision so the feedback need not be invoked (i.e, it is proven to satisfy the controller monitor). Yet the downside of a synthesized controller is that it sacrifices flexibility: if our synthesized monitor guarantees correctness but was not optimized for operational efficiency, fuel efficiency etc., we might one day decide we prefer a controller which meets those criteria instead. In this case the flexibility given by the monitoring approach is preferred. On the plant side, plant monitors are perfect for deployment in production systems or during field tests: they allow us to assess whether assumptions on the environment are met in practice. But in early stages of development, especially if there is not a purpose built simulator for our system already, we would much prefer to synthesize code which can actually (and faithfully) simulate the plant for us, which is helpful in designing the corresponding controller. The first two are monitors which merely detect noncompliance with a model, whereas the latter two actively control the system. Notably, the first two are supported by the existing ModelPlex tool while the latter two are not. Moreover, we obviously wish that we could synthesize these artifacts for arbitrarily complicated systems, whereas existing implementations of ModelPlex have limitations, for example that the controller must contain no differential equations and loops. Until recent advances, it was even more restrictive, with plants restricted to only nilpotent differential equations.

It is by looking at the advances that led to recent gains that we can then, however, discover the changes that achieve our lofty goal of synthesizing all four artifacts and doing so for arbitrary systems (and games). A key insight behind a recent advance in the ModelPlex tool (which generalized plants to polynomial ODEs so long as the plant is proven safe using only chains of inductive invariants) is that even in synthesizing a monitor, *the contents of a proof are essential to synthesis*. Specifically, that work extracted differential invariants from a safety proof in order to synthesize monitors In retrospect, this key insight is no longer surprising: synthesis should be at least as difficult as verification, and because verification of hybrid systems/games is undecidable, a general-case synthesis procedure should not exist. Yet even though verification is undecidable, proof *checking* is utterly decidable, as the proof contains all the difficult decisions made in determining the truth of a liveness statement. Therefore, the ultimate question behind this chapter is, "can the proof content be reused for synthesis"?, to which the thesis statement answers "yes". A proof in CdGL is nothing but a winning strategy for a hybrid game, which contains both all the control decisions made by the Angelic player and the assumptions monitored of the Demonic player. In short, the following could be called the punchline of the thesis:

A box proof witnesses monitorability and a diamond witnesses controllability, thus a CdGL proof witnesses syn

Whenever the players alternate turns in a game, the proof alternates modalities. Whenever we prove a $\langle\alpha\rangle P$ property, we do so by witnessing the decisions (computations) Angel performs to achieve P. In proving a $[\alpha]P$ property, we enumerate rules which Angel must uphold until control passes to Demon, and in doing so we enumerate the contents of a monitor.

The subthesis of this chapter is that this content can and should be used to automate the synthesis of all four artifacts, and that it can do so not just in special cases but in general cases, including:

- Model-predictive controllers using differential equations
- Differential equations with not only differential invariants but also differential ghosts, which together suffice to prove any polynomial invariant of a polynomial ODE
- Controllers containing loops
- Nonatomic plants which for example employ different ODEs in different cases, or may employ sequences of ODEs to model different stages of a system

The implementation of ProofPlex contains an implementation of the relaxation theorems of the previous section, which first characterize which classical proofs also work as constructive proofs, then compute a hybrid system relaxation of the game. Once a relaxation for the controller model (and controller proof) in a hybrid game has been computed, verification can be made end-to-end by hooking it in to the existing VeriPhy pipeline. What remains is to supply the untrusted controller. Because ProofPlex, unlike ModelPlex, also synthesizes monitors by exploiting angelic proofs, we automatically provide an "untrusted" controller which in fact is trustworthy, because it simply an implementation of the (operational) angelic dynamics of the control proof in CdGL. By the work in the previous chapter, these operational semantics refine the standard denotational semantics of both the hybrid game and of its systems relaxation, meaning they satisfy the monitor.

Chapter 7

Conclusion

7.1 Timeline

The timeline given below is from the time this document is completed (say, April 2019). Due to geographic issues, I expect to propose in early Fall (Sep 2019). If all goes according to plan, I intend to defend in late Spring 2020.

Total time: 13 months

- 1 month: Extend dL formalization in Isabelle to support dL_ι
- 1 month: Refine KeYmaera X to Isabelle/HOL interface, ensure sufficient support for case study
- 4 months: Develop CdGL theory (Submit to POPL in July 2019)
- 2 months: Implement CdGL proof checking
- 2 months: Develop ProofPlex implementation (Submit in January 2020)
- 1 month: Generalize ground robotics case study to CdGL
- 2 months: Write thesis document

References

Aczel, P., & Gambino, N. (2006). The generalised type-theoretic interpretation of constructive set theory. *J. Symb. Log.*, *71*(1), 67–103. Retrieved from https://doi.org/10.2178/jsl/1140641163 doi: 10.2178/jsl/1140641163

Alechina, N., Mendler, M., de Paiva, V., & Ritter, E. (2001). Categorical and kripke semantics for constructive S4 modal logic. In L. Fribourg (Ed.), *Computer science logic, 15th international workshop, CSL 2001. 10th annual conference of the eacsl, paris, france, september 10-13, 2001, proceedings* (Vol. 2142, pp. 292–307). Springer. Retrieved from https://doi.org/10.1007/3-540-44802-0_21 doi: 10.1007/3-540-44802-0_21

Allen, S. F., Bickford, M., Constable, R. L., Eaton, R., Kreitz, C., Lorigo, L., & Moran, E. (2006). Innovations in computational type theory using Nuprl. *J. Applied Logic*, *4*(4), 428-469. (http://www.nuprl.org/)

Althoff, M., & Dolan, J. M. (2014). Online verification of automated road vehicles using reachability analysis. *IEEE Trans. Robotics*, *30*(4), 903–918. Retrieved from https://doi.org/10.1109/TRO.2014.2312453

Alur, R., Henzinger, T. A., Lafferriere, G., & Pappas, G. J. (2000, July). Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, *88*(7), 971-984. doi: 10.1109/5.871304

Anand, A., & Knepper, R. A. (2015). ROSCoq: Robots powered by constructive reals. In C. Urban & X. Zhang (Eds.), *ITP 2015* (Vol. 9236, pp. 34–50). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-319-22102-1_3 doi: 10.1007/978-3-319-22102-1_3

Anand, A., & Rahli, V. (2014a). Towards a formally verified proof assistant. In (pp. 27–44). Retrieved from http://dx.doi.org/10.1007/978-3-319-08970-6_3 doi: 10.1007/978-3-319-08970-6_3

Anand, A., & Rahli, V. (2014b). Towards a formally verified proof assistant. In G. Klein & R. Gamboa (Eds.), *Itp* (Vol. 8558, pp. 27–44). Springer. doi: 10.1007/978-3-319-08970-6_3

Barras, B. (2010). Sets in Coq, Coq in sets. *J. Formalized Reasoning*, *3*(1), 29–48. doi: 10.6092/issn.1972-5787/1695

Barras, B., & Werner, B. (1997). *Coq in Coq* (Tech. Rep.). INRIA Rocquencourt.

Bhatia, A., Kavraki, L. E., & Vardi, M. Y. (2010). Motion planning with hybrid dynamics and temporal goals. In *Cdc* (pp. 1108–1115). IEEE. Retrieved from https://doi.org/10.1109/CDC.2010.5717440

Bishop, E. (1967). Foundations of constructive analysis.

Blazy, S., Paulin-Mohring, C., & Pichardie, D. (Eds.). (2013). *Interactive theorem proving*

- 4th international conference, ITP 2013, rennes, france, july 22-26, 2013. proceedings (Vol. 7998). Springer. Retrieved from https://doi.org/10.1007/978-3-642-39634-2 doi: 10.1007/978-3-642-39634-2

Bohrer, B., Rahli, V., Vukotic, I., Völp, M., & Platzer, A. (2017). Formally verified differential dynamic logic. In Y. Bertot & V. Vafeiadis (Eds.), *Cpp* (p. 208-221). ACM. doi: 10.1145/3018610.3018616

Bohrer, B., Tan, Y. K., Mitsch, S., Myreen, M. O., & Platzer, A. (2018). VeriPhy: Verified controller executables from verified cyber-physical system models. In D. Grossman (Ed.), *Pldi* (p. 617-630). ACM. doi: 10.1145/3192366.3192406

Boldo, S., Filliâtre, J., & Melquiond, G. (2009). Combining Coq and Gappa for certifying floating-point programs. In J. Carette, L. Dixon, C. S. Coen, & S. M. Watt (Eds.), *Mkm, held as part of CICM* (Vol. 5625, pp. 59–74). Springer. Retrieved from https://doi.org/10.1007/978-3-642-02614-0_10 doi: 10.1007/978-3-642-02614-0_10

Boldo, S., Jourdan, J., Leroy, X., & Melquiond, G. (2013). A formally-verified C compiler supporting floating-point arithmetic. In A. Nannarelli, P. Seidel, & P. T. P. Tang (Eds.), *Arith* (pp. 107–115). IEEE Computer Society. Retrieved from https://doi.org/10.1109/ARITH.2013.30 doi: 10.1109/ARITH.2013.30

Boldo, S., & Melquiond, G. (2011). Flocq: A unified library for proving floating-point algorithms in Coq. In E. Antelo, D. Hough, & P. Ienne (Eds.), *Arith* (pp. 243–252). IEEE Computer Society. Retrieved from https://doi.org/10.1109/ARITH.2011.40 doi: 10.1109/ARITH.2011.40

Bouissou, O., Goubault, E., Putot, S., Tekkal, K., & Védrine, F. (2009). Hybridfluctuat: A static analyzer of numerical programs within a continuous environment. In A. Bouajjani & O. Maler (Eds.), (Vol. 5643, pp. 620–626). Springer. Retrieved from https://doi.org/10.1007/978-3-642-02658-4_46 doi: 10.1007/978-3-642-02658-4_46

Bourke, T., Brun, L., Dagand, P., Leroy, X., Pouzet, M., & Rieg, L. (2017). A formally verified compiler for lustre. In A. Cohen & M. T. Vechev (Eds.), *Pldi* (pp. 586–601). ACM. Retrieved from http://doi.acm.org/10.1145/3062341.3062358 doi: 10.1145/3062341.3062358

Brandon Bohrer, A. P., Manuel Fernandez. (2019). dli: Definite descriptions in differential dynamic logic. In *Cade.*

Bridges, D. S., & Vita, L. S. (2007). *Techniques of constructive analysis*. Springer Science & Business Media.

Celani, S. A. (2001). A fragment of intuitionistic dynamic logic. *Fundam. Inform.*, *46*(3), 187–197. Retrieved from http://content.iospress.com/articles/fundamenta-informaticae/fi46-3-01

Chen, X., Ábrahám, E., & Sankaranarayanan, S. (2013). Flow*: An analyzer for non-linear hybrid systems. In N. Sharygina & H. Veith (Eds.), *Cav* (Vol. 8044, pp. 258–263). Springer. Retrieved from https://doi.org/10.1007/978-3-642-39799-8_18 doi: 10.1007/978-3-642-39799-8_18

Chen, X., Schupp, S., Makhlouf, I. B., Ábrahám, E., Frehse, G., & Kowalewski, S. (2015). A benchmark suite for hybrid systems reachability analysis. In *Nfm* (Vol. 9058).

Church, A. (1956). *Introduction to mathematical logic*. Princeton University Press.

Constable, R., Allen, S., Bromley, H., Cleaveland, W., Cremer, J., Harper, R., ... Smith, S.

(1986). *Implementing mathematics with the Nuprl proof development system*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Coq Proof Assistant. (n.d.). Retrieved from http://coq.inria.fr/ (Accessed: 2016-11-28)

Daumas, M., Rideau, L., & Théry, L. (2001). A generic library for floating-point numbers and its application to exact computing. In R. J. Boulton & P. B. Jackson (Eds.), *Tphols* (Vol. 2152, pp. 169–184). Springer. Retrieved from https://doi.org/10.1007/3-540-44755-5_13 doi: 10.1007/3-540-44755-5_13

Degen, J., & Werner, J. (2006). Towards intuitionistic dynamic logic. *Logic and Logical Philosophy*, *15*(4), 305–324.

Doberkat, E. (2011). Towards a coalgebraic interpretation of propositional dynamic logic. *CoRR*, *abs/1109.3685*. Retrieved from http://arxiv.org/abs/1109.3685

Driver, R. (1998). Torricelli's law: An ideal example of an elementary ODE. *The American Mathematical Monthly*, *105*(5), 453–455.

Duggirala, P. S., Mitra, S., Viswanathan, M., & Potok, M. (2015). C2E2: A verification tool for stateflow models. In C. Baier & C. Tinelli (Eds.), *Tacas* (Vol. 9035, pp. 68–82). Springer. Retrieved from https://doi.org/10.1007/978-3-662-46681-0_5 doi: 10.1007/978-3-662-46681-0_5

Fainekos, G. E., Girard, A., Kress-Gazit, H., & Pappas, G. J. (2009). Temporal logic motion planning for dynamic robots. *Automatica*, *45*(2). Retrieved from https://doi.org/10.1016/j.automatica.2008.08.008

Filippidis, I., Dathathri, S., Livingston, S. C., Ozay, N., & Murray, R. M. (2016). Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *Cca*. IEEE. Retrieved from https://doi.org/10.1109/CCA.2016.7587949

Finucane, C., Jing, G., & Kress-Gazit, H. (2010). LTLMoP: Experimenting with language, temporal logic and robot control. In *Iros*. IEEE. Retrieved from https://doi.org/10.1109/IROS.2010.5650371

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robot. Automat. Mag.*, *4*(1). Retrieved from https://doi.org/10.1109/100.580977

Franchetti, F., Low, T. M., Mitsch, S., Mendoza, J. P., Gui, L., Phaosawasdi, A., ... Veloso, M. (2017). High-assurance SPIRAL: End-to-end guarantees for robot and car control. *IEEE Control Systems*, *37*(2), 82-103. doi: 10.1109/MCS.2016.2643244

Frehse, G., Guernic, C. L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., ... Maler, O. (2011). Spaceex: Scalable verification of hybrid systems. In G. Gopalakrishnan & S. Qadeer (Eds.), *Cav* (Vol. 6806, pp. 379–395). doi: 10.1007/978-3-642-22110-1_30

Frittella, S., Greco, G., Kurz, A., Palmigiano, A., & Sikimic, V. (2016). A proof-theoretic semantic analysis of dynamic epistemic logic. *J. Log. Comput.*, *26*(6), 1961–2015. Retrieved from https://doi.org/10.1093/logcom/exu063 doi: 10.1093/logcom/exu063

Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., & Platzer, A. (2015). KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In A. P. Felty & A. Middeldorp (Eds.), *Cade* (Vol. 9195, pp. 527–538). Springer. doi: 10.1007/978-3-319-21401-6_36

Ghilardi, S. (1989). Presheaf semantics and independence results for some non-classical first-

order logics. *Arch. Math. Log.*, *29*(2), 125–136. Retrieved from https://doi.org/10.1007/BF01620621 doi: 10.1007/BF01620621

Ghosh, S. (2008). Strategies made explicit in dynamic game logic. *Logic and the foundations of game and decision theory*.

Guéneau, A., Myreen, M. O., Kumar, R., & Norrish, M. (2017). Verified characteristic formulae for CakeML. In H. Yang (Ed.), *Esop* (Vol. 10201, pp. 584–610). Springer. Retrieved from https://doi.org/10.1007/978-3-662-54434-1_22 doi: 10.1007/978-3-662-54434-1_22

Halbwachs, N., Lagnier, F., & Ratel, C. (1992). Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. *IEEE Trans. Software Eng.*, *18*(9), 785–793. Retrieved from https://doi.org/10.1109/32.159839 doi: 10.1109/32.159839

Harrison, J. (2006a). Floating-point verification using theorem proving. In M. Bernardo & A. Cimatti (Eds.), *Formal methods for hardware verification, sfm* (Vol. 3965, pp. 211–242). Springer. Retrieved from https://doi.org/10.1007/11757283_8 doi: 10.1007/11757283_8

Harrison, J. (2006b). Towards self-verification of HOL Light. In U. Furbach & N. Shankar (Eds.), *Ijcar 2006* (Vol. 4130, p. 177-191). Springer.

Henzinger, T. A. (1996). The theory of hybrid automata. In *Proceedings, 11th annual IEEE symposium on logic in computer science, new brunswick, new jersey, usa, july 27-30, 1996* (pp. 278–292). IEEE Computer Society. Retrieved from https://doi.org/10.1109/LICS.1996.561342 doi: 10.1109/LICS.1996.561342

Hilken, B. P., & Rydeheard, D. E. (n.d.). *A first order modal logic and its sheaf models.*

Hubbard, J. H., & West, B. H. (n.d.). *Differential equations: A dynamical systems approach* (Vol. 18). Springer. doi: 10.1007/978-1-4612-4192-8

Immler, F. (2015). Verified reachability analysis of continuous systems. In (pp. 37–51). Retrieved from http://dx.doi.org/10.1007/978-3-662-46681-0_3 doi: 10.1007/978-3-662-46681-0_3

Immler, F., & Traut, C. (2016). The flow of odes. In J. C. Blanchette & S. Merz (Eds.), *Itp* (Vol. 9807, pp. 184–199). Springer. Retrieved from https://doi.org/10.1007/978-3-319-43144-4_12 doi: 10.1007/978-3-319-43144-4_12

Isabelle. (n.d.). Retrieved from https://isabelle.in.tum.de/ (Accessed: 2016-11-28)

Jeannin, J., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., & Platzer, A. (2015). Formal verification of ACAS X, an industrial airborne collision avoidance system. In A. Girault & N. Guan (Eds.), *Emsoft* (p. 127-136). IEEE Press. doi: 10.1109/EMSOFT.2015.7318268

Kamide, N., & Wansing, H. (2010). Combining linear-time temporal logic with constructiveness and paraconsistency. *J. Applied Logic*, *8*(1), 33–61. Retrieved from https://doi.org/10.1016/j.jal.2009.06.001 doi: 10.1016/j.jal.2009.06.001

Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., ... Winwood, S. (2010). sel4: Formal verification of an operating-system kernel. *Commun. ACM*, *53*(6), 107–115. doi: 10.1145/1743546.1743574

Kloetzer, M., & Belta, C. (2008). A fully automated framework for control of linear systems from

temporal logic specifications. *IEEE Trans. Automat. Contr.*, *53*(1), 287–297. Retrieved from https://doi.org/10.1109/TAC.2007.914952

Krebbers, R., & Spitters, B. (2011). Type classes for efficient exact real arithmetic in Coq. *Logical Methods in Computer Science*, *9*(1). Retrieved from http://dx.doi.org/10.2168/LMCS-9(1:01)2013 doi: 10.2168/LMCS-9(1:01)2013

Krogh, B. (1998). The SIMPLEX architecture for safe on-line control system upgrades. In *Acc.* Retrieved from https://doi.org/10.1109/ACC.1998.703255

Kumar, R., Arthan, R., Myreen, M. O., & Owens, S. (2016a). Self-formalisation of higher-order logic - semantics, soundness, and a verified implementation. *J. Autom. Reasoning*, *56*(3), 221–259. Retrieved from http://dx.doi.org/10.1007/s10817-015-9357-x doi: 10.1007/s10817-015-9357-x

Kumar, R., Arthan, R., Myreen, M. O., & Owens, S. (2016b). Self-formalisation of higher-order logic: Semantics, soundness, and a verified implementation. *J. Autom. Reasoning*, *56*(3), 221–259. doi: 10.1007/s10817-015-9357-x

Kumar, R., Myreen, M. O., Norrish, M., & Owens, S. (2014). CakeML: a verified implementation of ML. In S. Jagannathan & P. Sewell (Eds.), *Popl'14* (pp. 179–192). ACM.

Lamport, L. (1992). Hybrid systems in TLA$^+$. In R. L. Grossman, A. Nerode, A. P. Ravn, & H. Rischel (Eds.), *Hybrid systems* (Vol. 736, pp. 77–102). Springer. Retrieved from http://dx.doi.org/10.1007/3-540-57318-6_25 doi: 10.1007/3-540-57318-6_25

Lipton, J. (1992). Constructive kripke semantics and realizability. In *Logic from computer science* (pp. 319–357).

Loos, S. M., Platzer, A., & Nistor, L. (2011). Adaptive cruise control: Hybrid, distributed, and now formally verified. In M. Butler & W. Schulte (Eds.), *Fm* (Vol. 6664, p. 42-56). Springer. doi: 10.1007/978-3-642-21437-0_6

Majumdar, R., Saha, I., & Zamani, M. (2012). Synthesis of minimal-error control software. In A. Jerraya, L. P. Carloni, F. Maraninchi, & J. Regehr (Eds.), *Emsoft* (pp. 123–132). ACM. Retrieved from http://doi.acm.org/10.1145/2380356.2380380 doi: 10.1145/2380356.2380380

Makarov, E., & Spitters, B. (2013). The picard algorithm for ordinary differential equations in coq. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.), *Interactive theorem proving - 4th international conference, ITP 2013, rennes, france, july 22-26, 2013. proceedings* (Vol. 7998, pp. 463–468). Springer. Retrieved from https://doi.org/10.1007/978-3-642-39634-2_34 doi: 10.1007/978-3-642-39634-2_34

Martin, B., Ghorbal, K., Goubault, E., & Putot, S. (2017). Formal verification of station keeping maneuvers for a planar autonomous hybrid system. In L. Bulwahn, M. Kamali, & S. Linker (Eds.), *FVAV@iFM* (Vol. 257, pp. 91–104). doi: 10.4204/EPTCS.257.9

Martinez, A. A., Majumdar, R., Saha, I., & Tabuada, P. (2010). Automatic verification of control system implementations. In L. P. Carloni & S. Tripakis (Eds.), *Emsoft* (pp. 9–18). ACM. Retrieved from http://doi.acm.org/10.1145/1879021.1879024 doi: 10.1145/1879021.1879024

Melquiond, G. (2012). Floating-point arithmetic in the Coq system. *Inf. Comput.*, *216*, 14–23. Retrieved from https://doi.org/10.1016/j.ic.2011.09.005 doi: 10.1016/j.ic.2011.09.005

Mitsch, S., Ghorbal, K., & Platzer, A. (2013). On provably safe obstacle avoidance for autonomous robotic ground vehicles. In P. Newman, D. Fox, & D. Hsu (Eds.), *Robotics: Science and systems.*

Mitsch, S., Ghorbal, K., Vogelbacher, D., & Platzer, A. (2017). Formal verification of obstacle avoidance and navigation of ground robots. *I. J. Robotics Res.*, *36*(12), 1312-1340. doi: 10.1177/0278364917733549

Mitsch, S., & Platzer, A. (2016). ModelPlex: Verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.*, *49*(1), 33-74. (Special issue of selected papers from RV'14) doi: 10.1007/s10703-016-0241-z

Myreen, M. O., & Davis, J. (2014). The reflective Milawa theorem prover is sound - (down to the machine code that runs it). In (pp. 421–436). Retrieved from http://dx.doi.org/10.1007/978-3-319-08970-6_27 doi: 10.1007/978-3-319-08970-6_27

Myreen, M. O., & Owens, S. (2012). Proof-producing synthesis of ML from higher-order logic. In P. Thiemann & R. B. Findler (Eds.), *Icfp* (pp. 115–126). ACM. Retrieved from http://doi.acm.org/10.1145/2364527.2364545 doi: 10.1145/2364527.2364545

Myreen, M. O., Owens, S., & Kumar, R. (2013). Steps towards verified implementations of HOL Light. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.), (Vol. 7998, pp. 490–495). Springer. Retrieved from https://doi.org/10.1007/978-3-642-39634-2 doi: 10.1007/978-3-642-39634-2

Nilsson, P., Hussien, O., Balkan, A., Chen, Y., Ames, A. D., Grizzle, J. W., ... Tabuada, P. (2016). Correct-by-construction adaptive cruise control: Two approaches. *IEEE Trans. Contr. Sys. Techn.*, *24*(4). Retrieved from https://doi.org/10.1109/TCST.2015.2501351

Platzer, A. (2008). Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, *41*(2), 143-189. doi: 10.1007/s10817-008-9103-8

Platzer, A. (2010). Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, *20*(1), 309-352. (Advance Access published on November 18, 2008) doi: 10.1093/logcom/exn070

Platzer, A. (2011, November). *The complete proof theory of hybrid systems* (Tech. Rep. No. CMU-CS-11-144). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Platzer, A. (2012a). A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, *8*(4), 1-44. (Special issue for selected papers from CSL'10) doi: 10.2168/LMCS-8(4:17)2012

Platzer, A. (2012b). Logics of dynamical systems. In *Lics* (p. 13-24). IEEE. doi: 10.1109/LICS.2012.13

Platzer, A. (2015). Differential game logic. *ACM Trans. Comput. Log.*, *17*(1), 1:1–1:51. doi: 10.1145/2817824

Platzer, A. (2016). Logic & proofs for cyber-physical systems. In N. Olivetti & A. Tiwari (Eds.), *Ijcar* (Vol. 9706, p. 15-21). Springer. doi: 10.1007/978-3-319-40229-1_3

Platzer, A. (2017a). A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, *59*(2), 219-265. doi: 10.1007/s10817-016-9385-1

Platzer, A. (2017b). Differential hybrid games. *ACM Trans. Comput. Log.*, *18*(3), 19:1-19:44.

doi: 10.1145/3091123

Platzer, A. (2018a). *Logical foundations of cyber-physical systems*. Switzerland: Springer. Retrieved from http://www.springer.com/978-3-319-63587-3

Platzer, A. (2018b). Uniform substitution for differential game logic. In D. Galmiche, S. Schulz, & R. Sebastiani (Eds.), *Ijcar* (Vol. 10900, p. 211-227). Springer. doi: 10.1007/978-3-319-94205-6_15

Platzer, A., & Quesel, J.-D. (2008a). KeYmaera: A hybrid theorem prover for hybrid systems. In A. Armando, P. Baumgartner, & G. Dowek (Eds.), *Ijcar* (Vol. 5195, p. 171-178). Springer. doi: 10.1007/978-3-540-71070-7_15

Platzer, A., & Quesel, J.-D. (2008b). Logical verification and systematic parametric analysis in train control. In M. Egerstedt & B. Mishra (Eds.), *Hscc* (Vol. 4981, p. 646-649). Springer. doi: 10.1007/978-3-540-78929-1_55

Quesel, J.-D., & Platzer, A. (2012). Playing hybrid games with keymaera. In B. Gramlich, D. Miller, & U. Sattler (Eds.), *Ijcar* (Vol. 7364, p. 439-453). Springer. doi: 10.1007/978-3-642-31365-3_34

Rahli, V., & Bickford, M. (2016). A nominal exploration of intuitionism. In J. Avigad & A. Chlipala (Eds.), *CPP 2016* (pp. 130–141). ACM. Retrieved from http://doi.acm.org/10.1145/2854065.2854077 doi: 10.1145/2854065.2854077

Ramanujam, R., & Simon, S. E. (2008). Dynamic logic on games with structured strategies. In G. Brewka & J. Lang (Eds.), *Principles of knowledge representation and reasoning: Proceedings of the eleventh international conference, KR 2008, sydney, australia, september 16-19, 2008* (pp. 49–58). AAAI Press. Retrieved from http://www.aaai.org/Library/KR/2008/kr08-006.php

Ricketts, D., Malecha, G., Alvarez, M. M., Gowda, V., & Lerner, S. (2015). Towards verification of hybrid systems in a foundational proof assistant. In *MEMOCODE 2015* (pp. 248–257). IEEE. Retrieved from http://dx.doi.org/10.1109/MEMCOD.2015.7340492 doi: 10.1109/MEMCOD.2015.7340492

Rizaldi, A., Immler, F., Schürmann, B., & Althoff, M. (2018). A formally verified motion planner for autonomous vehicles. In *Atva* (Vol. 11138). Springer. Retrieved from https://doi.org/10.1007/978-3-030-01090-4_5

Schwichtenberg, H. (2008). Realizability interpretation of proofs in constructive analysis. *Theory Comput. Syst.*, *43*(3-4), 583–602. Retrieved from https://doi.org/10.1007/s00224-007-9027-4 doi: 10.1007/s00224-007-9027-4

Scott, D. (1968). Extending the topological interpretation to intuitionistic analysis. *Compositio Mathematica*, *20*, 194-210. Retrieved from http://www.numdam.org/item/CM_1968__20__194_0

Sewell, T. A. L., Myreen, M. O., & Klein, G. (2013). Translation validation for a verified OS kernel. In H. Boehm & C. Flanagan (Eds.), *Pldi* (pp. 471–482). ACM. Retrieved from http://doi.acm.org/10.1145/2462156.2462183 doi: 10.1145/2462156.2462183

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2018). AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field serv. robot.* (Vol. 5, pp. 621–635).

Taly, A., & Tiwari, A. (2010). Switching logic synthesis for reachability. In L. P. Carloni & S. Tripakis (Eds.), *EMSOFT* (pp. 19–28). ACM. Retrieved from https://doi.org/

10.1145/1879021.1879025

Tan, Y. K., Myreen, M. O., Kumar, R., Fox, A. C. J., Owens, S., & Norrish, M. (2016). A new verified compiler backend for cakeml. In J. Garrigue, G. Keller, & E. Sumii (Eds.), *Icfp* (pp. 60–73). ACM. Retrieved from http://doi.acm.org/10.1145/2951913.2951924 doi: 10.1145/2951913.2951924

Tarski, A. (1998). A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition* (pp. 24–84). Springer.

Van Benthem, J. (2001). Games in dynamic-epistemic logic. *Bulletin of Economic Research*, *53*(4), 219–248.

van Benthem, J. (2015). Logic of strategies: What and how? In J. van Benthem, S. Ghosh, & R. Verbrugge (Eds.), *Models of strategic reasoning - logics, games, and communities* (Vol. 8972, pp. 321–332). Springer. Retrieved from https://doi.org/10.1007/978-3-662-48540-8_10 doi: 10.1007/978-3-662-48540-8_10

van Benthem, J., & Bezhanishvili, G. (2007). Modal logics of space. In M. Aiello, I. Pratt-Hartmann, & J. van Benthem (Eds.), *Handbook of spatial logics* (pp. 217–298). Springer. Retrieved from https://doi.org/10.1007/978-1-4020-5587-4_5 doi: 10.1007/978-1-4020-5587-4_5

van Benthem, J., Bezhanishvili, N., & Enqvist, S. (2017). A propositional dynamic logic for instantial neighborhood models. In A. Baltag, J. Seligman, & T. Yamada (Eds.), *Logic, rationality, and interaction - 6th international workshop, LORI 2017, sapporo, japan, september 11-14, 2017, proceedings* (Vol. 10455, pp. 137–150). Springer. Retrieved from https://doi.org/10.1007/978-3-662-55665-8_10 doi: 10.1007/978-3-662-55665-8_10

van Benthem, J., & Pacuit, E. (2011). Dynamic logics of evidence-based beliefs. *Studia Logica*, *99*(1-3), 61–92. Retrieved from https://doi.org/10.1007/s11225-011-9347-x doi: 10.1007/s11225-011-9347-x

van Benthem, J., Pacuit, E., & Roy, O. (2011). Toward a theory of play: A logical perspective on games and interaction. *Games*, *2*(1), 52–86. Retrieved from https://doi.org/10.3390/g2010052 doi: 10.3390/g2010052

van Oosten, J. (2002). Realizability: A historical essay. *Mathematical Structures in Computer Science*, *12*(3), 239–263. Retrieved from https://doi.org/10.1017/S0960129502003626 doi: 10.1017/S0960129502003626

Wijesekera, D. (1990). Constructive modal logics I. *Ann. Pure Appl. Logic*, *50*(3), 271–301. Retrieved from https://doi.org/10.1016/0168-0072(90)90059-B doi: 10.1016/0168-0072(90)90059-B

Wijesekera, D., & Nerode, A. (2005). Tableaux for constructive concurrent dynamic logic. *Ann. Pure Appl. Logic*, *135*(1-3), 1–72. Retrieved from https://doi.org/10.1016/j.apal.2004.12.001 doi: 10.1016/j.apal.2004.12.001

Yu, L. (2013). A formal model of IEEE floating point arithmetic. *Archive of Formal Proofs*. Retrieved from https://www.isa-afp.org/entries/IEEE_Floating_Point.shtml