Verifiably Safe Autonomy for Cyber-Physical Systems

Thesis Proposal

Nathan Fulton

July 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Andrè Platzer, Chair
Jeremy Avigad
Goran Frehse
Stefan Mitsch
Zico Kolter

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

Keywords: Cyber-Physical Systems, Hybrid Systems, Autonomous Systems, Formal Verification, Differential Dynamic Logic, Automated Theorem Proving, Reinforcement Learning

Abstract

This thesis focuses on verifiably safe reinforcement learning for cyber-physical systems. Cyber-physical systems, such as autonomous vehicles and medical devices, are increasingly common and increasingly autonomous. Designing safe cyber-physical systems is difficult because of the interaction between the discrete dynamics of control software and the continuous dynamics of the vehicle's physical movement.

Formal methods capable of reasoning about these hybrid discrete-continuous dynamics can help engineers obtain strong safety guarantees about safety-critical control systems. Several recent successes in applying formal methods to hybrid dynamical systems demonstrate that these tools provide a promising foundation for establishing safety properties about planes, trains, and cars. However, existing theory and tooling does not explain how to obtain formal safety guarantees for systems that use reinforcement learning to discover efficient control policies from data. This gap in existing knowledge is important because modern approaches toward building cyber-physical systems combine machine learning with classical controls engineering to navigate in open environments.

Previously completed work introduces KeYmaera X, a theorem prover for hybrid systems and uses KeYmaera X to obtain verified safety guarantees for control policies generated by reinforcement learning algorithms. These contributions enable strong safety guarantees for optimized control policies when the underlying environment matches a first-principles model. However, classical hybrid systems verification results do not provide guarantees for systems that deviate from an engineer's modeling assumptions.

This thesis introduces an approach toward providing safety guarantees for learned control policies even when reality deviates from modeling assumptions. The core technical contribution will be a new class of algorithms that learn safe ways to update a model in response to newly observed dynamics in the environment. We propose achieving this goal by discovering verification-preserving model updates (VPMUs), thus leveraging hybrid systems theorem proving during the learning process. These contributions will provide verifiable safety guarantees for systems that are controlled by policies obtained through reinforcement learning, justifying the use of reinforcement learning in safety-critical settings.

Contents

Chapter 1

Introduction

The automotive and aeronautical industries have continually improved the energy-efficiency, safety, comfort and automation of vehicles. Achieving these improvements required substantially increasing the size and complexity of vehicle software. The growing use of software in these safety-critical settings inspired the development of mathematical models – called *hybrid systems* – that model the interaction between discrete software systems and the continuous systems under control. Hybrid systems provide a fruitful formalism for stating and proving safety properties about systems that combine discrete computation with continuous control.

Over the past decade, designers of vehicles have moved on from low-level control problems. Tomorrow's software systems not only help control the engine and the brakes, but also make high level decisions about where and how a vehicle should move. Advanced driver-assist systems are already deployed. Most major automobile manufacturers are experimenting with fully autonomous vehicles. Designers of planes and trains are also deploying partially autonomous vehicles and experimenting with fully autonomous systems. The future of mobility is autonomous. These autonomous systems make extensive use of machine learning, such as reinforcement learning, to control in open environments. Therefore, designing safe autonomous cyber-physical systems requires establishing safety properties about systems that use reinforcement learning and other optimization techniques for control.

Unlike traditional software engineering domains where light-weight quality assurance mechanisms (e.g., testing) often suffice, best practices for safety-critical systems suggest the use of formal verification. Ideally, developers of safety-critical systems should construct a model of the system under control and then write a formal, computer-checked proof that their control software satisfies key safety properties with respect to the underlying model. For example, a developer might construct a system of differential equations describing how a car behaves and then prove that a piece of control software prevents the car from entering an unsafe state. Formal proofs of relevant safety properties ensure that a system is *verifiably safe*.

The proposed thesis will demonstrate that **autonomous cyber-physical systems that use reinforcement learning for control are amenable to formal verification**.

1.1 Proposed Work

Fully autonomous systems that make use of both offline and online learning will need to come with strong safety guarantees. Therefore, developing formal methods that are capable of providing safety guarantees for modern learning algorithms is an important challenge.

Despite recent successes in modeling and verifying safety properties about cyber-physical systems, existing hybrid systems verification approaches are not directly applicable to modern reinforcement learning algorithms. Classical software verification tools capable of verifying properties about reinforcement learning algorithms do not provide productive environments for hybrid systems verification. Conversely, existing verification tools specialized to hybrid systems analysis only describe relatively simple software. These simple descriptions of discrete dynamical systems are not productive environments for stating and proving properties about reinforcement learning algorithms. This thesis show how to use hybrid systems analysis tools to provide safety guarantees for systems that use reinforcement learning without resorting to encoding the entire reinforcement learning algorithm as an explicit part of the hybrid system.

Verifiably safe autonomy is an epistemically challenging goal. Verification results for cyber-physical systems are always with respect to a model of the world, but autonomy implies the ability to act reasonably even in situations that were not explicitly modeled by system designers. For this reason, our recent work toward verifiably safe autonomy distinguishes between states where the model is accurate and states where the model is inaccurate [40]. In this work, we establish that verification results transfer to policies obtained by learning whenever the model is accurate. We also suggest a way of deriving reasonable reward functions when the system goes off-model. However, this heuristic does not come with any guarantees.

Our approach toward verifiably safe autonomy will leverage existing cyber-physical systems verification tooling developed by the author and his collaborators over the past several years. Using this theory and tooling, we will extend our previous work [40] to provide formal safety and convergence guarantees even when the system goes off-model. The core technical contribution of this approach will be a way of updating models of cyber-physical systems in a way that preserves verification results and comports with observed deviations from an initial model. We will evaluate this contribution by learning model updates for systems with nonlinear continuous dynamics and complex safe control envelopes. The remainder of this proposal is organized as follows:

- Chapter 2 introduces the logic underlying our proposed approach.
- Chapter 3 presents our prior work on verifying hybrid systems and discusses related work on theorem proving and hybrid systems reachability analysis.
- Chapter 4 outlines our current approach toward verifying cyber-physical systems that uses reinforcement learning for control and compares our prior contributions on this topic with other work on safe/verifiable reinforcement learning.
- Chapter 5 discusses our proposed extensions to the work described in Chapter 4 and compares our proposed contributions to other work on program repair and program synthesis.

Discussion of related work occurs in Chapter 3, Chapter 4, and Chapter 5; Section 5.3 summarizes all of these discussions in Table 5.1.

Chapter 2

Background

The proposed thesis will show that cyber-physical systems that use reinforcement learning for control are amenable to formal verification. Achieving this goal requires combining formal verification results about hybrid systems with reinforcement learning algorithms. This chapter introduces differential dynamic logic (a logic for proving reachability properties of hybrid systems) and classical reinforcement learning algorithms.

2.1 Formal Verification of Hybrid Systems

Hybrid dynamical systems [7, 88] are dynamical systems that combine discrete and continuous dynamical systems. Hybrid dynamical systems are a compelling mathematical formalism for describing the mixture of discrete control and continuous movement that characterizes cyber-physical systems. Designers of control systems often wish to prove that a given hybrid system does not reach an unsafe state; these properties are called *reachability* properties.

This section presents a programming language for describing hybrid dynamical systems called hybrid programs, introduces a differential dynamic logic ($\mathsf{d}\mathcal{L}$) for specifying reachability properties of hybrid programs, and demonstrates how $\mathsf{d}\mathcal{L}$ is used to specify safety properties of hybrid programs.

2.1.1 Hybrid Programs

Hybrid programs [86, 87, 88] are a programming language model of hybrid dynamics. Hybrid programs extend nondeterministic imperative programs (i.e., regular programs) with differential equations. Usually, the discrete portion of a hybrid program describes a nondeterministic set of safe control actions and the continuous portion of a hybrid program describes the physical movement of the system. Although hybrid systems are an expressive mathematical tool capable of accurately modeling a broad range of physical and sociological phenomena, this thesis focuses on the use of hybrid systems to model control systems, such as those found in partially and fully autonomous vehicles. An informal description of hybrid programs is given in Table 2.1.

Program Statement	Meaning
$\alpha;\beta$	Sequentially composes α and β.
$\alpha \cup \beta$	Executes either α or β.
α^*	Repeats α zero or more times.
$x := \theta$	Evaluates θ and assign result to x.
$x := *$	Assigns an arbitrary real value to x.
$\{x_1' = \theta_1, ..., x_n' = \theta_n \& F\}$	Continuous evolution[1].
$?F$	Aborts if F is not true.

Table 2.1: Hybrid Programs

Example 1 (The Linear Car Program) *One of the simplest hybrid programs models a car moving along a straight line, choosing a new acceleration $a \in \{-B, A\}$ at least once every T seconds.*

Listing 2.1: The Linear Car Program

```
1 {
2   //Choose a new acceleration accel ∈ {−B, A}
3   { accel := -B ∪ accel := A }
4   //Reset the timer.
5   c := 0;
6   //System dynamics describing linear motion for at most T time.
7   { pos'=vel, vel'=accel, c'=1 & c ≤ T }
8 }* //loop 0 or more times
```

Example 1 allows only two choices of acceleration: maximum acceleration A or maximum braking $-B$. A more realistic model allows the choice of any acceleration $accel \in [-B, A]$. This behavior is expressible in dℒ by changing line 3 of Example 1 to read:

$$\texttt{accel := *; ?(-B} \leq \texttt{accel} \wedge \texttt{accel} \leq \texttt{A).}$$

This program first allows `accel` to take on *any* value, and then immediately asserts that this new value of `accel` must be between $-B$ and A.

Semantics Hybrid Programs have a denotational semantics defined in terms of states. A state is a mapping from one set of variable assignments to another set of variable assignments. The semantics of a program, denoted $[\![\alpha]\!]$, maps a state to a set of states.

The semantics of assignment and nondeterministic choice are illustrative examples of how the semantics of hybrid programs are defined. Assignment maps each state s to a new state s' that is identical to s except for the new value of x. Formally, $[\![x := \theta]\!](s) = \{(s, s_{x\to\theta})\}$ The semantics of nondeterministic choice $\alpha \cup \beta$ maps each state s to *two* new states: one which results from executing α and the other which results from executing β. Formally,

$$[\![\alpha \cup \beta]\!](s) = [\![\alpha]\!](s) \cup [\![\beta]\!](s)$$

[1] A continuous evolution along the differential equation system $x_i' = \theta_i$ for an arbitrary duration within the region described by formula F.

2.1.2 Differential Dynamic Logic

Differential dynamic logic (dL) [86, 87, 88, 90] is a first-order multimodal logic for specifying and proving properties of hybrid programs. Each hybrid program α has modal operators $[\alpha]$ and $\langle\alpha\rangle$, which express reachability properties of program α. The formula $[\alpha]\phi$ expresses that the formula ϕ is true in all states reachable by the hybrid program α. Similarly, $\langle\alpha\rangle\phi$ expresses that the formula ϕ is true after some execution of α.

Definition 1 (Formulas) *The formulas of dL are defined as follows (with θ as terms, p as predicates, C as quantifier symbols, and ϕ, ψ ranging over dL formulas):*

$$
\begin{array}{lll}
\langle \phi, \psi \rangle \ ::= \ \theta \geq \eta & & \textit{Comparisons} \\
\quad | \quad p(\theta_1, \ldots, \theta_k) & & \textit{Predicates} \\
\quad | \quad C(\phi) & & \textit{Quantifier Symbols} \\
\quad | \quad \neg\phi \mid \phi \wedge \psi \mid \forall x \ \phi \mid \exists x \ \psi & & \textit{First-order Logic} \\
\quad | \quad [\alpha]\phi \mid \langle\alpha\rangle\phi & & \textit{Modalities}
\end{array}
$$

The grammar given by Def. 1 is the minimal. In practice, hybrid systems reachability properties formalized in dL also make use of disjunction, implication, and equivalences:

$$\theta, \psi ::= \ldots \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi$$

The formal denotational semantics of dL assign to each formula the set of states in which the formula evaluates to true. For example, $[\![x > 0 \wedge y > 0]\!] = \{s \mid s(x) > 0 \wedge s(y) > 0\}$. The intuitive meaning of formulas that do not contain modalities matches that of classical first order logic.

Example 2 *Consider a car, moving in a straight line, that must stop before arriving at a stop sign. This property is expressible in dL by the formula*

Listing 2.2: The Linear Car Program

```
A > 0 ∧ B > 0 ∧ pos < stopSignPos() ∧ safe(pos, vel, -B) →
[
  {
    //Choose a new accleeration accel ∈ [−B, A],
    //accerlating only when safe.
    { accel := -B ∪ accel := *; ?safe(pos, vel, accel) }
    //Reset the timer.
    c := 0;
    //System dynamics describing linear motion for at most T time.
    { pos'=vel, vel'=accel, c'=1 & c ≤ T }
  }* //loop 0 or more times
]pos < stopSignPos()
```

where `stopSignPos()` *is the location of the stop sign and* `safe`: $\mathbb{R}^3 \rightarrow$ `Bool` *is a to-be-defined formula describing the set of positions and velocities in which it is safe to accelerate.*

Example 2 is a reachability property of a simple hybrid dynamical system with the canonical form `init`$\rightarrow$`[{ctrl; plant}*]safe`. Here, the system begins within some initial set `init`, repeatedly executes a discrete controller `ctrl` followed by some continuous physical dynamics described by the system of differential equations `plant`, and always ends in a state that satisfies `safe`.

The premise (or precondition) of Example 2 describes a set of initial states for the hybrid dynamical system. Aside from bounds on constants, we must also assume that the car begins in a configuration such that continuously braking will bring the car to a complete stop before reaching the stop sign. The precondition `safe(pos, vel, -B)` expresses this assumption. The conclusion of the implication states that *every execution* of Example 1 ends in a state where $pos < stopSignPos()$.

Compiling Nondeterminism Notice that the model is nondeterministic and is specifically designed to capture a safety property. The controller model does not choose a single action; instead, the model describes an entire set of possible safe actions. The physical model does not choose an exact amount of time to follow the flow of the ODEs; instead, any flow up to time $0 \leq c \leq T$ is possible. Furthermore, the verification condition does not mention fuel efficiency, passenger comfort, or other important fitness criteria.

Verification conditions capture only the critical safety property of the system. Both of these modeling choices are crucial for ensuring the tractability of hybrid systems analyses; a fully deterministic description of the system would prove intractable to verify automatically. Even interactive verification might be too labor intensive in many cases.

The reinforcement learning algorithms developed in this thesis provide a way to choose efficient resolutions to this nondeterminism that satisfy safety constraints while also optimizing for other objectives. Reinforcement learning may be viewed as an optimizing compiler for nondetermistic and underspecified models. The safety constraints specified in $\mathsf{d}\mathcal{L}$ may be viewed as a constraint this optimization process.

2.1.3 The $\mathsf{d}\mathcal{L}$ Hilbert Calculus

Proving specifications such as Example 2 requires a sound set of axioms and proof rules for $\mathsf{d}\mathcal{L}$. The KeYmaera X theorem prover implements a Hilbert-style proof calculus with three components: a set of axioms, proof rules for performing uniform substitutions, and contextual rewriting proof rule.

Axioms The axioms and proof rules of $\mathsf{d}\mathcal{L}$ from [89] are enumerated in Figures 2.1 and 2.2. These axioms are designed to support compositional proofs of $\mathsf{d}\mathcal{L}$ formulas by decomposing formulas and programs into their constituent parts. For example, the axiom of nondeterministic choice $[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$ decomposes a reachability property for a nondeterministic hybrid system into two reachability properties, one for each of the constituent programs in the nondeterminstic choice. The predicate $p(\bar{x})$ may bind all variables $x \in \bar{x}$.

In typical verification tasks, the axioms in Fig. 2.1 are used to symbolically decompose regular programs and the axioms in Fig. 2.2 enable various reasoning techniques for handling ordinary differential equations. For example, we used the axioms in Fig. 2.2 to implement an Ordinary Differential Equation solver based on logical deductions and have also implemented reasoning techniques based on differential invariants [41]. The CE proof rule allows for equational rewriting of equivalent subformulas, whereas CQ and CT allow for equational rewriting of equal terms.

$$\langle\cdot\rangle\ \langle a\rangle p(\bar{x}) \leftrightarrow \neg[a]\neg p(\bar{x})$$
$$[:=]\ [x := f]p(x) \leftrightarrow p(f)$$
$$[?]\ [?q]p \leftrightarrow (q \rightarrow p)$$
$$[\cup]\ [a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$$
$$[;]\ [a;b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x})$$
$$[^*]\ [a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a][a^*]p(\bar{x})$$
$$\text{K}\ [a](p(\bar{x}) \rightarrow q(\bar{x})) \rightarrow ([a]p(\bar{x}) \rightarrow [a]q(\bar{x}))$$
$$\text{I}\ [a^*](p(\bar{x}) \rightarrow [a]p(\bar{x})) \rightarrow (p(\bar{x}) \rightarrow [a^*]p(\bar{x}))$$
$$\text{V}\ p \rightarrow [a]p$$

$$\text{G}\ \frac{p(\bar{x})}{[a]p(\bar{x})}$$
$$\forall\ \frac{p(x)}{\forall x\, p(x)}$$
$$\text{MP}\ \frac{p \rightarrow q \quad p}{q}$$
$$\text{CT}\ \frac{f(\bar{x}) = g(\bar{x})}{c(f(\bar{x})) = c(g(\bar{x}))}$$
$$\text{CQ}\ \frac{f(\bar{x}) = g(\bar{x})}{p(f(\bar{x})) \leftrightarrow p(g(\bar{x}))}$$
$$\text{CE}\ \frac{p(\bar{x}) \leftrightarrow q(\bar{x})}{C(p(\bar{x})) \leftrightarrow C(q(\bar{x}))}$$
$$\text{US}\ \frac{\varphi}{\sigma(\varphi)}$$

Figure 2.1: Axioms and proof rules of differential dynamic logic; C is a quantifier symbol, p, q are predicate symbols, and c, f, g are function symbols.

Uniform Substitutions Typical axiom systems contain a countably infinite number of axioms generated from a finite set of axiom schemata. The Hilbert axiomatization of $\mathsf{d}\mathcal{L}$ does not have axiom schemata; rather, it has a finite number of axioms, a finite number of proof rules (represented as sets of formulas), and a proof rule called Uniform Substitution (US) for performing soundness-preserving substitutions on these axioms. For example, consider the axiom of nondeterministic choice

$$[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$$

Notice that a and b are concrete atomic programs, and that $p(\bar{x})$ is a concrete predicate. The choice axiom alone is not enough to prove

$$[x := 0 \cup x := 1]x \geq 0 \leftrightarrow [x := 0]x \geq 0 \wedge [x := 1]x \geq 1$$

Uniform substitutions provide a mechanism for using axioms that mention generic programs and predicates to prove theorems that contain concrete programs and formulas.

Uniform substitutions are mappings from functions $f(\bar{x})$ to terms, predicate symbols $p(\bar{x})$ to formulas, quantifier symbols $C(\_)$ to formulas, and program constants a to programs where $\bar{x}$ is a set of variables that may be bound and $\_$ a reserved quantifier symbol of arity zero. We may also use $p(\cdot)$ where $\cdot$ means that p may mention *any* variable. The substitution $a \rightsquigarrow x := 0$ substitutes any occurrence of the program variable a with program $x := 0$. And $p(\cdot) \rightsquigarrow x \geq 0$ substitutes a predicate $p(\theta)$ with a formula $\theta \geq 0$ for any argument term θ. For example, the substitution

$$a \rightsquigarrow x := 0$$
$$b \rightsquigarrow x := 1$$
$$p(\bar{x}) \rightsquigarrow x \geq 0$$

DW $[x' = f(x) \,\&\, q(x)]q(x)$

DC $\big([x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow [x' = f(x) \,\&\, q(x) \wedge r(x)]p(x)\big) \leftarrow [x' = f(x) \,\&\, q(x)]r(x)$

DE $[x' = f(x) \,\&\, q(x)]p(x, x') \leftrightarrow [x' = f(x) \,\&\, q(x)][x' := f(x)]p(x, x')$

DI $[x' = f(x) \,\&\, q(x)]p(x) \leftarrow \big(q(x) \to p(x) \wedge [x' = f(x) \,\&\, q(x)](p(x))'\big)$

DG $[x' = f(x) \,\&\, q(x)]p(x) \leftrightarrow \exists y\, [x' = f(x), y' = a(x)y + b(x) \,\&\, q(x)]p(x)$

DS $[x' = f \,\&\, q(x)]p(x) \leftrightarrow \forall t{\geq}0 \left((\forall 0{\leq}s{\leq}t\, q(x + fs)) \to [x := x + ft]p(x)\right)$

$[':=]$ $[x' := f]p(x') \leftrightarrow p(f)$

$+'$ $(f(\bar{x}) + g(\bar{x}))' = (f(\bar{x}))' + (g(\bar{x}))'$

$\cdot'$ $(f(\bar{x}) \cdot g(\bar{x}))' = (f(\bar{x}))' \cdot g(\bar{x}) + f(\bar{x}) \cdot (g(\bar{x}))'$

$\circ'$ $[y := g(x)][y' := 1]\big((f(g(x)))' = (f(y))' \cdot (g(x))'\big)$

Figure 2.2: Differential equation axioms and differential axioms

with $\bar{x} = \{x\}$ applied to the choice axiom $[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$ produces the formula

$$[x := 0 \cup x := 1]x \geq 0 \leftrightarrow [x := 0]x \geq 0 \wedge [x := 1]x \geq 1$$

A substitution is *uniform* if it satisfies various constraints on the occurrences of free and bound variables. The uniformity constraint is required to ensure the soundness of the uniform substitution proof rule. Logical deductions in dL may appeal to the truth-preserving nature of substitutions via the US proof rule (Fig. 2.1).

Example 3 (Admissible and Clashing Substitutions) *Restricting the US proof rule to admissible uniform substitutions is necessary for preserving the soundness of the calculus. Consider the substitution and formula*

$$\sigma = \{a \rightsquigarrow x := x - 1, p \rightsquigarrow x \geq 0\}$$
$$\phi \equiv p \to [a]p.$$

If σ were admissible for ϕ (it is not!), then the US proof rule would allow a proof of $x \geq 0 \to [x := x - 1]x \geq 0$

$$\frac{\dfrac{*}{p \to [a]p}}{x \geq 0 \to [x := x - 1]x \geq 0}$$

but this formula is clearly not valid. Conversely, consider the very similar substitution σ' and the formula φ:

$$\sigma' = \{a \rightsquigarrow x := x - 1, p(\bar{x}) \rightsquigarrow x \geq 0\}$$
$$\varphi \equiv [a]p(\bar{x})$$

for $\bar{x} = (x)$. Because σ' is φ-admissible, the US proof rule allows the deduction following

$$\frac{x \geq 0}{[x := x - 1]x \geq 0}$$

via a uniform substitution on the G proof rule.

Example 3 demonstrates that the US rule is not sound for naïve substitutions. A sound calculus must restrict uniform substitutions so that substitutions which introduce unsound deductions are not permitted. For this purpose, dℒ defines when a given substitution is *admissible* for a formula and restricts the US proof rule so that the rule is only applicable when the substitution σ is ϕ-admissible. The two cases in Example 3 demonstrate why admissibility of a substitution depends upon the formula to which a substitution is applied – a substitution may be sound for one formula and unsound for another.

The slight difference between the substitutions σ and σ' in Example 3 demonstrates the significance of the difference between p, $p(x)$, and $p(\bar{x})$. These three predicate symbols have different static semantics. The first symbol (p) has a nullary predicate symbol The second ($p(x)$) has a predicate symbol where the variable x may occur freely, and the third ($p(\bar{x})$) has a predicate symbol where any $x \in \bar{x}$ may occur freely. These free variables of p continue to be permitted in its replacement. Additional free variables are allowed by the US rule under certain admissibility conditions (see [89]).

The definition of admissibility depends upon the static semantics of dℒ formulas, so this difference in the static semantics of p, $p(x)$, and $p(\bar{x})$ is crucial when determining whether a substitution is admissible.

The explication of admissibility for uniform substitutions in dℒ is critical for soundness but non-trivial (see [89] for details). Therefore, the results presented in this thesis paper abstract over the particularities by simply assuming the existence of a mechanism for checking whether a given substitution is admissible for a given formula and assuming that there is therefore a sound implementation of the US proof rule. Readers interested in a constructive definition of admissibility for uniform substitutions in dℒ may consult Platzer (in particular, Fig. 1) [89]. In this thesis we use dℒ and its axiomatization [89] as implemented by KeYmaera X for verifying hybrid systems models of cyber-physical systems.

2.2 Reinforcement Learning

Reinforcement learning (RL) [98] is one approach toward learning control policies. Reinforcement learning algorithms search for effective control strategies by taking actions, observing how these actions change an underlying environment, and then computing a numerical reward based upon some combination of the actions taken and the observed changes to the environment.

Most approaches toward reinforcement learning provide no guarantee about the safety of the learned controller or about the safety of actions taken during learning. Absence of safety guarantees becomes a crippling problem when reinforcement learning is applied to safety-critical CPSs where industry best practices demand evidence of safety, such as cars or planes [59, 94]. Incorporating verified models into safety cases for reinforcement learning-based controllers is important because testing alone is an intractable approach toward system verification and validation for systems operating in open environments, such as self-driving cars [63].

Reinforcement Learnining is an enormous research area, and many approaches toward incorporating formal methods into reinforcement learning were recently proposed. A discussion of

this and other related work on safe reinforcement learning is postponed until Chapter 4 so that this work may be presented in terms of our current and proposed contributions.

Chapter 3

Trustworthy Theorem Proving for Hybrid Dynamics

The proposed thesis will show that cyber-physical systems that use reinforcement learning for control are amenable to formal verification. Formal verification requires the use of analysis software, such as a model checker or a theorem prover. Analysis software must be trustworthy; an untrustworthy tool translates uncertainty about the system under analysis into uncertainty about the analysis tool. Analysis software must also be useful; a maximally trustworthy analysis tool that cannot even establish simple properties about simple systems is not useful in practice.

In previous work with collaborators, I developed a trustworthy interactive theorem prover for hybrid dynamical systems called KeYmaera X. The distinguishing features of KeYmaera X are a small soundness-critical core that ensures the correctness of the system, and a tactics language called Bellerophon for implementing custom hybrid systems proof construction and proof search procedures on top of the small core. This combination of a small core with a hybrid systems proof programming environment make KeYmaera X the most trustworthy and extensible hybrid systems analysis tool available today.

3.1 The KeYmaera X Core

KeYmaera X [41] is structured to maintain a trustworthy core. The KeYmaera X core is trustworthy because it is a small and simple piece of software with a defined logical foundations.

The KeYmaera X implementation of the $\mathsf{d}\mathcal{L}$ Hilbert calculus is attractive from a soundness perspective because it is simple and small. The implementation contains two main components: a text file containing verbatim copies of axioms, and a small amount of Scala code implementing the proof rules (including Uniform Substitution). All reasoning executed by the KeYmaera X theorem prover runs through this small, soundness-critical core. Unlike existing foundations of hybrid systems, the Hilbert calculus of $\mathsf{d}\mathcal{L}$ easily enables sound theorem prover implementations. Unlike existing hybrid systems analysis tools, KeYmaera X isolates all soundness-critical reasoning in a small core.

3.1.1 Proof Terms for dℒ

Differential dynamic logic has an implicit notion of proof; there is no way of saying, in the logic, that some sequence of proof rule applications results in a proof of a dℒ formula. Type theories [27, 76] and justification logics [12, 13] have explicit notions of proof; instead of only formulas φ these logics also have formulas of the form $t : \varphi$ meaning t *is a proof of* φ. Theorem proving tools built on logics with an explicit notion of proof have many attractive qualities, including an explicit syntax for exporting proofs generated by the theorem prover. In prior work we designed a Logic of Proofs for Differential Dynamic Logic, a dℒ Hilbert calculus with explicit proof terms motivated by justification logic[39]

In an effort to improve the reliability and interoperability of the KeYmaera X theorem prover, we designed the Logic of Proofs for Differential Dynamic Logic, a dℒ Hilbert calculus with explicit proof terms[39].

In separate work, Bohrer et al. implemented *verified* implementations of a differential dynamic logic proof checker in both Isabelle and Coq. Using their implementation of a variant of our proof term calculus, these researchers were able to export proofs generated by KeYmaera X. These exported proofs were then checked by these other theorem provers. The verified implementations of dℒ proof checkers increase our confidence in the soundness of the dℒ Hilbert calculus and allows interfacing with other verification efforts.

3.1.2 Challenges Introduced by the dℒ Hilbert Calculus

Unfortunately, getting real work done in this raw Hilbert calculus is nontrivial. Even simple properties have verbose proofs. For example, the proof of $[x := 0 \cup x := 1]x \geq 0$ is far from concise [39]:

$$\dfrac{\text{US}\ \dfrac{[\cup]\ \dfrac{*}{[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})}}{[x := 0 \cup x := 1]x \geq 0 \leftrightarrow [x := 0]x \geq 0 \wedge [x := 1]x \geq 0} \qquad \Delta}{[x := 0 \cup x := 1]x \geq 0}\ \text{MP}$$

with $\bar{x} = \{x\}$ where $\Delta =$

$$\text{Prop}\ \dfrac{\text{Prop}\ \dfrac{\Delta_1 \qquad \Delta_2}{[x := 0]x \geq 0 \wedge [x := 1]x \geq 0}}{([x := 0 \cup x := 1]x \geq 0 \leftrightarrow [x := 0]x \geq 0 \wedge [x := 1]x \geq 0) \rightarrow [x := 0 \cup x := 1]x \geq 0}$$

where $\Delta_1 =$

$$\dfrac{\text{US}\ \dfrac{[:=]\ \dfrac{*}{[x := t]p(t) \leftrightarrow p(x)}}{[x := 0]x \geq 0 \leftrightarrow 0 \geq 0} \qquad \text{Prop}\ \dfrac{\mathbb{R}\ \dfrac{*}{0 \geq 0}}{[x := 0]x \geq 0 \leftrightarrow 0 \geq 0 \rightarrow [x := 0]x \geq 0}}{[x := 0]x \geq 0}\ \text{MP}$$

and $\Delta_2 =$

$$\text{MP, Prop, US}\ \frac{[:=]\ \dfrac{*}{[x := t]p(t) \leftrightarrow p(x)} \quad \mathbb{R}\ \dfrac{*}{1 \geq 0}}{[x := 1]x \geq 0}$$

This lengthy derivation (which *still* elides some details!) demonstrates that although the Hilbert calculus provides a compelling target for sound theorem prover implementations, even trivial hybrid systems reachability properties require extremely verbose proofs.

3.2 Bellerophon: Tactical Theorem Proving for Hybrid Systems

Practical use of the dℒ Hilbert calculus requires careful design of interactive and automated theorem proving that leverage the simplicity of the Hilbert calculus to ensure soundness while also enabling verification of complex reachability properties for realistic models of cyber-phyiscal systems.

Bellerophon, introduced by Fulton et al. [42], is a programming language and standard library for constructing dℒ proofs in KeYmaera X. Bellerophon implements a high-level sequent calculus on top of the simpler dℒ Hilbert calculus, enabling human-readable proofs for realistic hybrid systems. The standard library also contains several automatic proof search procedures, called tactics, that can automatically prove properties about common subclasses of hybrid systems. Bellerophon's combinators provide a mechanism for composing these building blocks in to proof construction procedures and proof search algorithms. As a result, the verbose proof presented above reduces to the very small Bellerophon program invoking tactics in the Bellerophon standard library:

```
1 choiceb(1); <(
2   //[x:=0]x>=0 case
3   assignb(1); QE
4   ,
5   //[x:=1]x>=0 case
6   assignb(1); QE
7 )
```

This tactic splits the proof into one case for each program on either side of the nondeterministic choice operator ∪. In each of these cases, the assignment is symbolically executed to produce a purely arithmetic subgoal, which is then discharged using a decision procedure for real arithmetic. The primary automated theorem prover implemented in KeYmaera X, `master`, will automatically construct this tactic.

3.2.1 Encoding General Results about Dynamical Systems in KeYmaera X

Despite the relative simplicity and small size of the KeYmaera X core, Bellerophon and KeYmaera X are capable of expressing many properties about continuous dynamical systems.

One significant tactic is the axiomatic ODE solver, which uses the axioms and proof rules of dℒ to prove the existence of solutions to a subset of linear ordinary differential equations.

The ODE solver, written by the author of this proposal, provides validated solutions to differential equations and is one of the largest tactics in the Bellerophon standard library. The ModelPlex algorithm [79], which generates runtime monitors for hybrid systems, is implemented as a Bellerophon tactic and makes essential use the axiomatic ODE solver.

KeYmaera X and dℒ are designed for hybrid program verification, not general-purpose mathematics. Therefore, the variety of results about dynamical systems that are encodable using a combination of dℒ and Bellerophon tactics is often surprising. The Axiomatic ODE Solve is a great example of this surprising expressiveness. The existence of closed-form solutions for linear systems is not directly expressible in dℒ. However, a schema of dℒ formulas describing all such systems combined with a Bellerophon tactic that can prove the invariance of a solution for any system in this schema produces the same result. The dℒ formulaic schema is a theorem about a general class of dynamical systems, and the Bellerophon tactic generates a proof for each formula in the schema.

Bifurcation theory provides another nice example of the often surprising expressiveness of KeYmaera X and Bellerophon. The existence of a bifurcation point in the 1D saddle-node [69] – and the location of the fixed points of the system on either side of this bifurcation – can be encoded in dℒ and Bellerophon. The following dℒ formula encodes the fact that some fixed point f exists for any choice of a parameter $r \leq 0$: $r \leq 0 \rightarrow \exists f(x = f \rightarrow [x' = r + x^2]x = f)$

The proof of this property must identify the bifurcation point at $r = 0$ and discover the corresponding fixed points ($f = -\sqrt{-r}$ when $r < 0$ and $f = 0$ when $r = 0$):

The equilibrium points on either side of the 1D saddle-node bifurcation[1]

```
/* Move r <= 0 to the antecedent */
implyR(1);
/* Introduce and prove r=0 ∨ r<0 so that we can split the
 * rest of our analysis along this bifurcation. */
cut(r=0 ∨ r<0); <(hideL(-1), hideR(1) ; QE);
/* Split the proof. */
orL(-1); <(
  /* CASE 1: r=0 */
  existsR({`0`}, 1);                    /* choose f=0 */
  implyR(1);
  dG({y'=-xy}, yx=0 ∧ y>0, 1); /* x=0 ↔ ∃y(yx=0 ∧ y>0) */
  existsR({`1`}, 1);                    /* choose y≠0; e.g., y=1 */
  /* Consider y*x=0 and y>0 differential invariants separately */
  boxAnd(1); andR(1); <(
    /* y*x=0 is differentially inductive because:
       (yx)'=0 ↔ y'x+x'y=0 and y'x+x'y=0 ↔ -xy^2+r+x^2=0
       (recall: r=0)
    */
    dI(1)
    ,
    /* y>0 case needs an extra cut; see note on differential ghosts. */
    dG(z'=x/(2z), z^2y=1, 1);
    existsR({`1`}, 1);
    dI(1)
  )
  ,
  /* CASE 2: r<0 */
```

```
28   /* introduce new variable s = sqrt(-r) and prove s exists. */
29   cut(∃s.r = −s^2); <(nil, hideR(1) ; QE);
30   /* Some cleanup work about s */
31   existsL(-2) ; existsR(−s, 1) ; implyR(1) ;
32   /* See note on differential ghosts. */
33   dG({y' = (s − x)y}, y(x + s) = 0 ∧ y > 0, 1) ; existsR({`1`}, 1) ;
34   boxAnd(1) ; andR(1); <(
35     dI(1),
36     dG({z' = (x−s)/(2z)}, z^2 y = 1, 1) ; existsR({`1`}, 1) ; dI(1)
37   )
38 )
```

The main idea is that many results about dynamical systems which are not traditionally thought of as reachability properties are none-the-less expressible using a combination of dℒ and Bellerophon even though KeYmaera X is not explicitly designed to support general-purpose mathematics. I.e., a *combination* of Bellerophon tactics and schematic dℒ formulas can express general results about dynamical systems. This fact enables our proof-producing axiomatic ODE solver without any direct appeal to Picard–Lindelöf. With a bit of ingenuity, KeYmaera X's approach toward hybrid systems analysis is surprisingly powerful.

3.2.2 Related Work on Formal Methods for Hybrid Dynamics

Trustworthy and productive hybrid systems theorem proving requires a small soundness-critical core, automation specific to hybrid systems, and a mechanism for composing this automation. Even though these ingredients can be found scattered across a multitude of theorem provers, their combination provides a novel a tactical theorem proving technique for hybrid systems. Table 3.1 compares several tools along the dimensions that we identify as crucial to productive hybrid systems verification (SC indicates a soundness-critical dependency on user-defined tactics or on an external implementation of a more scalable arithmetic decision procedure).

Table 3.1: Comparison to related verification tools and provers

Tool	Small Core	HS Library	HS Auto	Scriptable	External Tools
KeYmaera X	Yes	Yes	Yes	Yes	SC
Hybrid Systems Tools[2]	No	No	Yes	No	SC
Theorem Provers[3]	Yes	No	No	Yes	No
dℒ in Isabelle,Coq[20]	Yes	Yes[4]	No[5]	Yes	No
KeYmaera 3	No	Yes	Yes	SC	SC

[2]E.g., Ariadne[19], C2E2 [32], Charon [10], Checkmate [81], CORA [5], d/dt [14], sReach/dReach/dReal [44, 66, 101], FLOW* [23, 24], the MATLAB Hybrid Toolbox [16], HyReach [75], hyPro [56, 96], HyTech [8, 55], PHAver [36], Kronos [26], Ptolemy [22, 34], S-Taliro [11], SpaceEx [38], UPPAAL [17, 18, 25, 70], and others.

[3]E.g., Coq [77], Isabelle [83], HOL [50, 97], and Lean [28, 29].

[4]via encodings in dℒ

[5]via KeYmaera X proof term extraction discussed in Section 3.1.1

Logics of Hybrid Systems Theorem provers for hybrid dynamics use deduction in a formal calculus to establish reachability properties about a programming languages model of hybrid dynamics. Differential Dynamic Logic and Hybrid CSP [73] both fit this paradigm. The first theorem prover for $\mathsf{d}\mathcal{L}$ was KeYmaera[6] [91]. KeYmaera was built as an extension to the KeY system [3] and is compared to KeYmaera X in Table 3.1.

General-Purpose Theorem Proving Differential dynamic logic is a specialized logic for stating and proving reachability properties of hybrid dynamical systems. Several researchers have proposed building a theory of hybrid systems within existing theorem provers. The most complete approach is that of Bohrer et al. [20], whose work is compared with KeYmaera X in Table 3.1. The approach of Bohrer et al. leverages purpose-built systems such as KeYmaera X while building an interface with the larger theories available in tools like Coq and Isabelle.

Some libraries implemented within general-purpose theorem provers are relevant to the analysis of continuous systems. For example, Immler and Hölzl formalize Picard-Lindelöf in Isabelle [57, 58]. Foster et al. [35] recently proposed a new approach toward verifying hybrid systems using a Hoare logic implemented in Isabelle. Foster et al. focus on unifying theories and their current exploration of interactive and automted hybrid systems verification focuses on demonstrating that these properties are encodable within their framework.

Delta Decidability δ-decidability [43] introduces a relaxed notion of correctness for real arithmetic decision procedures. The dReal [66] and dReach [44] tools leverage this new notion of decidability to obtain verification results about hybrid systems.

Timed Automata Timed automata express a restricted subset of hybrid dynamics in which continuous dynamics are restricted to a finite set of real-valued resettable clocks. This setting does not allow expression of the rich continuous dynamics typically found in cyber-phyiscal systems, but is more tractable. Tools such as UPAAL [17, 18, 25, 70] and [26] are able to automatically model-check properties of (networks of) timed automata.

Hybrid Automata Hybrid automata [54] are a representation of hybrid dynamics based on automata theory. Several approaches toward model checking for hybrid automata have been proposed and implemented; all of these approaches attempt to compute a set of a states that are reachable from an initial configuration. The reachable set is often over-approximated to make automated analysis tractable. Over-representations might be represented as ellipsoids, convex polytopes, zonotopes, and/or support functions. Many of these approaches consider subsets of hybrid dynamics; e.g., PHAver [36] considers linear hybrid automata and SpaceEx [38] considers piecewise affine dynamics. KeYmaera X currently provides less automation than these tools but is also not restricted to a subclass of hybrid systems[7]. KeYmaera X also differs from these tools

[6]Not to be confused with KeYmaera X, a clean-slate implementation that shares no code with KeYmaera, implements a different core calculus, and takes a different approach toward both interactive and automated theorem proving.

[7] This limitation is one of the implementation rather than the theory. Section 3.2.1 discusses how automation can be built on top of KeYmaera X using a combination of schematic $\mathsf{d}\mathcal{L}$ formulas and Bellerophon tactics.

because it provides a robust interactive theorem proving environment; see Fulton et al. [42] Mitsch and Platzer [78].

Among tools based on hybrid automata, Ariadne[19] is notable because – like KeYmaera X– Ariadne aspires to provide both an alaysis tool and a development environment for for constructing new hybrid systems analyses. Unlike KeYmaera X, Ariadne does not isolate soundness-critical reasoning from user-defined verification algorithms.

Conclusion

KeYmaera X is a novel hybrid systems theorem prover that provides: *1)* a small foundational core; *2)* a library of high-level primitives automating common deductions (e.g., computing Lie Derivatives, computing and proving solutions of ODEs, propagating quantities across dynamics in which they do not change, automated application of invariant candidates, and conservation/symmetry arguments); and *3)* scriptable heuristic search automation. KeYmaera X can also automatically generate ODE invariants [92] and generate code via a verified toolchain [21].

Chapter 4

Verifiably Safe Reinforcement Learning

Autonomous vehicles should be deterministic, efficient, and safe. Chapter 3 introduced our approach toward for ensuring safety for nondeterministic controllers. Verified hybrid programs distinguish between safe and unsafe actions in each state, but do not single out the specific action that should be taken in order to achieve an objective other than safety. The car in Example 2 has two different options in most of the state space, and can *always* choose to activate its brakes. Although KeYmaera X tells us that this controller is safe, the resulting theorem does not explain the sequence of actions that will help the car actually *reach* the stop sign.

KeYmaera X establishes a safe set of actions, but does not explain which of these actions ought to be taken.

Fortunately, reinforcement learning solves exactly the problem that KeYmaera X does not solve. Given an appropriate reward signal, a reinforcement learning algorithm could tell us which sequence of actions should be taken so that the car reaches the stop sign without over-shooting the stop sign. However, learning this control policy might require thousands of iterations before the algorithm eventually learns how to avoid over-shooting the stop sign. This situation is unacceptable in the real world, where over-shooting a stop sign might result in property damage or even loss of life. In addition, the exploration of obviously unsafe or unfeasible policies also contributes to the notorious data-inefficiency of reinforcement learning algorithms.

In prior work [40], we suggested an approach that combines the best of both worlds. In our approach, KeYmaera X constrains the search space for a reinforcement learning algorithm so that only safe actions are taken, while reinforcement learning is free to search the subset of safe policies for a policy that achieves objectives other than safety. Our approach, called justified speculative control (JSC), ensures that verification results transfer to policies learned via reinforcement learning. This approach has the nice auxiliary advantage of increasing the data efficiency of reinforcement learning, because obviously unsafe actions do not need to be explored during reinforcement learning.

KeYmaera X, like all formal methods tools, can only provide guarantees relative to a model of the world. When the model is inaccurate, guarantees disappear. For example, if the stop sign in Example 2 begins to move forward[1], then the car has left the modeled portion of the state space and might over-shoot the stop sign even though its controller is verified. We call the set of states

[1] E.g., consider a situation in which the stop sign is held by a partially occluded construction worker along a work zone and the construction worker begins to walk forward.

where the model is accurate the *model space*; traditional verification results only apply within their model space.

Designing a safe autonomous system requires either building a perfect model of the world, or else ensuring that the system will behave well outside of model space.

This section explains how justified speculative control guarantees safety within model space, discusses one approach toward controlling well outside of model space, and discusses other approaches toward safe and verifiable reinforcement learning. The work presented in this chapter was previously published by Fulton and Platzer [40]. We also discuss the limitations of this approach; those limitations will motivate the proposed work discussed in Chapter 5.

4.1 Justified Speculative Control

Justified speculative control ensures safety within model space and suggests one approach toward leveraging verification conditions to control well outside of model space. The algorithm takes as input a $\mathsf{d}\mathcal{L}$ formula with the form

$$\vdash \texttt{init} \rightarrow [\{\texttt{ctrl};\texttt{plant}\}^*]\texttt{safe}$$

where

$$\texttt{ctrl} \equiv ?\texttt{guard}_0;\texttt{act}_0 \cup \cdots \cup ?\texttt{guard}_n;\texttt{act}_n$$ [2]

From this model, JSC uses the Bellerophon ModelPlex tactic to generate a controller monitor and model monitor. The controller monitor specifies, for each state, which actions are permitted by the guards. The model monitor specifies, for each (s_{pre}, a, s_{post}) tuple, whether taking the action a in state s_{pre} and then following the flow of the differential equations could lead to state s_{post}.

Justified Speculative Learning

```
1  JSCGeneric(init, (S,A,R,E), choose, update, done, CM, MM) {
2    prev := curr := init;
3    a0   := NOP;
4    while (!done(curr)) {
5      if (MM(prev, a0, curr))
6        u := choose({a ∈ A | CM(a,curr)});
7      else
8        u := choose(A);
9      prev := curr;
10     curr := E(u, prev);
11     update(prev, u, curr);
12   }
13 }
```

JSC uses model monitors to constrain the learning process so that the reinforcement learning algorithm only takes safe actions while in model space. Listing 4.1 provides pseudo-code for the basic algorithm. Within model space, JSC acts like a sandbox; each $\texttt{act}_i$ discrete state update

[2] the guard conditions are optional; i.e., some actions can be unguarded.

in the model is an action in an underlying Markov Decision Process and each $\texttt{guard}_i$ specifies the states in which this action should be available to the optimization procedure.

Justified speculative control also considers situations where the system exits model space. When the system leaves model space, JSC may *speculate* by choosing actions that are not allowed in model space. However, during speculation, the system uses a real-valued variant of the model monitor in order to speculatively guide the system back into modeled state space. When exiting model space, justified speculative control allows possibly unsafe actions but abandons all goals other than getting back to model space.

This approach toward controlling outside of model space works well for intermittent deviations, such as a stop sign that moves forward a little bit once and then maintains its new position. However, this approach comes with no guarantees – it is simply a heuristic, albeit one based on a verification result, for controlling well in unmodeled space. These limitations are discussed at greater length in Section 4.3.

Advantages of Justified Speculative Control Leveraging an existing model and control policy substantially increases the data efficiency of JSC relative to naïve learning. Unlike traditional verification approaches, JSC explains how to optimize for goals that are not safety-crtiical. Verification results transfer not only to final policies, but also to the learning process itself. This might enable the use of online reinforcement learning in production environments. This combination on improved data efficiency and safety guarantees for the training process itself might enable the use of reinforcement learning in real, non-simulated environments.

4.2 Related Work on Safe and Verifiable Reinforcement Learning

Obtaining verification results for Artificial Intelligence is an emerging area of interest [95], but there is a rich history of research on safe control in the absence of perfect models.

There are a myriad of approaches toward safe reinforcement learning that do not take advantage of formal verification, many of which are summarized by García and Fernández [45]. García and Fernández decompose these approaches into two broad categories: modification of the optimality criterion, and modification of the exploration process. In this section, we compare our approach to these approaches, following García and Fernández's taxonomy [45]. We also include discussion of new related work that postdates the García and Fernández survey.

Our work makes two contributions relative to this prior work. First, we leverage hybrid systems verification results and runtime monitor synthesis to appropriately sandbox the exploration process, instead of relying on more ad-hoc sources of knowledge about how to act safely. The chain of evidence transfers from a high-level model to runtime monitors and ultimately to the reinforcement learning process via the theorems presented in this paper. Second, we distinguish between *optimizing among known safe policy options* and *speculating about portions of the state space that are not a priori modeled*. This distinction is crucial to determine what level of speculation should be allowed, and when.

When compared to existing approaches to reinforcement learning, our approach either 1)

suggests a way to strengthen the existing approach by incorporating not just a known-safe policy but a *formally verified* safe policy; or 2) is possibly compositional with the existing approach (by further modifying our exploration process to perform more robust decision making when the model monitor is already violated).

Constrained Criterion

Safety properties can be thought of as constraints on an optimization process. Many researchers have suggested various ways of characterizing these constraints and for optimizing in the presence of constraints.

Several approaches of this variety directly constrain the policy search space by only allowing agents to choose from a set of control actions that are conjectured[3] to be safe [47, 62, 80]. Altman's Constrained Markov Decision Processes provide a theoretical framework for characterizing constrained optimization of dynamical systems [6]. Recent work by Achiam et al. [2] leverages this framework to constrain learning for high-dimensional control problems. Held et al. [53] characterize safety in terms of thresholds on damage. Another approach toward safe reinforcement learning adopts worst case criterion [52] or risk-sensitive criterion [82, 99], in which the optimization criterion is modified to reflect safety concerns.

Comparison with JSC and Proposed Work Each of these approaches assumes a tractable characterization of safety constraints, such as a cost function or a characterization of forbidden regions. Constraints on cyber-physical systems are often fundamentally dynamical. For example, safety constraints for collision avoidance algorithms must reference the dynamics of interlocutors.

Constrained criterion approaches require the system designer to explicitly state safety constraints, usually as purely arithmetic properties that do not mention e.g., differential equations or discrete dynamics. However, the problem of coming up with good constraints is analogous to the problem of coming up with good reward functions. Although it is often easy to state the dynamical systems interpretation of a constraint (e.g., the robot does not run into the interlocutor whose movement is governed by given ODEs), translating these dynamical descriptions into a arithmetic descriptions that are useful as costs or constraints for an optimization algorithm is non-trivial and error prone. The problem of reducing reachability properties about dynamical systems to unquantified real arithemtic constraints is ultimately reducible to the problem of full-blown hybrid systems theorem proving[4]; in fact, this is exactly the methodolgy that KeYmaera X takes toward verification.

Unlike both JSC and our proposed approach, none of this work suggests what should be done when it becomes clear that the constraints originated from faulty assumptions about the dynamics of the system. Our proposed approach links a hybrid dynamical system to each runtime constraint. We propose introducing explicit model updates that separate the *assumptive* components of a hybrid system from the *prescriptive* models of a hybrid system. This allows us update model specifications in a way that captures and corrects for incorrect assumptions and then syn-

[3] but not formally proven
[4] which is undecidable

thesize new monitoring conditions that incorporate these updated assumptions. Our proposed approach is also backed by formal proofs.

KeYmaera X and ModelPlex are able to automatically and correctly reduce reachability properties of hybrid dynamical systems to formulas of real arithmetic. Relative to these constrained criterion approaches, we introduce a methodology – based on hybrid systems theorem proving – for correctly and often automatically generating useful optimizing constraints from statements about reachability properties of hybrid dynamical systems.

Initial Knowledge Approaches

Another approach toward safe learning attempts to initialize the learner in order to direct policy exploration away from unsafe states [31]. Our approach is analogous; the guards on control decisions in our hybrid programs are a form of initial knowledge. Unlike most approaches that leverage initial knowledge, we explicitly characterize the difference between states where our initial knowledge is trustworthy from states where our initial knowledge is not trustworthy.

Analysis of Learned Policies

Constrained criterion and initial knowledge approaches ensure safety by modifying the learning/optimization process. An alternative approach toward safe reinforcement learning suggests analyzing the policies that are constructed from a learning process. For example, Katz et al. introduce an SMT solver for analyzing deep neural networks and apply this technique to analysis of a DNN implementation of collision avoidance for aircraft [64]. Wang et al. take a similar approach toward reasoning about security properties [102].

Analyzing learned models is attractive when the modifying the learning process is intractable or impossible, which is often the case in practical settings where a cutting-edge algorithm is not trivial to modify with a constrained criterion, or especially when safety analysis is treated as a post-hoc concern.

Although recent successes demonstrate the feasibility of analyzing very large learned policies, the curse of dimensionality tells use that these approaches will always require clever optimization. Our current and proposed approaches leverage the insight that safety problems are often of lower dimension than optimization problems. Furthermore, unlike our approach, analyzing learned policies does not provide guarantees about the learning process itself.

Verified Perception

Some recent results focus on obtaining formal guarantees about neural networks used for perception. Examples include the predictor/verifier framework of Dvijotham et al. [33] and VeriVis by Pei et al. [85]. Our work focuses on control, not perception; however, finding ways to unify work on safe perception with work on safe control is a promising direction for future research agendas.

Formal Methods for Linear Temporal Logic

Alshiekh et al. and Hasanbeig et al. each propose an approach toward logically constraining reinforcement learning based on Linear Temporal Logic (LTL) [4, 51]. Like JSC, these approaches have a logical foundation. Unlike JSC, these approaches do not use a logic capable of expressing hybrid dynamical systems and do not explain what to do when a model deviation is detected. Therefore, these approaches do not solve the problem this thesis proposes addressing: providing verifiably safety guarantees for *cyber-physical systems* when reality deviates from modeling assumptions.

The use of LTL limits the applicability of these approaches in cyber-physical systems, but does succinctly capture many constraints on discrete or discretized planning and optimization problems. One fruitful avenue of future work could combine these approaches with JSC, using LTL-based approaches for expressing constraints on global, coarse-grained planning while using $\mathsf{d}\mathcal{L}$ for expressing constraints on more local, fine-grained control decisions.

4.3 Limitations of Justified Speculative Control

The justified speculative control algorithm discussed in this chapter presents one approach toward verifiably safe learning. Reinforcement learning is sandboxed by monitors derived from verification results, and a quantitative version of the monitor provides a signal that appears to give reasonable heuristics in unmodeled environments. However, this approach has two significant limitations.

Limited Applicability Justified speculative control is currently limited to simple hybrid systems with few state variables and linear continuous dynamics. There are two reasons for this limitation. First, the ModelPlex tactic can only generate exact monitoring conditions when the differential equations have a solution that exists in the first order theory of real closed fields[5]. Second, justified speculative control assumes it is possible to enumerate the entire state space during the training phase. These two limitations are crippling: JSC is not applicable to hybrid systems with nonlinear continuous dynamics and does not work well for models with large state spaces. Because of these limitations, the justified speculative control algorithm introduced by Fulton et al. [40] is only validated on very simple models.

Lack of Guarantees for Unmodeled Environments Ideally, an approach toward verified safe autonomy should come with guarantees of safety and/or optimality even in cases where the original model is slightly wrong. Small deviations from the model should not typically lead to catastrophic failures.

Overcoming these limitations requires two new contributions. First, we must expand justified speculative control to large and/or continuous state and action spaces. Second, we must find a way to identify and resolve systematic disparities between models and reality. And when a more accurate model is discovered, this new model must be incorporated into justified speculative control.

[5] in particular, these solutions may not contain any exponential or trigonometric functions

Chapter 5

Future Work and Proposed Contributions

Justified speculative control takes a first step toward verifiably safe learning by transferring safety guarantees to learned controllers within model space and by leveraging insights from verification during exploration outside of model space. The proposed work will address the two limitations of JSC discussed in Section 4.3 by introducing *verification-preserving model updates* and leveraging these updates during the learning process.

We propose developing a set of hybrid systems model updates that capture common ways in which reality might deviate from modeling assumptions. For example, recall the situation where an ostensibly static obstacle (such as the stop sign in Example 2) begins moving toward the system under control[1].

Ideally, the robot should update its model of the world by introducing a new set of differential equations that describe how the ostensibly static obstalce is moving. The robot should then update its controller so that the existing safety constraint is preserved whenever possible. This pair of updates should be coupled so that systematic changes to the robot's model of the environment correspond with changes to the robot's control logic. Ideally, the pair of updates should also come with a formal correctness proof: if the original system avoided the obstacle, and the obstacle dynamics changed in a given way, then the controller should be changed in such a way that the original safety constraints are preserved.

The proposed work will realize this vision by introducing a set of model updates, inventing a combination of offline and online verification for ensuring that model updates preserve verification results, associating model identification algorithms with each of these verified model updates, and explaining how to obtain safety results in the presence of model deviations by incorporating model identification algorithms into justified speculative learning.

The remainder of this proposal is organized as follows.

Section 5.1 introduces the conceptual foundations of verifcation-preserving model updates.

Section 5.1.1 demonstrates the proposed interaction between offline verification, online verification, model identification, and reinforcement learning in a concrete example.

Section 5.1.2 discusses other possible updates.

Section 5.1.3 discusses two possible case studies for validating this approach.

[1] e.g., a stop sign held by a partially occluded construction worker

Section 5.2 introduces more related work and places our proposed work in context. Section 5.3 succinctly summarizes all three of our related work discussions.

5.1 Verification-Preserving Model Updates

Engineers often modify models in response to new data. For example, an engineer might modify a hybrid system $\{\texttt{ctrl}; x' = v\}$ by adding a constant additive term $\{\texttt{ctrl}; x' = v + n\}$ or a constant multiplicative factor $\{\texttt{ctrl}; x' = nv\}$. In most cases, making this change to the environmental model will necessitate a corresponding change to the controller $\texttt{ctrl}$.

In some cases, such as the addition of constant additive or multiplicative terms, our engineer might also introduce some new assumptions about the bounds on the parameter n. In these cases, both the hybrid program and the dL safety specification will change. For example, the engineer might change

$$\texttt{init} \rightarrow [\texttt{ctrl}; x' = v]\texttt{safe}$$

by adding a new assumption that the parameter n is strictly positive:

$$\texttt{init} \wedge n > 0 \rightarrow [ctrl; x' = nv]\texttt{safe}$$

These are examples of *model updates*. This thesis will consider the problem of automatically generating model updates during reinforcement learning. The thesis will consider the canonical form

$$\texttt{init} \rightarrow [\{ctrl; plant\}^*]\texttt{safe}$$

where the formulas $\texttt{init}$ and $\texttt{safe}$ do not contain modalities or quantifiers, $\texttt{ctrl}$ is a loop-free discrete program, and $\texttt{plant}$ is a system of ordinary differential equations (possibly nonlinear). A *model update* is a systematic modification to each part of the model

$$\begin{aligned} \texttt{init}_1 &\rightsquigarrow \texttt{init}_2 \\ \texttt{ctrl}_1 &\rightsquigarrow \texttt{ctrl}_2 \\ \texttt{plant}_1 &\rightsquigarrow \texttt{plant}_2 \\ \texttt{safe}_1 &\rightsquigarrow \texttt{safe}_2 \end{aligned}$$

such that if $\vdash \texttt{init}_1 \rightarrow [\{\texttt{ctrl}_1; \texttt{plant}_1\}^*]\texttt{safe}_1$ then $\vdash \texttt{init}_2 \rightarrow [\{\texttt{ctrl}_2; \texttt{plant}_2\}^*]\texttt{safe}_2$ as well.

Example 4 *Consider the valid formula*

$$\vdash x > 0 \rightarrow [v := *; ?v > 0; x' = v]x > 0$$

The following hybrid program mutation adds a multiplicative term to the ODE (with no modifications to other parts of the model):

$$x' = v \rightsquigarrow x' = nv$$

This model mutation is not verification-preserving because

$$\nvdash x > 0 \rightarrow [v := *; ?v > 0; x' = nv]x > 0$$

However, several extended model mutations are verification-preserving. For example, we can place bounds on the parameter n in the initial conditions

$$x' = v \rightsquigarrow x' = nv$$
$$x > 0 \rightsquigarrow x > 0 \land n > 0$$

resulting in a valid formula

$$\vdash x > 0 \land n > 0 \rightarrow [v := *; ?v > 0; x' = nv]x > 0$$

We could also modify the controller to insert a test that asserts $n > 0$ before following the flow of the differential equation:

$$x' = v \rightsquigarrow x' = nv$$
$$v := *; ?v > 0 \rightsquigarrow v := *; ?v > 0; ?n > 0$$

resulting in a valid formula

$$\vdash x > 0 \rightarrow [v := *; ?v > 0; ?n > 0; x' = nv]x > 0$$

Defining Model Updates In general, we define model updates as modifications to dℒ formulas with the form:

$$\texttt{init} \rightarrow [\{\{\texttt{ctrl\_0} \cup \texttt{ctrl\_1} \cup \cdots \cup \texttt{ctrl\_n}\}; \texttt{plant}\}^*]\texttt{safe}$$

We sometimes also allow the insertion of a timer

$$\texttt{init} \rightarrow [\{\{\texttt{ctrl\_0} \cup \texttt{ctrl\_1} \cup \cdots \cup \texttt{ctrl\_n}\}; c := 0; \texttt{plant,c'=1 \& P}\}^*]\texttt{safe}$$

and/or the insertion of non-alternating interleavings of discrete and continuous dynamics:

$$\texttt{init} \rightarrow [\{\texttt{ctrl} \cup \texttt{plant}\}^*]\texttt{safe}$$

The first new contribution in this thesis is a set of model updates and corresponding Bellerophon programs that translate safety proofs for the original model into safety proofs for the new model. These model updates will capture common ways which which models might change in response to observed data (e.g., a presumably stationary object might move, requiring use to update our model of the object and the collision avoidance protocols relevant to that object). The resulting set of model updates will enable us to address the core limitations of JSC identified in this chapter.

5.1.1 Obstacle Updates: An Example

Perhaps the most pervasive safety constraint is collision avoidance. One approach toward collision avoidance is maximally pessimistic, assuming that all obstacles are capable of quick acceleration and high speed in any direction. This assumption will ensure safety, but is impractical

because the system will make no progress. Thus, practical collision avoidance involves a difficult tradeoff between staying conservative enough for safety, but optimistic enough to make progress toward a goal.

This section demonstrates how we plan to use model updates for verifiably safe learning outside of model space. We introduce a set of verification-preserving model updates that correspond to a set of obstacle models. We then discuss how a combination of offline and online learning can be used to instantiate these models at runtime based upon sensor data using a model identification algorithm. Finally, we incorporate this approach into the JSC algorithm. The result is an approach toward switching between obstacle models based upon sensor data in a way that ensures collision avoidance as long as we identify the correct model.

The Initial Model We recall the car, moving in a straight line, that must stop before arriving at an obstacle in the road. This property is expressible in dℒ by the formula

Listing 5.1: The Linear Car Program

```
A>0 ∧ B>0 ∧ T>0 ∧ obsPos-pos > minBrakingDistance(vel) →
[{
  {
    accel := -B;
    ∪
    ?safe(pos, vel, accel, obsPos);
    accel := A;
  }
  c := 0;
  { pos' = vel, vel' = accel & vel <= 0 & c <= T }
}*@invariant(obsPos - pos > minBrakingDistance(vel))]pos < obsPos
```

where `obsPos` is the location of the obstacle, `T` is an upper bound on the amount of time that passes between executions of the controller, and the functions are defined as follows:

Listing 5.2: Function Definitions for the Linear Car Program

```
  /*Minimum distance required to brake if traveling with velocity vel.*/
  R minBrakingDistance(R vel) = vel^2/(2B).
  /* new velocity after n time units when accelerating with accel from
     current velocity vel. */
  R newVel(R vel, R n, R accel)  = vel+n·accel.
  /* The distance the car travels with initial velocity vel and constant
     acceleration accel. */
  R travelDistance(R vel, R accel) = T^2accel/2+T·vel.
  /* Predicate defining when it's safe to accelerate with accel. */
  B safe(R pos, R vel, R accel, R obsPos) <-> (
    obsPos - (pos + travelDistance(vel, accel)) >
      minBrakingDistance(newVel(vel,T,accel))
  ).
```

The automatic tactic `master` implemented in KeYmaera X proves this property automatically using the loop invariant annotation.

Obstacle Model Updates

We introduce two obstacle model updates. The first allows the obstacle to move according to a set of differential equations and the second allows the obstacle to appear and disappear as a discrete event.

The Moving Obstacle The first update introduces some simple obstacle dynamics that allow the car to move forward[2]:

$$\begin{aligned}
\texttt{init} &\rightsquigarrow \texttt{init}_{\texttt{const}} \\
&\quad \wedge \texttt{obsPos-pos > minBrakingDistance(vel - obsVel)} \\
&\quad \wedge \texttt{obsVel} \geq 0 \\
\texttt{safe}(\cdots) &\rightsquigarrow \texttt{(obsPos + travelDistance(obsVel, 0))} \\
&\quad \texttt{-(pos + travelDistance(vel, accel))} \\
&\quad \texttt{> minBrakingDistance(newVel(vel,T,accel))} \\
\texttt{plant} \,\&\, \texttt{domain} &\rightsquigarrow \texttt{plant}, \texttt{obsPos}' = \texttt{obsVel} \,\&\, \texttt{domain}
\end{aligned}$$

Notice that the controller is updated by changing the definition of the safe predicate, expanding the set of states where it is safe to accelerate. The corresponding proof update relaxes the loop invariant used by `master`:

$$\texttt{loopInvariant} \rightsquigarrow \texttt{obsPos - pos > minBrakingDistance(vel - obsVel)}$$

This model update is pre-computed at design time and can be applied automatically for certain classes of models. Like many tactics, a full definition of the applicability of this update is non-trivial and best described by the concrete implementation[3].

The Jumping Obstacle We also consider the case where an obstacle occasionally disppears, only to reappear further up the road:

$$\texttt{ctrl} \rightsquigarrow \texttt{ctrl}; \texttt{jmp} := (-\infty, 0]; \texttt{obsPos} := \texttt{obsPos} + \texttt{jmp}$$

where `ctrl` stands in for Lines 6–8 of the original model.

The rest of the model and the proof remain the same. This model update is pre-computed at design time and can be applied automatically for certain classes of models[4].

Identifying the Correct Model

Differentiating between an arbitrary discrete jump and smooth change in location requires multiple obsevations. We treat model identification as a simple classification task that observes a sequence of sensor readings and determines which model provides the most accurate explanation for the observed dynamics. We also allow for the possibility that no model provides a good explanation.

[2]The definitions used in this listing are presenting in Listing 5.2.

[3]For details, see `edu.cmu.cs.ls.keymaerax.vpmu.Static2Linear` in the KeYmaera X source code.

[4]See `edu.cmu.cs.ls.keymaerax.vpmu.Static2Jump` for details.

Incorporating Obstacle Model Updates into JSC

Incorporating these model updates into JSC requires computing each model update and its associated ModelPlex monitors offline. JSC is then augmented with a model identification algorithm and chooses its sandboxing constraint based upon the currently accepted model. Whenever the model identification algorithm does not identify an acceptable model, JSC attempts to abort by braking to a full stop.

5.1.2 Other Proposed Model Updates

The proposed thesis will extend the example in Section 5.1.1 to consider nonlinear dynamics and continuous action spaces, with corresponding updates to the JSC algorithm. In addition to extending our exploration of model updates for obstacle avoidance, we will also discuss several other model updates. Examples include updates to the system's own dynamics (e.g., introduction of bounded noise and disturbance in state variables during actuation), updates to the system's control logic (e.g., parameter insertion, parameter instantiation, refinement of Taylor expansions used to estimate the position of obstacles in the environment), updates to other agents in the system (e.g., obstacle updates), and/or updates to portions of the environment without agency (e.g., unannounced lane closures in lane keeping).

For each update, we will introduce an appropriate model identification algorithm for deciding when to use the update and incorporate the update procedure into a reinforcement learning algorithm. Some of these updates (e.g., parameter insertion and instantiation) will require a light-weight theorem prover capable of applying uniform substitutions, instantiating quantifiers, and performing simple propositional proof steps *at runtime*. Through a combination of offline verification of model updates and online validation of concrete instantiations of these updates, we will maintain a chain evidence that allows composition of schematic updates at runtime.

5.1.3 Evaluation

We propose evaluating the model update approach on two sets of environments: one motivated by autonomous vehicles and the other motivated by a safety-critical wearable device. We have chosen these two environments because of the prevalence of modeling uncertainty in these domains.

Collision Avoidance for an Autonomous Car We propose developing a set of model updates for the kinematic model of an autonomous vehicle similar to the model described by Paden et al. in [84]. We will consider a multi-modal tracked environment in which the perception layer might change its judgement about whether an obstacle is static, a pedestrian, or another vehicle. We will also investigate control techniques that satisfy safety constraints for multiple obstacle models simultaneously. This work will require developing a lightweight online verifier capable of performing uniform substitutions and quantifier instantiations at runtime. The collision avoidance case study will evaluate the effectiveness of model updates at capturing uncertainty about other actors in an environment.

SCUBA Ascent Environment We further propose a second case study that will evaluate the effectiveness of model updates at capturing uncertainty and changes to the actuated system. This case study will consider the problem of advising a SCUBA diver how to ascend from depth without running out of oxygen and without risking decompression sickness. In previous work with Bajaj and Elmaaroufi, we have developed and verified a model of a SCUBA ascent protocol[5]. This model is an interesting case study for model updates because the system contains several parameters that are difficult or impossible to instantiate at design time. Therefore, parameter insert and instantiation updates are key components of an effective control algorithm based on model updates.

Summary We suggest developing a set of model updates that are capable of expressing model deviations in at least two realistic hybrid systems models. We further propose developing a framework that blends offline verification, online verification, model identification, and reinforcement learning to enforce dynamic safety constraints in dynamic environments while still allowing the system to optimize for other, non-safety-critical constraints. Achieving this goal will require:

- identifying a set of verification-preserving model updates (verification preservation may be assured by offline verification, online (i.e., lightweight runtime) verification, or some combination thereof);
- extending the JSC algorithm to support continuous state and action spaces; and
- incorporating model update identification algorithms into JSC.

Through these contributions, we will show that autonomous cyber-physical systems that use reinforcement learning for control are amenable to formal verification, even outside from an initial model space.

5.2 Related Work on System Identification and Program Repair

Automatically modifying or constructing programs based upon logical specifications, test data, and/or environmental data is a common approach used by researchers in many different disciplines. This section reviews approaches from the control theory, software engineering, AI/ML, and programming languages research communities.

Summary of Comparisons Unlike existing work, the verification-preserving model updates discussed in Section 5.1 learn how to modify a *continuous* system in response to data, discover a *corresponding* update to a *discrete* system that preserves a hybrid reachability property, and then synthesize a formal and computer-checked proof that the resulting combination of discrete/continuous model updates continue to satisfy relevant safety invariants.

[5] https://github.com/nrfulton/scuba-release

Model and System Identification

System identification algorithms use data to build a discrete, continuous, or even hybrid dynamical systems model of the observed process. System identification is an enormous and mature area of research. Diester provides a historical overview of system identification [30]. Garg et al. survey approaches toward system identification for control that decomposes techniques across several different dimensions [46]. Juloski et al. survey and compare approaches toward system identification for certain classes of hybrid systems [61].

Each of the model updates proposed in Section 5.1.1 will associate with each model update one or more model identification algorithms for determining whether the update is appropriate. For example, the parameter instantiation update discussed in Section 5.1.2 will be used in both of the case studies discussed in Section 5.1.3 and will use system identification techniques to determine when the update is appropriate and how it should be applied.

In general, each VPMU can be viewed as a particular subclass of system identification problems for discrete, continuous, or hybrid systems. When viewed from a system identification perspective, this thesis proposes contributing:

1. an approach toward preserving reachability properties while applying system verification techniques (or at least recognizing when the identified system no longer satisfies a desired invariant), and
2. a technique for combining system identification procedures for different subclasses of systems.

Program Synthesis and Repair

Program synthesis algorithms attempt to automatically generate programs in a fixed programming language. The basis for synthesis might be a formal specification, a test suite, or a set of I/O pairs. Automatic program repair algorithms are a special class of synthesis algorithms that attempt to synthesize bug-fixing patches for existing programs. Approaches toward program repair differ on the basis of the programming language under consideration, the domain of relevance, the method for producing patch candidates, and the basis for accepting or rejecting a repair.

GenProg by Le Goues et al. [72] uses genetic programming to generate repairs for C programs using test cases as the primary definition of correctness. Rothenberg and Grumberg [93] leverage SAT and SMT solvers to generate repairs using logical assertions as a basis for correctness. Le et al. [71] leverage a combination of deductive verification and genetic programming to generate repairs.

Our proposed approach leverages data-driven approaches to decide how environmental models should change. Especially for autonomous systems, program repairs based upon designer intent are irrelevant to the choice of an accurate environmental model – *the world wants what it wants*. However, once a repair to the environmental model is identified, we leverage logic-driven approaches to decide how a controller should change in response to identified changes in environmental models. This combination that allows us to adapt to unforeseen environmental behaviors without sacrificing reachability proofs or losing track of designer intent.

When viewed from the perspective of program synthesis and repair, the proposed work

1. contributes the first library of mutations for continuous programs (ODEs) and hybrid programs,
2. demonstrates how to combine data-oriented mutations for environmental models with logically constrained mutations for controller models in a way that preserves reachability properties for the combined hybrid dynamical system, and
3. shows how program repairs provide a setting for characterizing situations in which learning algorithms are capable of operating safely.

Programming Language Representations in Reinforcement Learning

Recent work leverages programming language theory as part of a reinforcement learning algorithm. For example, Verma et al. use a simple functional programming language to ensure intrepretability of learned policies [100]. Our work has a similar goal, but focuses on a language with a rich combination of discrete and continuous dynamics. Like other related work on using LTL specifications to constrain reinforcement learning, this work might compose well with our current and proposed work by providing an attractive setting for planning problems while $\mathsf{d}\mathcal{L}$-based approaches provide an attractive setting for problems where interpretability is fundamentally limited by the lack of differential equations in the policy language (i.e., most controls problems).

Ghosh et al. [48] propose an approach toward constrained optimization that monitors for safety violations using a specification monitor and uses a grammatical approach toward bounding model deviation. Unlike the proposed work, Ghosh et al. do not consider the case where the grammar can express not just arithmetic constraints but entire hybrid dynamical systems and also do not consider the problem of preserving formal proofs.

Other Related Work on Hybrid Systems

We propose a data-driven approach toward safe control, mediated by model identification algorithms specialized to preserve verification results. Many other data-driven approaches toward hybrid systems control are suggested in the literature; for example, Kushner et al. take a data-driven approach toward control for an artificial pancreas [68] and Althoff et al. suggest a demonstration-driven approach [49]. Kumar et al. introduce an approach toward learning based upon Hamiltonian control [67] inspired by earlier work by Nerode and Kohn [65].

Our proposed approach uses (online mutations of) specifications to monitor cyber-physical systems. Our monitors are generated by KeYmaera X using Modelplex; Bartocci et al. review various other approaches toward monitoring CPS [15]. Some of our proposed model updates make use of both offline and online verification; Johnson et al. suggest other approaches toward the online component of this verification effort [60]. Some (but not all) of our proposed model updates are refinements of hybrid programs; we could show these are verification-preserving using Loos' differential refinement logic [74].

5.3 Summary of All Related Work

The proposed thesis leverages several research areas: hybrid systems verification (HSV), reinforcement learning/optimal control (RL/OC) constrained reinforcement learning (CRL), model/system identification (MI/SI), and program synthesis/repair (PS&R). Table 5.1 summarizes all of these research areas as they relate to our previous and proposed work. The Safe columns indicate whether the technique guarantees safety in model space ($\in$ MS) and outside of model space ($\notin$ MS); for techniques that have multiple phases (e.g., reinforcement learning), safety is taken to mean safety throughout system execution. The HS column indicates whether the family of techniques is applicable to hybrid systems. The "Changes Model" column indicates whether the family of techniques will produce a human-readable model explaining why the system behaved as it did (either inside or outside of model space). The "Explainable" column indicates whether the approach offers some other method for ensuring that safety guarantees are explainable to system designers and other stakeholders.

Table 5.1: Summary of All Related Work.

Approach	**Safe $\in$ MS**	**Safe $\notin$ MS**	**HS**	Changes Model	Explainable
JSC+VPMU[6]	Formal	Formal	Yes	Yes	Yes
JSC[40]	Formal	Informal	Yes	No	Yes
Software Verification	Formal	No	No	No	Yes
HS Verification[7]	Formal	No	Yes	No	Yes
PS&R w/ formal specs[8]	Some	Some	No	Yes (not HS)	Yes
PS&R w/ test cases	No	No	No	Yes (not HS)	Yes
MI/SI[9]	Some (not formal)	Some (not formal)	Some	Some	Some
RL/OC[10]	No	No	Some	No	Some
Constrained RL/OC	Informal	No	Some	No	Some
LTL FM for RL[4, 51]	Formal	No	No	No	Yes
MC for NNs (e.g., [64, 102])	Formal	No	Some	No	Yes

[6]Proposed.
[7]A more in-depth comparison of software and hybrid systems verification techniques is presented in Table 3.1.
[8]Section 5.2 compares JSC+VPMU to program synthesis and repair techniques in more detail.
[9]Section 5.2 discusses the relationship between JSC+VPMU and model/system identification in more detail.
[10]Section 4.2 discusses the relationship between JSC and (constrained) learning/optimization in more detail.

5.4 Timeline

My thesis proposal will be scheduled over the summer. The remainder of my thesis work will be scheduled as follows:

- A paper evaluating this approach on several sets of model updates and their associated environments will be submitted to AAAI 2019 on September 1.
- A paper establishing safety and convergence guarantees for JSC with Model Updates in a generic setting will be submitted to TACAS in September.
- The final thesis document will be prepared concurrently to the preparation of these submissions, and a final defense will occur in Fall 2018.

5.5 Conclusion and Future Work

The future of mobility is autonomous, so we must figure out today how to design autonomous systems that are demonstrably safe. Traditional approaches toward hybrid systems verification assume a single, sufficiently accurate model and ignore non-safety constraints. The proposed thesis will move beyond this assumption without abandoning the manifold advantages provided by model-based approaches toward engineering autonomous systems.

System designers who use this methodology will be able to encode the many ways in which the world might be different from nominal expectations. By making explicit the model corresponding to a control policy and coupling changes in the policy with changes to the underlying continuous dynamics, we ensure that systems will at least attempt to adapt instead of obliviously continuing even after detecting demonstrably false assumptions. And instead of blinding changing the control policy in ways that might cause catastrophic failures, we will preserve a powerful extension to the usual verificationist guarantee: *if* the system is able to find an accurate model of the system, then the system will control safely.

In future work, we will continue to develop new verification-preserving hybrid systems model updates and continue to improve the heuristics used to select model updates. This work will be enabled, in part, by continued improvements in hybrid systems theorem provers and reinforcement learning algorithms.

Bibliography

[1] *Conference on Automated Deduction - CADE-25*, volume 9195 of *LNCS*, 2015. Springer. ISBN 978-3-319-21400-9. doi: 10.1007/978-3-319-21401-6. 5.5

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 22–31, 2017. URL http://proceedings.mlr.press/v70/achiam17a.html. 4.2

[3] Wolfgang Ahrendt, Bernhard Beckert, Daniel Bruns, Richard Bubel, Christoph Gladisch, Sarah Grebing, Reiner Hähnle, Martin Hentschel, Mihai Herda, Vladimir Klebanov, Wojciech Mostowski, Christoph Scheben, Peter H. Schmitt, and Mattias Ulbrich. The KeY platform for verification and analysis of Java programs. In Dimitra Giannakopoulou and Daniel Kroening, editors, *Verified Software: Theories, Tools and Experiments - 6th International Conference, VSTTE 2014, Vienna, Austria, July 17-18, 2014, Revised Selected Papers*, volume 8471 of *LNCS*, pages 55–71. Springer, 2014. ISBN 978-3-319-12153-6. doi: 10.1007/978-3-319-12154-3_4. 3.2.2

[4] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211. 4.2, 5.1

[5] Matthias Althoff and Dmitry Grebenyuk. Implementation of interval arithmetic in CORA 2016. In Frehse and Althoff [37], pages 91–105. URL http://www.easychair.org/publications/paper/334330. 2

[6] Eitan Altman. Constrained markov decision processes, 1999. 4.2

[7] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992. ISBN 3-540-57318-6. doi: 10.1007/3-540-57318-6_30. 2.1

[8] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the Real-Time Systems Symposium. Raleigh-Durham, NC, USA, December 1993*, pages 2–11. IEEE Computer Society, 1993. ISBN

0-8186-4480-X. doi: 10.1109/REAL.1993.393520. URL https://doi.org/10.1109/REAL.1993.393520. 2

[9] Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors. *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*, 1996. Springer. ISBN 3-540-61155-X. doi: 10.1007/BFb0020931. URL https://doi.org/10.1007/BFb0020931. 5.5

[10] Rajeev Alur, Radu Grosu, and Bow-Yaw Wang. Automated refinement checking for asynchronous processes. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, volume 1954 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2000. ISBN 3-540-41219-0. doi: 10.1007/3-540-40922-X_5. URL https://doi.org/10.1007/3-540-40922-X_5. 2

[11] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*, pages 254–257. Springer, 2011. ISBN 978-3-642-19834-2. doi: 10.1007/978-3-642-19835-9_21. URL https://doi.org/10.1007/978-3-642-19835-9_21. 2

[12] Sergei N. Artemov. Operational modal logic. Technical Report MSI 9529, Cornell University, 1995. 3.1.1

[13] SergeiN. Artemov and LevD. Beklemishev. Provability Logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 13 of *Handbook of Philosophical Logic*, pages 189–360. Springer Netherlands, 2005. ISBN 978-1-4020-3520-3. doi: 10.1007/1-4020-3521-7_3. 3.1.1

[14] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 365–370. Springer, 2002. ISBN 3-540-43997-8. doi: 10.1007/3-540-45657-0_30. URL https://doi.org/10.1007/3-540-45657-0_30. 2

[15] Ezio Bartocci, Jyotirmoy V. Deshmukh, Alexandre Donzé, Georgios E. Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 135–175. Springer, 2018. ISBN 978-3-319-75631-8. doi: 10.1007/978-3-319-75632-5_5. URL https://doi.org/10.1007/978-3-319-75632-5_5. 5.2

[16] A. Bemporad. Hybrid Toolbox - User's Guide, 2004. http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox. 2

[17] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995. 2, 3.2.2

[18] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In Alur et al. [9], pages 232–243. ISBN 3-540-61155-X. doi: 10.1007/BFb0020949. URL https://doi.org/10.1007/BFb0020949. 2, 3.2.2

[19] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*, volume 7550 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2012. ISBN 978-3-642-33511-2. doi: 10.1007/978-3-642-33512-9_8. URL https://doi.org/10.1007/978-3-642-33512-9_8. 2, 3.2.2

[20] Brandon Bohrer, Vincent Rahli, Ivana Vukotic, Marcus Völp, and André Platzer. Formally verified differential dynamic logic. In *Certified Programs and Proofs - 6th ACM SIGPLAN Conference, CPP 2017*, pages 208–221. ACM, 2017. doi: 10.1145/3018610.3018616. 3.1, 3.2.2

[21] Brandon Bohrer, Yong Kiam Tan, Stefan Mitsch, Magnus O. Myreen, and André Platzer. VeriPhy: Verified controller executables from verified cyber-physical system models. In Dan Grossman, editor, *PLDI*, pages 617–630. ACM, 2018. doi: 10.1145/3192366.3192406. 3.2.2

[22] Christopher Brooks. Ptolemy ii: An open-source platform for experimenting with actor-oriented design, February 2016. URL http://chess.eecs.berkeley.edu/pubs/1166.html. Poster presented at the ¡a href="https://www.terraswarm.org/conferences/16/bears/"¿2016 Berkeley EECS Annual Research Symposium¡/a¿. 2

[23] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2013. ISBN 978-3-642-39798-1. doi: 10.1007/978-3-642-39799-8_18. URL https://doi.org/10.1007/978-3-642-39799-8_18. 2

[24] Xin Chen, Sriram Sankaranarayanan, and Erika Ábrahám. Flow* 1.2: More effective to play with hybrid systems. In Goran Frehse and Matthias Althoff, editors, *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015.*, volume 34 of *EPiC Series in Computing*, pages 152–159.

EasyChair, 2015. URL http://www.easychair.org/publications/paper/248659. 2

[25] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, Aug 2015. ISSN 1433-2787. doi: 10.1007/s10009-014-0361-y. URL https://doi.org/10.1007/s10009-014-0361-y. 2, 3.2.2

[26] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In Alur et al. [9], pages 208–219. ISBN 3-540-61155-X. doi: 10.1007/BFb0020947. URL https://doi.org/10.1007/BFb0020947. 2, 3.2.2

[27] N. G. de Bruijn. *AUTOMATH, a Language for Mathematics*, pages 159–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_11. URL https://doi.org/10.1007/978-3-642-81955-1_11. 3.1.1

[28] Leonardo Mendonça de Moura, Jeremy Avigad, Soonho Kong, and Cody Roux. Elaboration in dependent type theory. *CoRR*, abs/1505.04324, 2015. URL http://arxiv.org/abs/1505.04324. 3

[29] Leonardo Mendonça de Moura et. al. The Lean theorem prover (system description). In *CADE-25* DBL [1], pages 378–388. ISBN 978-3-319-21400-9. doi: 10.1007/978-3-319-21401-6_26. 3

[30] Manfred Deistler. System identification and time series analysis: Past, present, and future. In Bozenna Pasik-Duncan, editor, *Stochastic Theory and Control*, pages 97–109, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-48022-8. 5.2

[31] Kurt Driessens and Saso Dzeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004. doi: 10.1023/B:MACH.0000039779.47329.3a. 4.2

[32] Parasara Sridhar Duggirala, Matthew Potok, Sayan Mitra, and Mahesh Viswanathan. C2E2: a tool for verifying annotated hybrid systems. In Antoine Girard and Sriram Sankaranarayanan, editors, *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 307–308. ACM, 2015. ISBN 978-1-4503-3433-4. doi: 10.1145/2728606.2728646. URL http://doi.acm.org/10.1145/2728606.2728646. 2

[33] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *CoRR*, abs/1805.10265, 2018. URL http://arxiv.org/abs/1805.10265. 4.2

[34] Johan Eker, Jorn W Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, and Stephen Neuendorffer. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003. URL http://chess.eecs.berkeley.edu/pubs/488.html. 2

[35] Simon Foster and Jim Woodcock. Towards verification of cyber-physical systems

with UTP and isabelle/hol. In Thomas Gibson-Robinson, Philippa J. Hopcroft, and Ranko Lazic, editors, *Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*, volume 10160 of *Lecture Notes in Computer Science*, pages 39–64. Springer, 2017. ISBN 978-3-319-51045-3. doi: 10.1007/978-3-319-51046-0_3. URL https://doi.org/10.1007/978-3-319-51046-0_3. 3.2.2

[36] Goran Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008. doi: 10.1007/s10009-007-0062-x. 2, 3.2.2

[37] Goran Frehse and Matthias Althoff, editors. *ARCH@CPSWeek 2016, 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems, Vienna, Austria*, volume 43 of *EPiC Series in Computing*, 2017. EasyChair. URL http://www.easychair.org/publications/volume/ARCH16. 5.5

[38] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *23rd CAV, 2011*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011. ISBN 978-3-642-22109-5. doi: 10.1007/978-3-642-22110-1_30. 2, 3.2.2

[39] Nathan Fulton and André Platzer. A logic of proofs for differential dynamic logic: toward independently checkable proof certificates for dynamic logics. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, 2016*, pages 110–121, 2016. doi: 10.1145/2854065.2854078. URL http://doi.acm.org/10.1145/2854065.2854078. 3.1.1, 3.1.2

[40] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In Sheila McIlraith and Kilian Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Louisiana, USA.*, pages 6485–6492. AAAI Press, 2018. 1.1, 4, 4.3, 5.1

[41] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy P. Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015. doi: 10.1007/978-3-319-21401-6_36. 2.1.3, 3.1

[42] Nathan Fulton, Stefan Mitsch, Brandon Bohrer, and Andrè Platzer. Bellerophon: Tactical theorem proving for hybrid systems. In *Interactive Theorem Proving 2017*, 2017. 3.2, 3.2.2

[43] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. Delta-decidability over the reals. *CoRR*, abs/1204.6671, 2012. URL http://arxiv.org/abs/1204.6671. 3.2.2

[44] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013. ISBN 978-3-642-38573-5. doi: 10.1007/978-3-642-38574-2_14. 2, 3.2.2

[45] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015. 4.2

[46] A. Garg, K. Tai, and B. N. Panda. System identification: Survey on modeling methods and models. In Subhransu Sekhar Dash, K. Vijayakumar, Bijaya Ketan Panigrahi, and Swagatam Das, editors, *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, pages 607–615, Singapore, 2017. Springer Singapore. ISBN 978-981-10-3174-8. 5.2

[47] Peter Geibel. Reinforcement learning for MDPs with constraints. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pages 646–653, 2006. doi: 10.1007/11871842_63. 4.2

[48] Shromona Ghosh, Felix Berkenkamp, Gireeja Ranade, Shaz Qadeer, and Ashish Kapoor. Verifying Controllers Against Adversarial Examples with Bayesian Optimization. *CoRR*, abs/1802.08678, 2018. URL http://arxiv.org/abs/1802.08678. 5.2

[49] Andrea Giusti, Martijn J. A. Zeestraten, Esra Icer, Aaron Pereira, Darwin G. Caldwell, Sylvain Calinon, and Matthias Althoff. Flexible automation driven by demonstration: Leveraging strategies that simplify robotics. *IEEE Robot. Automat. Mag.*, 25(2):18–27, 2018. doi: 10.1109/MRA.2018.2810543. URL https://doi.org/10.1109/MRA.2018.2810543. 5.2

[50] John Harrison, Konrad Slind, and Rob Arthan. HOL. In Freek Wiedijk, editor, *The Seventeen Provers of the World, Foreword by Dana S. Scott*, volume 3600 of *Lecture Notes in Computer Science*, pages 11–19. Springer, 2006. ISBN 3-540-30704-4. doi: 10.1007/11542384_3. URL https://doi.org/10.1007/11542384_3. 3

[51] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-correct reinforcement learning. *CoRR*, abs/1801.08099, 2018. URL http://arxiv.org/abs/1801.08099. 4.2, 5.1

[52] Matthias Heger. Consideration of risk in reinformance learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 105–111. Morgan Kaufmann, 1994. ISBN 1-55860-335-2. 4.2

[53] David Held, Zoe McCarthy, Michael Zhang, Fred Shentu, and Pieter Abbeel. Probabilistically safe policy transfer. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 5798–5805. IEEE, 2017. ISBN 978-1-5090-4633-1. doi: 10.1109/ICRA.2017.7989680. URL https://doi.org/10.1109/ICRA.2017.7989680. 4.2

[54] Thomas A. Henzinger. 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*, pages 278–292. IEEE Computer Society, 1996. ISBN 0-8186-7463-6. doi: 10.1109/LICS.1996.561342. 3.2.2

[55] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *STTT*, 1(1-2):110–122, 1997. doi: 10.1007/s100090050008. URL https://doi.org/10.1007/s100090050008. 2

[56] Jannik Hüls, Stefan Schupp, Anne Remke, and Erika Abraham. Analyzing hybrid petri nets with multiple stochastic firings using hypro. In *Proc. of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'17)*, 2017. 2

[57] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in isabelle/hol. In Lennart Beringer and Amy P. Felty, editors, *Interactive Theorem Proving - Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings*, volume 7406 of *Lecture Notes in Computer Science*, pages 377–392. Springer, 2012. ISBN 978-3-642-32346-1. doi: 10.1007/978-3-642-32347-8_26. URL https://doi.org/10.1007/978-3-642-32347-8_26. 3.2.2

[58] Fabian Immler and Johannes Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, 2012, 2012. URL https://www.isa-afp.org/entries/Ordinary_Differential_Equations.shtml. 3.2.2

[59] ISO. ISO 26262 road vehicles functional safety. 2011. 2.2

[60] Taylor T. Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. *ACM Trans. Embedded Comput. Syst.*, 15(2):26:1–26:27, 2016. doi: 10.1145/2723871. URL http://doi.acm.org/10.1145/2723871. 5.2

[61] Aleksandar Lj. Juloski, W. P. M. H. Heemels, Giancarlo Ferrari-Trecate, René Vidal, Simone Paoletti, and J. H. G. Niessen. Comparison of four procedures for the identification of hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 354–369, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31954-2. 5.2

[62] Yoshinobu Kadota, Masami Kurano, and Masami Yasuda. Discounted Markov decision processes with utility constraints. *Computers and Mathematics with Applications*, 51(2): 279 – 284, 2006. ISSN 0898-1221. doi: http://dx.doi.org/10.1016/j.camwa.2005.11.013. 4.2

[63] Nidhi Kalra and Susan M. Paddock. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, 2016. URL http://www.jstor.org/stable/10.7249/j.ctt1btc0xw. 2.2

[64] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017. doi: 10.1007/978-3-319-63387-9_5. 4.2, 5.1

[65] Wolf Kohn, Anil Nerode, and Jeffrey B. Remmel. Hybrid systems as finsler manifolds: Finite state control as approximation to connections. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994*, volume 999 of *Lecture Notes in Computer Science*, pages 294–321. Springer, 1994. ISBN 3-540-60472-3. doi: 10.1007/3-540-60472-3_15. URL https://doi.org/10.1007/3-540-60472-3_15. 5.2

[66] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015*, volume 9035 of *LNCS*, pages 200–205. Springer, 2015. ISBN 978-3-662-46680-3. doi: 10.1007/978-3-662-46681-0_15. 2, 3.2.2

[67] Peeyush Kumar, Wolf Kohn, and Zelda B. Zabinsky. Near optimal hamiltonian-control and learning via chattering. *CoRR*, abs/1703.06485, 2017. URL http://arxiv.org/abs/1703.06485. 5.2

[68] Taisa Kushner, David Bortz, David M. Maahs, and Sriram Sankaranarayanan. A data-driven approach to artificial pancreas verification and synthesis. In Chris Gill, Bruno Sinopoli, Xue Liu, and Paulo Tabuada, editors, *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018, Porto, Portugal, April 11-13, 2018*, pages 242–252. IEEE / ACM, 2018. URL http://dl.acm.org/citation.cfm?id=3207925. 5.2

[69] Yuri A. Kuznetsov. *Elements of Applied Bifurcation Theory (2Nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1998. ISBN 0-387-98382-1. 3.2.1

[70] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and developments. Number 1254 in LNCS, pages 456–459. Springer–Verlag, June 1997. 2, 3.2.2

[71] Xuan-Bach D. Le, Quang Loc Le, David Lo, and Claire Le Goues. Enhancing automated program repair with deductive verification. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*, pages 428–432. IEEE Computer Society, 2016. ISBN 978-1-5090-3806-0. doi: 10.1109/ICSME.2016.66. URL https://doi.org/10.1109/ICSME.2016.66. 5.2

[72] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *IEEE Trans. Software Eng.*, 38(1):54–72, 2012. doi: 10.1109/TSE.2011.104. URL https://doi.org/10.1109/TSE.2011.104. 5.2

[73] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. A calculus for hybrid CSP. In Kazunori Ueda, editor, *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, volume 6461 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010. ISBN 978-3-642-17163-5. doi: 10.1007/978-3-642-17164-2_1. URL https://doi.org/10.1007/978-3-642-17164-2_1. 3.2.2

[74] Sarah M. Loos and André Platzer. Differential refinement logic. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 505–514. ACM, 2016. ISBN 978-1-4503-4391-6. doi: 10.1145/2933575.2934555. URL http://doi.acm.org/10.1145/2933575.2934555. 5.2

[75] Ibtissem Ben Makhlouf, Norman Hansen, and Stefan Kowalewski. Hyreach: A reachability tool for linear hybrid systems based on support functions. In Frehse and Al-

thoff [37], pages 68–79. URL http://www.easychair.org/publications/paper/334327. 2

[76] Per Martin-Löf. An Intuitionistic Theory of Types. 1998. 3.1.1

[77] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. URL http://coq.inria.fr. Version 8.0. 3

[78] Stefan Mitsch and André Platzer. The keymaera X proof IDE - concepts on usability in hybrid systems theorem proving. In Catherine Dubois, Paolo Masci, and Dominique Méry, editors, *Proceedings of the Third Workshop on Formal Integrated Development Environment, F-IDE@FM 2016, Limassol, Cyprus, November 8, 2016.*, volume 240 of *EPTCS*, pages 67–81, 2016. doi: 10.4204/EPTCS.240.5. URL https://doi.org/10.4204/EPTCS.240.5. 3.2.2

[79] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.*, 49(1):33–74, 2016. ISSN 0925-9856. doi: 10.1007/s10703-016-0241-z. Special issue of selected papers from RV'14. 3.2.1

[80] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in Markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012. 4.2

[81] Markus Nick, Sören Schneickert, Jürgen Grotepaß, and Ingo Heine. Checkmate. *KI*, 21(4):34–37, 2007. URL http://www.kuenstliche-intelligenz.de/fileadmin/template/main/archiv/pdf/ki2007-04_page34_web_teaser.pdf. 2

[82] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005. doi: 10.1287/opre.1050.0216. URL https://doi.org/10.1287/opre.1050.0216. 4.2

[83] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. ISBN 3-540-43376-7. doi: 10.1007/3-540-45949-9. URL http://dx.doi.org/10.1007/3-540-45949-9. 3

[84] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry S. Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intelligent Vehicles*, 1(1):33–55, 2016. doi: 10.1109/TIV.2016.2578706. URL https://doi.org/10.1109/TIV.2016.2578706. 5.1.3

[85] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Towards practical verification of machine learning: The case of computer vision systems. *CoRR*, abs/1712.01785, 2017. URL http://arxiv.org/abs/1712.01785. 4.2

[86] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. ISSN 0168-7433. doi: 10.1007/s10817-008-9103-8. 2.1.1, 2.1.2

[87] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. ISBN 978-3-642-14508-7. doi: 10.1007/

978-3-642-14509-4. 2.1.1, 2.1.2

[88] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012. ISBN 978-1-4673-2263-8. doi: 10.1109/LICS.2012.13. 2.1, 2.1.1, 2.1.2

[89] André Platzer. A uniform substitution calculus for differential dynamic logic. In *CADE* DBL [1]. ISBN 978-3-319-21400-9. 2.1.3, 2.1.3

[90] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–266, 2017. doi: 10.1007/s10817-016-9385-1. 2.1.2

[91] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. ISBN 978-3-540-71069-1. doi: 10.1007/978-3-540-71070-7_15. 3.2.2

[92] André Platzer and Yong Kiam Tan. Differential equation axiomatization: The impressive power of differential ghosts. In Anuj Dawar and Erich Grädel, editors, *LICS*, pages 819–828, New York, 2018. ACM. ISBN 978-1-4503-5583-4. doi: 10.1145/3209108.3209147. 3.2.2

[93] Bat-Chen Rothenberg and Orna Grumberg. Sound and complete mutation-based program repair. In John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*, volume 9995 of *Lecture Notes in Computer Science*, pages 593–611, 2016. ISBN 978-3-319-48988-9. doi: 10.1007/978-3-319-48989-6_36. URL https://doi.org/10.1007/978-3-319-48989-6_36. 5.2

[94] RTCA. DO-178C software considerations in airborne systems and equipment certification. 2012. 2.2

[95] Stuart J. Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *CoRR*, abs/1602.03506, 2016. URL http://arxiv.org/abs/1602.03506. 4.2

[96] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. Hypro: A c++ library for state set representations for hybrid systems reachability analysis. In *Proc. of the 9th NASA Formal Methods Symposium (NFM'17)*, volume 10227 of *LNCS*, pages 288–294. Springer International Publishing, 2017. 2

[97] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008. ISBN 978-3-540-71065-3. doi: 10.1007/978-3-540-71067-7_6. URL https://doi.org/10.1007/978-3-540-71067-7_6. 3

[98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. A Bradford Book. 2.2

[99] Aviv Tamar, Huan Xu, and Shie Mannor. Scaling up robust MDPs by reinforcement learn-

ing. *CoRR*, abs/1306.6189, 2013. URL http://arxiv.org/abs/1306.6189. 4.2

[100] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5052–5061. JMLR.org, 2018. URL http://proceedings.mlr.press/v80/verma18a.html. 5.2

[101] Qinsi Wang, Paolo Zuliani, Soonho Kong, Sicun Gao, and Edmund M. Clarke. Sreach: Combining statistical tests and bounded model checking for nonlinear hybrid systems with parametric uncertainty. *CoRR*, abs/1404.7206, 2014. URL http://arxiv.org/abs/1404.7206. 2

[102] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. *CoRR*, abs/1804.10829, 2018. URL http://arxiv.org/abs/1804.10829. 4.2, 5.1