The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning

Nathan Fulton, André Platzer
Computer Science Department,
Carnegie Mellon University, Pittsburgh, USA
{nathanfu, aplatzer}@cs.cmu.edu

Abstract

Formal verification provides a high degree of confidence in safe system operation, but only if reality matches the verified model. Although a good model will be accurate most of the time, even the best models are incomplete. This is especially true in Cyber-Physical Systems because high-fidelity physical models of systems are expensive to develop and often intractable to verify. Conversely, reinforcement learning-based controllers are lauded for their flexibility in unmodeled environments, but do not provide guarantees of safe operation.

This paper presents an approach for provably safe learning that provides the best of both worlds: the exploration and optimization capabilities of learning along with the safety guarantees of formal verification. Our main insight is that formal verification combined with verified runtime monitoring can ensure the safety of a learning agent. Verification results are preserved whenever learning agents limit exploration within the confounds of verified control choices as long as observed reality comports with the model used for off-line verification. When a model violation is detected, the agent abandons efficiency and instead attempts to learn a control strategy that guides the agent to a modeled portion of the state space.

We prove that our approach toward incorporating knowledge about safe control into learning systems preserves safety guarantees, and demonstrate that we retain the empirical performance benefits provided by reinforcement learning. We also explore various points in the design space for these justified speculative controllers in a simple model of adaptive cruise control model for autonomous cars.

Introduction

Cyber-physical systems (CPSs) are difficult to get right, which is why formal verification provides rigorous ways of establishing the safety of controllers with respect to a physical model of the system under control. However, difficulties with formally verified controllers arise whenever there are discrepancies between the verified models and the real implementation. Such discrepancies between model and reality are inevitable in physical systems operating in open environments (Mitsch and Platzer 2016).

Reinforcement learning (RL) (Sutton and Barto 1998) provides ways of learning controllers that tend to perform well without the need for a perfect model – or even any model at all. 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 become 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 (ISO 2011; RTCA 2012). 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 (Kalra and Paddock 2016).

This paper contributes a technique, called *Justified Speculative Control* (JSC), for transferring formal verification results to controllers obtained via reinforcement learning. This approach combines the safety assurance provided by verification with the optimality and robustness to model incompleteness provided by reinforcement learning. Our primary contribution is a set of proofs that transfer computer-checked safety proofs for hybrid dynamical systems to the policy obtained by a generic reinforcement learning algorithm. We also present a case study and experiment that demonstrates leveraging verification results speeds up learning.

Justified Speculative Control combines verified runtime monitoring – backed by formally verified models – with reinforcement learning. When no modeling inaccuracy is detected, the system's actions are constrained to a set of verified control actions. When modeling inaccuracy is detected, the system is justified in taking actions that are unverified and possibly unsafe with respect to the (inaccurate) model.

Reinforcement learning is useful even when the model is accurate because models used for verification are often nondeterministic. The verified controller is not a single deterministic policy, but rather a nondeterministic policy with multiple safe actions. Verification results establish the safety of a set of actions, but do not solve the optimization problem without significant extra effort. A verification result usually will not tell the system how to best choose the most efficient control action from among as set of safe control actions.

For accurate models, Justified Speculative Control explains how to sandbox the learning process by a formally verified nondeterministic model. The JSC sandbox monitors the environment and checks that observed state transitions comport with the system of differential equations used to

model the environment during verification. The sandbox allows the learning agent to *safely* optimize over the set of safe control actions. Unlike other sandboxing techniques such as Simplex, Justified Speculative Control transfers not just the existing controller but also the verification result about that controller, which is crucial in safety-critical settings.

For inaccurate models, we present experimental evidence that sandboxing constraints provide a useful signal for safe reinforcement learning even when verified models are not available. We translate boolean-valued sandboxing constraints into a real-valued metric and then use this metric as a reward signal, effectively prioritizing policies that drive the system back into well-modeled portions of the state space.

Our approach is compositional with other approaches to safe learning. We leverage the formal verification capabilities of the theorem prover KeYmaera X (Fulton et al. 2015) for differential dynamic logic (Platzer 2008; 2012b; 2012a; 2017) to verify the CPS controller for its model. We exploit the capabilities of the ModelPlex approach (Mitsch and Platzer 2016) to generate monitor conditions that provably correctly check the compliance of observed controller and model actions with the models. Finally, we rely on reinforcement learning algorithms with guarantees of convergence toward an optimal control policy.

Summarily, this paper contributes:

- The first approach toward provably safe Reinforcement Learning: Justified Speculative Control (JSC). JSC combines off-line formal verification, runtime monitoring, and reinforcement learning to transfer *proofs* of safety to learned policies.
- A proof that JSC controllers preserve the safety properties of verified controllers, both during learning and once a policy is extracted. The ability to transfer proofs of safety to reinforcement learning algorithms lowers the barriers to using these algorithms in safety-critical settings where best engineering practice demands verification.
- An empirical evaluation of a Reinforcement Learning algorithm that incorporates real-valued sandboxing constraints into its objective function. Our experiments are preliminary but promising. We observe two trends that deserve further investigation. Within modeled portions of the state space, we experimentally observe that JSC finds safe policies more quickly than the original learning algorithm. Within unmodeled portions of the state space, our experiments demonstrate that metricizing the sandboxing constraints decreases the number of unsafe states.

This paper begins an exploration of the interaction between formal verification and safe reinforcement learning by introducing a general scheme for transferring formal verification results to reinforcement learning algorithms. This general scheme lays the foundation for exploring many new questions in verified safe learning. It is amenable to extension to other learning algorithms, approaches toward safe learning after model deviation, and approaches toward formal verification.

Background

We recall Differential Dynamic Logic, a logic for verifying properties about safety-critical control software. We then recall the approach, ModelPlex, that allows us to determine when a model built in differential dynamic logic is violated. Finally, we present a generic but formal definition of reinforcement learning.

Differential Dynamic Logic

This section reviews hybrid programs, a programming language for hybrid systems and differential dynamic logic ($\mathsf{d}\mathcal{L}$) for specifying and verifying their properties.

Hybrid (dynamical) systems (Alur et al. 1992; Platzer 2012b) are mathematical models for analyzing the interaction between discrete and continuous dynamics. Hybrid programs (Platzer 2008; 2012b; 2012a; 2017) are a programming language for hybrid systems. Their syntax and informal semantics is given in Table 1.

The following hybrid program is a simple one-dimensional model of a car with straight-line dynamics and two control choices.

Example 1 (Model of a car following a straight line)**.** $\big(\underbrace{(a := A \cup a := 0)}_{ctrl}\,;\ \underbrace{\{p' = v,\ v' = a\}}_{plant}\big)^*$

Example 1 describes a car that chooses nondeterministically to accelerate with a maximum acceleration A or to not accelerate, and then follows a differential equation. This process may repeat arbitrarily many times (indicated by the repetition operator *). Because there is no evolution domain constraint on $plant$, each continuous evolution has any arbitrary non-negative duration $r \in \mathbb{R}$.

Hybrid Program Semantics. Hybrid Programs have a denotational semantics (Platzer 2008; 2012a; 2012b; 2017) formalizing the intuitions in Table 1. The semantics are given in terms of states, which assign a real number to each variable. Each program's semantics $[\![\alpha]\!]$ is a set of pairs (s_1, s_2) where s_2 is a state that is reachable by executing the program from state s_1. For example,

$$[\![x := t]\!] = \{(s_1, s_2) \mid s_1 = s_2 \text{ except } s_2(x) = s_1[\![t]\!]\}.$$

The semantics of programs are set-valued, allowing expression of nondeterminism. For example, $[\![\alpha \cup \beta]\!] = [\![\alpha]\!] \cup [\![\beta]\!]$.

Differential Dynamic Logic. Differential dynamic logic ($\mathsf{d}\mathcal{L}$) (Platzer 2008; 2012b; 2017) is a first-order multimodal logic for specifying and proving properties of hybrid programs. Each hybrid program α is associated with modal operators $[\alpha]$ and $\langle\alpha\rangle$, which express state reachability properties of the parametrizing program. The formula $[\alpha]\phi$ states that the formula ϕ is true in any state reachable by the hybrid program α. Similarly, $\langle\alpha\rangle\phi$ expresses that the formula ϕ is true after some execution of α. The $\mathsf{d}\mathcal{L}$ formulas are generated by the grammar

$$\phi \ ::= \ \theta_1 \backsim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

where θ_i are arithmetic expressions over the reals, ϕ and ψ are formulas, α ranges over hybrid programs, and $\backsim$

Table 1: Hybrid Programs

Program Statement	Meaning
$\alpha;\beta$	Sequentially composes β after α.
$\alpha \cup \beta$	Executes either α or β nondeterministically.
α^*	Repeats α zero or more times nondeterministically.
$x := \theta$	Evaluates term θ and assigns 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 formula F is not true.

is a comparison operator $=, \neq, \geq, >, \leq, <$. The quantifiers quantify over the reals. We denote by $s \models P$ the fact that P is true in state s; e.g., we denote by $s \models [\alpha]P$ the fact that $(s,t) \in [\![\alpha]\!]$ implies $t \models P$ for all states t. When P is true in every state (i.e., valid) we simply write $\models P$.

Example 2 (Safety specification for straight-line car model)**.**

$$\underbrace{v{\geq}0 \wedge A{>}0}_{\textit{initial condition}} \rightarrow [\big(\underbrace{(a{:=}A \cup a{:=}0)}_{\textit{ctrl}} ; \underbrace{\{p'{=}v,\, v'{=}a\}}_{\textit{plant}}\big)^*]\underbrace{v{\geq}0}_{\textit{post cond.}}$$

This formula states that if the car begins with a non-negative velocity, then it will also have a non-negative velocity after choosing new acceleration or coasting and moving for a nondeterministic period of time.

ModelPlex: Verified Runtime Validation

Central to our approach is the ability to check, at runtime, whether or not the current state of the system can be explained by a $\mathsf{d}\mathcal{L}$ formula. The KeYmaera X theorem prover provides a mechanism for translating a $\mathsf{d}\mathcal{L}$ formula of the form $P \rightarrow [\alpha^*]Q$ into a formula of real arithmetic, which checks whether the present behavior of a system fits to this model. The resulting arithmetic is checked at runtime and is accompanied by a correctness proof. This algorithm, called ModelPlex (Mitsch and Platzer 2016), can be used to extract provably correct monitors that check compliance with the model as well as with the controller. If non-equivalence transformations have been used in the ModelPlex monitor synthesis proofs, the resulting monitor may be conservative, i.e. give false alarms. But if the monitor formula evaluates to true at runtime, the execution is guaranteed to be safe.

Controller Monitors ModelPlex controller monitors are boolean functions that monitor whether or not the controller portion of a hybrid systems model has been violated. The monitor takes two inputs – a "pre" state and a "post" state. The controller monitor returns true if and only if there is an execution of the $ctrl$ fragment of the program that, when executed on the "pre" state, produces the "post" state. For example, the controller monitor for Model 2 is:

$$(v_{post} = v \wedge p_{post} = p \wedge a_{post} = A) \vee$$

$$(v_{post} = v \wedge p_{post} = p \wedge a_{post} = 0)$$

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

where a_{post} is the value of a chosen by the controller. Similarly, v_{post} and p_{post} are the values v and a chosen by the controller. Therefore, this controller monitor states that the controller may choose $a := A$ or $a := 0$, but may not change the values of p or v.

We write the controller monitor as a function

$$\mathit{CM} : S \times A \rightarrow \mathit{Bool}$$

mapping a current state $s \in S$ and an action $\mathit{act} \in A$ to a boolean value. This formulation is equivalent to the pre/post state formulation (e.g., $v_{post} = \mathit{act}(v_{pre})$) where

$$\mathit{act}(s)$$

is the state reached by performing the action *act* in state s.

Model Monitors ModelPlex can also produce full model monitors, which check that the *entire* system model is accurate – including the model of the system's physics – for a single control loop. The full model monitor returns true only if the controller for the system chooses a control action that is allowed by the model of the system and also the observed physics of the system correspond to the differential equations describing the system's physical dynamics. The full model monitor for Model 2 is:

$$(t_{post} \geq 0 \ \wedge \ a_{post} = A \ \wedge \ v_{post} = At_{post} + v_{pre} \wedge$$

$$p_{post} = \frac{At_{post}^2}{2} + v_{pre}t_{post} + p_{pre}) \vee$$

$$(t_{post}{\geq}0 \wedge v_{post}{=}v_{pre} \wedge p_{post}{=}v_{post}t_{post} + p_{pre} \wedge a_{post}{=}0)$$

Each side of the disjunction corresponds to a control decision, and the constraints on v and p come from solving the differential equation $p' = v, v' = a$.

We write the ModelPlex monitor as a function

$$\mathit{MM} : S \times A \times S \rightarrow \mathit{Bool}$$

where S is a set of states and A is a set of actions allowed by the controller; the first argument is the state before the control action, the second argument specifies the control action, and the third argument specifies the state after following the plant with the chosen control action.

The ability to perform verified runtime monitoring is essential to our approach – these arithmetic expressions are the conditions that allow us to determine when to use a speculative controller, and when to avoid deviating from the various options available in the verified nondeterministic control policy. We define model monitors semantically.

Reinforcement Learning Models

Reinforcement Learning problems are definable in terms of a Reinforcement Learning Model.

Definition 1 (Reinforcement Learning Model)**.** *Fix a finite set of state variables V. A* state *S is a mapping from each variable in V to a value in $\mathbb{R}$. A* Reinforcement Learning Model *consists of: a set of states S; a set of actions A; a transition mapping $E : S \times A \to S$ mapping a state/action pair to a new state; i.e., $E(s_1, a) = s_2$ if the environment transitions the state $a(s_1)$ to a state s_2; and a reward function R mapping each state and action to a real-valued reward signal.*

We assume that the agent can directly observe the entire state. Crucially, we make very few assumptions about the transition mapping – basically only fixing its I/O type and assuming that the learner can act to follow a transition.

A reinforcement learning task consists of computing a policy π that (at least approximately) optimizes the reward. There are many approaches toward finding an optimal policy π of a reinforcement learning model. The safety results in this paper consider a completely generic approach.

Policies are computed via a learning process, which consists of a strategy for choosing actions based upon the current state ($choose(a, s)$) and determining if the current state is a terminal state ($done(s)$).

Connecting Verification to Learning via Monitors

Given a verified model $init \to [\{ctrl; plant\}^*]safe$, if controller monitor $CM(s, act) =$ True then

$$(s, act(s)) \in [\![ctrl]\!]$$

and if model monitor $MM(s_{pre}, act, s_{post}) =$ True then

$$(act(s_{pre}), s_{post}) \in [\![plant]\!].$$

In this paper, we use model monitors to transfer a safety proof of the above form into guarantees about a reinforcement learning process acting in domains where the environment never results in a model monitor returning false.

Q-learning One approach toward computing an optimal policy π is via Q-learning. At its simplest, Q-learning discovers an optimal policy by computing the discounted future reward of taking a particular action in a particular state; i.e., the reward associated with each state/action pair is a sum of the observed reward and the expected future reward under optimal action, multiplied by a discount factor. A Q-table records this reward for each state/action pair, and is updated during the learning phase. Q-learning is attractive because it does not require an accurate model of the environment. Our transition function E may, for example, be implemented by reading off sampled data from a simulator or real system.

Provably Safe Learning

Justified Speculative Control extends model-based safety theorems to policies obtained through reinforcement learning. All reinforcement learning algorithms, from Q-learning to A3C (Mnih et al. 2016), share common features. Justified Speculative Control respects a generalized safety theorem that applies to any generic reinforcement learning algorithm.

Generic Justified Speculative Control Algorithm

The listing below presents a generic reinforcement learning algorithm with justified speculation. This algorithm corresponds to the approach described in the introduction – the system chooses among a set of verified safe actions whenever the environment is accurately modeled, and otherwise selects any action in the action space.

The inputs are a reinforcement learning model (S, A, R, E), a strategy $choose$ for selecting actions, a function $update$ that records state transitions, and a predicate $done$ over states indicating which states are terminal. Each of these functions has access to the learning model (S, A, R, E) and, typically, some additional state (e.g., a Q table, policy/value function approximator, etc.).

The Justified Speculative Control algorithm leverages the ModelPlex runtime monitors to ensure that, whenever the system is accurately modeled, only safe actions are taken. The model monitor is used to determine if the previously observed state, previous control action, and current state are accurately described within the system model. Whenever the model monitor is true, the controller monitor is then used to prune the action space to only known-safe actions.

JSC also takes as input both $CM : A \times S \to Bool$ as a controller monitor and $MM : S \times A \times S \to Bool$ as model monitor where S is the set of states and A the set of actions.

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 }
```

Lines 2 – 3 begin the process in an initial state; the model monitor will always return true for inputs $s_1, \texttt{NOP}, s_2$ where $s_1 = s_2$. Lines 4 – 12 choose the next action and execute the chosen action until reaching a terminal state. If the model describes the transition from the previous state to the current state via the chosen control action, then a safe action that comports with the controller action is chosen (Lines 5 – 6). Otherwise, the system is allowed to speculate because the model that lead to MM does not accurately characterize the system E (lines 7 – 8). This code assumes that the model's control policy exhibits a *liveness* property; i.e., the set

$$\{a \in A \mid CM(a, curr)\}$$

is always non-empty; we allow non-liveness in the theoretical treatment following this section.[2]. Finally, on lines 9 –

[2]The theoretical treatment following this section handles model monitors for non-live models, but such models are often broken.

11, the state is updated according to the environment and the learning model (e.g., a Q-table or NN) is updated.

Safety Results

The JSC algorithm explores safely whenever the environment E is accurately modeled by the $\mathsf{d}\mathcal{L}$ theorem from which *CM* and MM are defined. This section presents a precise statement of this assertion, effectively demonstrating how to transfer hybrid systems formal verification results to reinforcement learning. Proofs of these theorems are provided in tech report (Fulton and Platzer 2017).

We begin with the definition of a learning process, which is essentially a dynamical systems encoding of the pseudo-code for the JSCGeneric algorithm. Re-stating this algorithm as a dynamical system allows us to give a precise argument without defining a formal semantics for pseudo-code.

Definition 2 (Learning Process)**.** *A tuple of sequences (s_i, u_i, L_i) is a **learning process** for*

$$(init, (S, A, R, E), choose, update, done, \mathit{CM}, MM)$$

if it satisfies the recurrence relations

$$u_i = choose_{L_i}(\{u_i \in A \mid specOK(u, s, i)\}) \quad (1a)$$

$$s_{i+1} = E(u_i, s_i) \quad (1b)$$

$$L_{i+1} = update(L_i, s_i, u_i) \quad (1c)$$

where $s_0 \models init$, L_i is a sequence of learned models, and

$$specOK(u, s, i) \equiv \mathit{CM}(u_i, s_i) \vee \neg MM(s_{i-1}, u_{i-1}, s_i)$$

The three sequences L_i, u_i, s_i all terminate whenever there is no choice for u_i (i.e., an empty set is passed into the $choose$ function), or else when $done(s_i)$. Indices i are non-negative and the predicate $MM(s_{i-1}, u_{i-1}, s_i)$ evaluates to true whenever $i < 1$.

The sequences u, s, L are the selected control action, state, and learned model (e.g., Q table or NN) at each step of the JSCGeneric algorithm. The recurrence relations are equivalent to the computations performed in the JSCGeneric pseudo-code, except the liveness caveat discussed in the previous section.

Corollary 1 (Meaning of Controller Monitor)**.** *Suppose CM is a controller monitor for $P \rightarrow [\{ctrl; plant\}^*]Q$ and $s \in S$ and $u : S \rightarrow S$. Then CM(u, s) implies $(s, u(s)) \in [\![ctrl]\!]$.*

Corollary 2 (Meaning of Model Monitor)**.** *Suppose MM is a model monitor for $init \rightarrow [\{ctrl; plant\}^*]Q$, and that (u, s, L) is a learning process. If MM(s_{i-1}, u_{i-1}, s_i) for all i then $s_i \models Q$, and also $(s_i, u_i(s_i)) \in [\![ctrl]\!]$ implies $(u_i(s_i), s_{i+1}) \in [\![plant]\!]$.*

We are now ready to state the first major safety property – that JSCGeneric does not violate the system's safety properties *during* reinforcement learning if the environment is accurately modeled.

Definition 3. *An environment E is **accurately modeled** by a system $\{ctrl; plant\}^*$ for a set of actions A and states S if for all $s \in S$ and $u \in A$,*

$$(s, u(s)) \in [\![ctrl]\!] \text{ implies } (u(s), E(s, u)) \in [\![plant]\!] \quad (2)$$

Theorem 1 (JSCGeneric Explores Safely in Modeled Environments)**.** *Assume a valid safety specification*

$$\models init \rightarrow [\{ctrl; plant\}^*]safe \quad (3)$$

i.e., any repetition of $\{ctrl; plant\}$ starting from a state in init will end in a state described by safe. Then $u_i(s_i) \models$ safe for all u_i, s_i satisfying the learning process for

$$(init, (S, A, R, E), choose, update, done, \mathit{CM}, MM)$$

where CM and MM are the controller and model monitor for $init \rightarrow [\{ctrl; plant\}^*]safe$.

Theorem 1 states that whenever the environmental model is accurate, every state we reach via JSC satisfies the safety property *safe*. The proof of this property exploits the fact that any proof of $init \rightarrow [\{ctrl; plant\}^*]safe$ will proceed by identifying a loop invariant (Platzer 2012a).

Policy Extraction The ultimate output of the learning algorithm is a control policy π. Many reinforcement algorithms allow this policy to be extracted after some learning period. If a known-safe fallback policy is provided, Theorem 1 also extends verification results to extracted policies. Formal definitions and proofs for safe policy extraction are presented in the accompanying technical report (Fulton and Platzer 2017).

Experimental Validation

The theoretical results presented in the Provably Safe Learning section apply to generic reinforcement learning algorithms. This section presents two sets of simulations that expore JSC in the concrete setting of classical Q-learning on a simple model of adaptive cruise control. The first set of experiments validate the theoretical results within this concrete setting. The second set of experiments consider a case where the environment deviates from modeling assumptions, violating the key assumption made in Theorem 1. The second set of experiments demonstrate that verification can help improve the learning process itself.

The JSC algorithm is parametric in the approach toward reinforcement learning. We choose Q-learning because it is a simple algorithm, and our setting is simple enough that discretized Q-learning is possible. We leave exploration of JSC-style control in more complex environments, and with more effective reinforcement learning algorithms, as future work.

The setting of this experiment is a simple model of adaptive cruise control. These experiments are implemented in a new linear Adaptive Cruise Control[3] OpenAI Gym environment (Brockman et al. 2016) based on (Loos, Platzer, and Nistor 2011).

Adaptive Cruise Control

Adaptive Cruise Control (ACC) is an increasingly common feature in passenger vehicles. Unlike traditional cruise control, ACC adjusts the speed of a car relative to the speed of a leader car. The safety property for adaptive cruise control

[3]This implementation is available at github.com/nrfulton/JSC

Table 2: Experiment 1: JSC vs. Classical Q Learning in a Modeled Environment

	JSC			Normal		
Training steps	Crash	Fall Behind	Steady	Crash	Fall Behind	Steady
1,000	0	12559	289	10644	2162	42
10,000	0	12538	310	10462	2291	95
100,000	0	12375	473	10492	2284	72

is simple: an actuated follower car must avoid crashing into a leader car.

Our experiment considers Adaptive Cruise Control for two cars. We use relative coordinates to reduce the state space, so instead of two positions $pos_{follower}$ and pos_{leader} we have a single relative position

$$r_{pos} = |pos_{leader} - pos_{follower}|$$

A relative coordinate system allows us to exploit symmetries in the state and action space during learning and increases the likelihood that the system will return to a modeled state after an environmental perturbation.

The relative position of the two cars r_{pos}, is non-negative whenever the cars do not collide. The relative velocity between the cars is zero whenever the cars are stationary relative to one another, positive whenever the cars are moving apart, and negative whenever the cars are moving closer together. We consider a discrete action space – the actuated car may brake with constant force B, accelerate with constant force A, or maintain its current relative velocity.

Model 1 (Relative Acceleration Along a Straight Line)**.**

$$\begin{aligned}
&r_{pos} \geq 0 \wedge A > 0 \wedge B > 0 \wedge T > 0 \wedge r_{pos} \geq \tfrac{r_{vel}^2}{2A} \rightarrow \\
&[\{\{r_{acc} := A \\
&\quad \cup\ ?r_{pos} - \tfrac{-BT + r_{vel}^2}{2A} + r_{vel}T - \tfrac{BT^2}{2} \geq 0; \\
&\qquad r_{acc} := -B \\
&\quad \cup\ ?r_{vel} = 0;\ r_{acc} := 0 \\
&\quad \};\ \{r'_{pos} = r_{vel}, r'_{vel} = r_{acc}, c' = 1 \& c \leq T\} \\
&\}^*]\ r_{pos} \geq 0
\end{aligned}$$

Model 1 presents a dL formula that corresponds to this system. On every iteration of the control loop, the follower actuates the relative acceleration. From this model, we extract controller and model monitors for JSC Q-learning using ModelPlex. Notice that this controller can be rather inefficient. In particular, one safe but inefficient deterministic implementation of this model could choose to always increase the distance between the two cars by choosing $r_{acc} := A$.

Experimental Setup and Results

This section presents two experiments. In the first experiment, we validate the safe learning theorem presented in the previous section. In the second experiment, we go beyond provably safe sandboxing to determine whether verification results might be useful even when models are inaccurate; i.e., even when the *accurately modeled* assumption (as stated in Definition 3) is violated.

JSC in an Accurately Modeled Environment Our first experiment validates our theoretical results by the performance of JSC and Q-learning in an accurately modeled environment. The results of this experiment are recorded in Table 2. We ran training for a specified number of steps ($n = 1,000; 10,0000;$ and $100,0000$). We then iterated across each of the initial states and determined how the policy behaved in each case, omitting in which no safe policy was available (i.e., states where even braking with maximum braking force would result in crashes within the chosen discretized space).

The Crash columns indicate, for the policy extracted by each approach, the number of initial states that resulted in a crashing state. The Fall Behind columns indicate the number of initial states that resulted in a policy that brakes too often; i.e., where the follower car loses track of the lead car. The Steady columns indicate the number of initial states that resulted in an ideal policy – one that does not lose the lead car, but also does not result in a collision.

We observe that the JSC controller never enters unsafe states during training. But we also observe that JSC encounters significantly fewer fall-behind states and more steady states. This result indicates that learning with JSC is more efficient than normal learning. The experimental observation that JSC is more efficient at learning effective policies deserves further exploration.

JSC in an Environment that Admits Speculation The results presented above demonstrate, via proof and experiment, that Justified Speculative Control effectively transfers proofs of correctness from verified models to reinforcement learning algorithms.

In our second experiment (Table 3), we force speculation by simulating an environment that behaves differently from the modeled system. We introduce a slight perturbation to the relative position of the cars (-2 units) with 5% probability. The policy extracted after running our JSC algorithm for any period of time results in relatively few crashes (< 10 out of $20,000$ possible initial states result in a crash). All of these crashing states are attributable to actuator faults.

During each experiment, we extracted the learned policy for JSC and for classical Q-learning after N simulations. We then evaluated the resulting policy from every possible initial position. These results demonstrate that JSC can control reasonably well even when there are small perturbations between the model and the simulated environment.

JSC with a Quantitative Model Monitor In our final experiment, we compared the algorithm presented in this paper

Table 3: JSC vs. Q-learning with Error Injection (.05 error rate)

	JSC			Normal		
Training steps	Crash	Fall behind	Steady	Crash	Fall behind	Steady
1,000	3	12539	306	10950	1745	153
10,000	7	12502	339	10546	2215	87
100,000	5	12359	484	10561	2242	45

with a slightly modified algorithm. The modified algorithm uses the distance between the current state and the modeled environment as an objective function during speculation; when the system leaves the modeled state space, the reinforcement learning agent optimizes for returning to the modeled portion of the state space. We call this approach JSCQ.

Table 4: Crashing states for JSC and JSCQ control

Perturbation	JSC	JSCQ
5%	3	2
25%	18	16
50%	41	34

For small positional perturbations, JSCQ and JSC perform equally well. However, we found that as the positional perturbation increases, the modified algorithm begins to outperform the original algorithm. Table 4 presents these results.

Related Work

Safe Artificial Intelligence is an emerging area of interest, 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 these approaches are summarized in (García and Fernández 2015), who 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.

The primary novel contribution of our work, compared to this body of literature, is two-fold. 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).

Many approaches to safe reinforcement learning work by constraining the optimality criterion used in reinforcement learning. The idea is simple – instead of selecting from the entire policy space, the agent may only choose from a set of control actions that are a priori known to be safe (García and Fernández 2015). There are several approaches with this basic flavor (Kadota, Kurano, and Yasuda 2006; Geibel 2006; Moldovan and Abbeel 2012). Our primary contribution, relative to this work, is that we retain formal verification results for the restricted policy space through the use of provably correct runtime monitors, and allow speculation when policy space restriction becomes unjustified due to deviation from modeling assumptions. Other approaches do not provide such strong results and do not relax constraints when modeling assumptions are violated, but do perform more sophisticated approaches toward learning (we simply perform naive Q-learning on a discretized state space). Incorporating these more sophisticated approaches in a way that retains the safety theorems presented in this paper is left as promising future work.

Another approach toward safe reinforcement learning adopts worst case criterion (Heger 1994) or risk-sensitive criterion (Tamar, Xu, and Mannor 2013; Nilim and Ghaoui 2005), in which the optimization criterion is modified to reflect safety concerns.

Another set of approaches initialize the learner with some initial knowledge, with the goal of directing policy exploration away from unsafe states (Driessens and Dzeroski 2004). Our approach is analogous – for example, in the context of teacher/learner reinforcement learning, our nondeterministic controls could be thought of as an infinite set of demonstrations. The problem in both cases is how to safely control in cases where no demonstration is provided. The primary strength of our approach is that we leverage formal verification, and preserve these results during exploration. We also believe it is important to distinguish between optimization within a known safe policy space, and exploration of inherently speculative control options.

Another approach toward safe reinforcement learning involves analysis of the policy constructed from a learning process (Katz et al. 2017). These approaches are appropriate when the learning phase is not safety-critical, but not appropriate when the system must behave safely *while learning*. We do both and also work with verified models, instead of depending upon conjectures or assumptions about the model. Our approach is also computationally attractive when compared to these approaches, because we enforce

safety during reinforcement learning. Therefore, we directly benefit from improvements to the data efficiency of reinforcement learning algorithms without any additional consideration.

Conclusions

This paper proposes a way of incorporating formal verification results into safe reinforcement learning systems. The theorems and examples explored in this paper validate the feasibility of this idea.

Acknowledgements This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246 and Grant No. CNS-1446712. This research was supported as part of the Future of Life Institute (futureoflife.org) FLI-RFP-AI1 program, grant #2015-143867.

References

Alur, R.; Courcoubetis, C.; Henzinger, T. A.; and Ho, P. 1992. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman, R. L.; Nerode, A.; Ravn, A. P.; and Rischel, H., eds., *Hybrid Systems*, volume 736 of *LNCS*, 209–229. Springer.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *CoRR* abs/1606.01540.

Driessens, K., and Dzeroski, S. 2004. Integrating guidance into relational reinforcement learning. *Machine Learning* 57(3):271–304.

Fulton, N., and Platzer, A. 2017. Safe reinforcement learning via formal methods: Toward safe control through proof and learning (technical report). Technical Report CMU-CS-17-127, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Fulton, N.; Mitsch, S.; Quesel, J.-D.; Völp, M.; and Platzer, A. 2015. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Felty, A. P., and Middeldorp, A., eds., *CADE*, volume 9195 of *LNCS*, 527–538. Springer.

García, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16:1437–1480.

Geibel, P. 2006. Reinforcement learning for MDPs with constraints. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, 646–653.

Heger, M. 1994. Consideration of risk in reinformance learning. In Cohen, W. W., and Hirsh, H., eds., *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, 105–111. Morgan Kaufmann.

ISO. 2011. ISO 26262 road vehicles functional safety.

Kadota, Y.; Kurano, M.; and Yasuda, M. 2006. Discounted Markov decision processes with utility constraints. *Computers and Mathematics with Applications* 51(2):279 – 284.

Kalra, N., and Paddock, S. M. 2016. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. 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*, 97–117.

Loos, S. M.; Platzer, A.; and Nistor, L. 2011. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, 42–56.

Mitsch, S., and Platzer, A. 2016. ModelPlex: Verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.* 49(1):33–74. Special issue of selected papers from RV'14.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *CoRR* abs/1602.01783.

Moldovan, T. M., and Abbeel, P. 2012. 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*.

Nilim, A., and Ghaoui, L. E. 2005. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research* 53(5):780–798.

Platzer, A. 2008. Differential dynamic logic for hybrid systems. *J. Autom. Reas.* 41(2):143–189.

Platzer, A. 2012a. The complete proof theory of hybrid systems. In *LICS*, 541–550. IEEE.

Platzer, A. 2012b. Logics of dynamical systems. In *LICS*, 13–24. IEEE.

Platzer, A. 2017. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.* 59(2):219–265.

RTCA. 2012. DO-178C software considerations in airborne systems and equipment certification.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press. A Bradford Book.

Tamar, A.; Xu, H.; and Mannor, S. 2013. Scaling up robust MDPs by reinforcement learning. *CoRR* abs/1306.6189.