Safe Reinforcement Learning via Probabilistic Timed Computation Tree Logic

Li Qian
East China Normal University
Shanghai, China

Jing Liu *
East China Normal University
Shanghai, China

***Abstract*—Reinforcement learning aims to discover an optimal policy that maximizes reward based on the feedback signal. Although the method succeeds in numerous systems, it may not apply to safe-critical systems due to the absence of safety protection mechanism. Besides, the agent is unable to model the environment accurately if getting biased observation. We present a safe algorithm called *Safe Control with Supervisor* (SCS) for addressing the limitation. If the model is accurate, the supervisor monitors the system and repairs the action of the agent at runtime, which guides the system to obey the specification described by *probabilistic timed Computation Tree Logic* (ptCTL). If not, the supervisor would maximize the probability of satisfying a given task specification. We validate our method through experiments of adaptive cruise control under uncertainty.**

***Index Terms*—Reinforcement learning, Probabilistic timed computation tree logic, Safe control**

I. Introduction

Reinforcement Learning (RL) [1], [2], provides a method for autonomous agents to explore policies that maximize long-term reward. RL succeeds in finding solutions for decision problems. However, there is no safety guarantee during exploring and executing stages in traditional RL. It is fatal to some systems, particularly in which safety is critical. For instance, the mobile robot navigation system and adaptive cruise controller. In these systems, not only maximizing long-term reward should be considered, but also avoiding damage. The safety concept, in many works, has taken various forms in the RL area, is not always refers to physical issues, even optimal policy. However, most are related to the stochasticity of the environment [3].

Traditional RL algorithms aim to find a function that specifies an action or a strategy for some states of the system to optimize a criterion. The optimization criterion may be to maximize rewards, to minimize time or any other cost metric, etc.. There are some optimization criterions of safe RL in research, such as worst-case criterion, risk-sensitive criterion, constrained criterion [3].

A exploring or executing process is capable of being called *safe*, which means that no unsafe states are visited during the process. There are two common ways to modify exploration process to avoid accessing undesirable states: incorporating external knowledge and risk-directed exploration [4], [5].

In this paper, we propose a framework, which allows implementing RL to safe control system. The heart of the framework is a logic called *probabilistic timed Computation Tree Logic* (ptCTL), a probability and timed extension of *Computation Tree Logic* (CTL). ptCTL provides a method to express constraints to the system, such as safety and reliability.

Corresponding author: Jing Liu, jliu@sei.ecnu.edu.cn

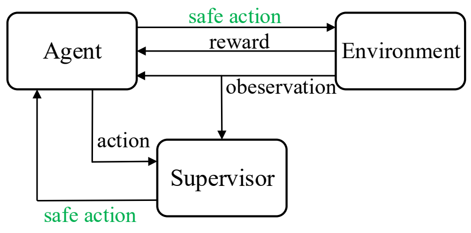

Fig. 1. Safe Control with Supervisor

We contribute an algorithm called *Safe Control with Supervisor* (SCS), combines traditional RL with a formal analysis at runtime. The mechanism of SCS is shown in Fig. 1. The supervisor monitors the chosen action and modifies it only if the action is unsafe. The purpose of the algorithm is generating a safe and optimal control policy π. Furthermore, there is a distinct advantage in that SCS does not concentrate on details of RL algorithms. The independence implies that no matter how complex inner mechanisms of RL algorithms are, SCS may still ensure the system following the given specification.

For accurate models, SCS converts formal verification results into the control strategy. The verification result determines whether the selected action can be taken in the current state. If not, the controller would replace it with a safe action and fix the policy.

However, if the uncertain environment that the agent observed is not accurate, that means the observed state of the environment is not exactly in line with reality. It causes difficulty of optimizing the policy by interacting with the environment. Therefore, we introduce the concept of safety threshold. The optimization criterion is maximizing the probability of safety via quantitive verification. We ensure that the probability that the optimal strategy being safe beyond the threshold.

To validate our approach, we establish an evaluation environment *Adaptive Cruise Control* (ACC) and conduct four experiments of the environment. Experiments measure the performance of the algorithm in different conditions whose model is respectively accurate or not.

Summarily, our main contributions in this paper are:

- Framework for safe RL under uncertainty, provides a way for safe controlling of hybrid dynamical systems by combining formal analysis and RL.
- Proposal and formal definition of ptCTL, a branching time-interval logic with stochastic factor. It provides a manner to express stochastic properties of systems.
- An extended evaluation environment of RL based on OpenAI gym. A model for measuring the performance of RL algorithms.

The remainder of this paper is as follows. In the next section, we discuss the related work about safe RL and safe control. Then we introduce the theories and technologies used in Section III. In Section IV, we define the syntax and semantics of ptCTL. Subsequently, we show our algorithm after the definition of safety threshold and controller monitor in Section V. We construct a new evaluation environment and validate our method base on the environment in Section VI. Finally, we conclude this paper in Section VII.

II. Related Work

In recent years, there has been increasing interest in safe artificial intelligence. A variety of methods and algorithms for strengthening the safety of RL is explored recently [3].

Numerous strategies for safe RL are based on optimizing constrained criteria, that is, excluding unsafe states from state space. The agent only visits the states in the safe state set that satisfies the given specification. In those methods, temporal logic is popular due to their strong ability of expression. One of the most popular temporal logic is *Linear Temporal Logic* (LTL), which provides an approach to express the safety, reliability, and liveness of the system. The reactive system would monitor the states and actions of the agent, and corrects only if the chosen action violates the given specification [6]–[8]. Another popular one is CTL, which is used to describe the properties of branching systems. Recently, some novel logic of specification are emerging for safe RL, e.g. *Probabilistic Signal Temporal Logic* (PrSTL) is used for safe control under uncertainty [9]. Methods below establish risky bound of the system to ensure its safety. Once the agent damages either environment or itself, policies would be repaired [10].

However, temporal logic and their probabilistic versions require space and time discrete, which is not often met in *Cyber-physical systems* (CPS). Especially, in branching systems that contain several state transition relationships at a state. In those systems, the methods mentioned above are unable to fulfill all requirements well. Our method may satisfy the requirements in hybrid probabilistic branching systems. Besides, ptCTL can express specifications that over a specific period due to the introduction of the time interval operator. If the environment observed is not precise, the previously mentioned method does not perform well. Compared with them, our method is more robust. Even though the perceived environment is not accurate, the algorithm still guarantees the safety of the system.

Another optimal criterion for safe learning is worst-case criterion. $\widehat{Q}$-Learning is based on dynamic programming for the minimax criterion. The minimax criterion exclusively focuses on risk-avoidance policies, agent maximizes the reward related to the worst-case policy [11]. In risk-sensitive RL, the agent has to balance between maximizing reward and minimizing the possibility of damage even if risks occur with a minuscule probability. The risk-sensitive criterion based on the usage of exponential utility functions includes a scalar parameter that provides the risk tolerance level to be dominated [12].

Another category for safe RL is modifying the exploration process. Incorporate external knowledge is one of the available methods. Providing initial knowledge [4], [13], initializes prior knowledge of the problem to the agent, for learning more efficient and reducing time spent on random action. Deriving a policy from a finite set of demonstrations is also a method to incorporate external knowledge [14]. A teacher demonstrates the mission, and the state-action tuples of the demonstration are recorded before the learning process [15]. There are two ways for advising by the teacher. First, the teacher guide agent to act in promising space, that suggested by the teacher's policy. In the second method, the teacher advises the learner for avoiding damage only if the teacher deems it essential. The methods above can restrict the agent to a safe state and action space before the learning process.

Risk-directed Exploration [5] is another practical approach to modify the learning process. Take controllability as the reward of the agent means inspiring the agent to explore controllable policy. The method renders the agent explore with a defined risk metric based on controllability. Our work is more closely to teacher-advising RL, due to the supervisor is similar to a teacher, which monitors the agent and generate safe actions. Different from the methods above, our method is monitoring and repair the agent at runtime. The algorithm is more efficient and avoids vast state exploration.

Recently, an interesting concept is proposed, that is runtime monitoring [16], [17]. We adopt this idea and propose our method to guarantee the safety of systems at runtime.

III. Background

In this section, we introduce notations and definitions used throughout this paper. We introduce a description of the hybrid dynamical system and the CTL, then describe the RL.

A. Hybrid Dynamical System

In a continuous state dynamical control system, there exist both discrete and continuous variables. Its behaviors are widely mixed with discrete control signals and continuous real-valued states so that behaviors are complex and challenging to master and control. Mathematically, the lower layer of the system and controller are modeled together as a differential equation evolving in real-time. However, the lower layer is considered as a discrete system consists of events traces that unfold in logical time. For maintaining consistency, it is necessary that correlating two models [18].

Hybrid model semantics combines the continuous and discrete variables by a differential equation:

$$\dot{x}_t = f_t(x) = \frac{dx}{dt} \tag{1}$$

where $\dot{x}_t$ is a discrete variable of system at time t, $f_t(x)$ is a differential function of x, dx is the change of continuous variable x in the time interval dt.

Example 1: Let us consider a hybrid dynamical model (A model of a car driving on a straight road):

$$\dot{x}_t = \frac{dx}{dt} = v_t \;\; ; \;\; \dot{v}_t = \frac{dv}{dt} = a_t \tag{2}$$

Equation 2 describes a model of a car driving on a straight road. The car can choose to accelerate with discrete acceleration: $a = A(A > 0), a = 0$ *or* $a = B(B < 0)$*, and then follows the differential equation above. In the Equation 2,* v*,* x*, and* t *are continuous variables, the value of* a *is discrete. This continuous system can be discretized using time interval* $dt > 0$*.* $\dot{x}$ *is the differential of* x *on* dt*, equals to the velocity* v*, whose differential* $\dot{v}$ *represents the acceleration of the car.*

B. Computation Tree Logic

CTL is a branching logic for expressing finite model properties using path quantifiers [19]. The timed model of CTL is not determined. Formally, the CTL formula is defined as:

$$\begin{aligned} \phi \; ::= \bot \mid \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \\ AX\phi | EX\phi | AF\phi | AG\phi | EG\phi | A(\phi U \phi) | E(\phi U \phi) \end{aligned} \tag{3}$$

where ϕ is a CTL formula, composing of predicates and Boolean operators. $\top$ and $\bot$ are Boolean constants for *True* and *False*, p is atom formula, $\neg$ is a negation which means *not*, $\wedge$ is a conjunction which indicates *and*, $\vee$ is a disjunction which represents*or*, U is the $Until$ temporal operator, F is the $Future$ temporal operator, G is the $Globally$ temporal operator. E and A are path quantifiers, E is the $Exist$ path quantifiers that means there exists a path satifies target properties, A is the All path quantifiers that means for all paths in the model satisfy given properties.

The specification described by CTL allows the expression of system properties. However, it is not capable of expressing task missions under uncertainty and requires variables in the model to be discrete.

C. Reinforcement Learning

Reinforcement learning is a method to maximize rewards in *Markov Decision Processes* (MDPs) that defined as a tuple $\mathcal{M} = (S, A, T, R)$. In the tuple, S is a finite set of system states, a unique initial state $s_0 \in S$; A is a finite set of available actions of the agent while under a state $s \in S$; $T(s, s')$ denotes a probable transition function from state $s \in S$ to $s' \in S$ by action $a \in A$: $S \times A \rightarrow S$, R is an immediate reward function related to transition $t \in T$, $S \times A \times S \rightarrow R$.

The objective of RL is to explore an optimal policy $\pi : S \rightarrow A$ that makes rewards maximum. Policies consist of a set of selection strategies for choosing actions based on the current state. Mechanism of standard RL as shown in Fig. 2 [1]. In standard RL, the agent consists of three parts: observer I, learner L and decision-maker D. I transforms the observation from the environment into internal arguments i, L updates the policy according to reward from the environment and i from

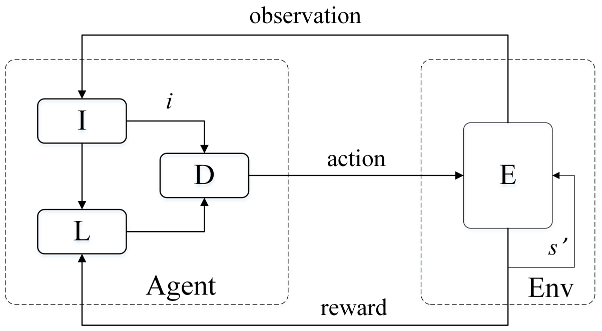

Fig. 2. Standard reinforcement learning

I, then D would choose the most suitable action based on internal argument i and current state.

Q-learning is a form of model-free reinforcement learning. Agent adjusts Q-values via Bellman equation [20]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma max_{a'} Q\left(s', a'\right) - Q(s,a)\right] \tag{4}$$

where $\alpha \in (0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discounter factor that trades off the importance of immediate and later rewards. By selecting the highest value action in each state, the optimal strategy can be easily derived.

Unlike traditional techniques, Q-network stores Q-values through a network instead of Q-table to solve the problem since Q-table cannot read the Q-value fastly if the number of states is too large.

IV. Probabilistic Timed Computation Tree Logic

We propose a probabilistic timed extension of CTL, that supports stochastic temporal properties on real-value and dense-time states, and offers a quantitative measure for probabilistic specification. ptCTL has a new ability that specifies stochastic properties during a time stage, applies to constrain synthesis controller. A mission specification defined by ptCTL is capable of expressing safety, reliability, and other complicated properties based on the stochastic models.

A. Syntax

ptCTL is defined with respect to discrete valued time t. The syntax of ptCTL is defined using $Backus\_Naur$ normal form as:

$$\varphi \;\; ::= \;\; \bot \,|\top|\neg\varphi|\varphi_1 \vee \varphi_2|\varphi_1 \wedge \varphi_2|\psi|E\psi|A\psi|P_{\sim\lambda}[\varphi] \tag{5}$$

$$\psi \;\; ::= \;\; \mu|\neg\psi|X\psi|\psi_1 U_{[a,b]}\psi_2|F_{[a,b]}\psi|G_{[a,b]}\psi \tag{6}$$

Here, $\varphi, \varphi_1, \varphi_2$ are ptCTL formulae, ψ, ψ_1, ψ_2 are ptCTL formulae without probabilistic operators. Formulae are composing of predicates and Boolean operators. $\top$ and $\bot$ are Boolean constants for *True* and *False*, μ is a atom formula, $\neg$ is a negation, which means *not*, $\wedge$ is a conjunction, which means *and*, $\vee$ is a disjunction, which means *or*, U is the $Until$ trmporal operator, F is the $Future$ temporal operator, G is the $Globally$ temporal operator, scopes that all state in the current path. E and A are path operators, E is the $Exist$ operator, A is the All. Different from CTL, there are additional stochastic and time factors., $P_{\sim\lambda}$ is probabilistic operator, $\sim \in \{<, \leq, \geq, >\}$, $\lambda \in [0,1]$ is a constant. $[a,b]$ denotes the time period from a to b after current state, $a, b \in [0, +\infty)$ and $b \geq a$. With atomic propositions and operators, complex tasks specifications can be defined.

B. Semantics

$(\mathcal{M}, s, t) \vDash \varphi$ implies that the model $\mathcal{M}$ in state s satisfies the formula φ at time t. For expressing stochastic properties of the model, we introduce the probabilistic statement $P_{\sim\lambda}[\cdot]$ with the stochastic factor λ:

$$(\mathcal{M}, s, t) \vDash P_{\sim\lambda}[\psi] \;\Leftrightarrow\; Pr((\mathcal{M}, s, t) \vDash \psi) \sim \lambda \tag{7}$$

Here, ψ is a ptCTL formula without probabilistic operator, $\lambda \in [0,1]$ is a constant, determines the tolerance level in satisfaction of the probabilistic properties, $\sim \in \{<, \leq, \geq, >\}$. For instance, formula $(\mathcal{M}, s', t') \vDash P_{\geq p'}[\psi']$ holds means the probability that in state s' the model $\mathcal{M}$ satisfying ψ' is higher than p' at time t'. The probability of the formula holding is accessible to be obtained by iterate the value of λ and determine if the formula holds. $P_{\sim\lambda}[\cdot]$ is used to determines weather the probability of satisfying the given specification holds true for $\sim \lambda$. $Pr(\cdot)$ computes the probability of the event holding by:

$$Pr((\mathcal{M}, s, t) \vDash \psi_{[a,b]}) = \textstyle\prod_{t+a}^{t+b}(Pr_{t'}(\psi)) \tag{8}$$

where $\psi_{[a,b]}$ is the properties ψ during the time interval $[t + a, t + b]$, $Pr_{t'}(\psi)$ is the probability of $\mathcal{M} \vDash \psi$ at time $t' \in [t + a, t + b]$. The probability of the formula ψ holding is the product of the possibility of ψ holding in every time step t'.

Moreover, we define the semantics of ptCTL with a satisfaction relation $\vDash$ over the states and paths of a MDP model $\mathcal{M} = (S, A, T, R)$, the satisfaction relation defined as:

$(\mathcal{M}, s, t) \vDash \mu$	$\Leftrightarrow$	at time $t, \mu \in L(s)$
$(\mathcal{M}, s, t) \vDash \neg\varphi$	$\Leftrightarrow$	$(\mathcal{M}, s, t) \nvDash \varphi$
$(\mathcal{M}, s, t) \vDash \varphi_1 \wedge \varphi_2$	$\Leftrightarrow$	$(\mathcal{M}, s, t) \vDash \varphi_1 \wedge (\mathcal{M}, s, t) \vDash \varphi_2$
$(\mathcal{M}, s, t) \vDash \varphi_1 \vee \varphi_2$	$\Leftrightarrow$	$(\mathcal{M}, s, t) \vDash \varphi_1 \vee (\mathcal{M}, s, t) \vDash \varphi_2$
$(\mathcal{M}, s, t) \vDash P_{\sim\lambda}[\varphi]$	$\Leftrightarrow$	$Pr((\mathcal{M}, s, t) \vDash \varphi) \sim \lambda$
$(\mathcal{M}, s, t) \vDash X\varphi$	$\Leftrightarrow$	$(\mathcal{M}, s', t') \vDash \varphi$, (s', t') is the next state of (s, t)
$(\mathcal{M}, s, t) \vDash G_{[a,b]}\varphi$	$\Leftrightarrow$	$\forall t' \in [t{+}a, t{+}b], (\mathcal{M}, s, t') \vDash \varphi$
$(\mathcal{M}, s, t) \vDash F_{[a,b]}\varphi$	$\Leftrightarrow$	$\exists t' \in [t{+}a, t{+}b], (\mathcal{M}, s, t') \vDash \varphi$
$(\mathcal{M}, s, t) \vDash \varphi_1 U_{[a,b]} \varphi_2$	$\Leftrightarrow$	$\exists t' \in [t + a, t + b], (\mathcal{M}, s', t') \vDash \varphi_2 \wedge \forall t'' \in [t, t'], (\mathcal{M}, s'', t'') \vDash \varphi_1$
$(\mathcal{M}, s, t) \vDash E\ \varphi$	$\Leftrightarrow$	Existing a path start at state s satisfies $\forall t' \in [t, +\infty], (\mathcal{M}, s', t') \vDash \varphi$
$(\mathcal{M}, s, t) \vDash A\ \varphi$	$\Leftrightarrow$	For all paths start at state s satisfies $\forall t' \in [t, +\infty], (\mathcal{M}, s', t') \vDash \varphi$

In ptCTL formulae, $(\mathcal{M}, s, t) \vDash \varphi$ means φ is satisfied by model $\mathcal{M}$ at state s and time point t. For any time $t \in [0, +\infty)$ and state $s \in S$ in model $\mathcal{M}$, we can determine satisfaction status by the semantics above.

Example 2: (Safety specification of a car driving on a straight road)

$$EG_{[0,+\infty)}(dis > 0 \wedge v \geq 0) \tag{9}$$

Equation 9 represents a safety specification of a car driving on a straight road. dis represents the distance between the vehicle and others in front, v is the velocity of the car. The specification means that there is at least one running path throughout the driving, makes the distance of the vehicle from other vehicles is greater than 0 and that the vehicle speed must be greater than 0.

Given a MDP model $\mathcal{M} = (S, A, T, R)$, and a ptCTL formula φ, we use the probabilistic model checking (PMC) for determine if $(\mathcal{M}, s, t) \vDash \varphi (s \in S)$, t is the time point of state s. PMC is decidable in time exponential in linear in the size of $\mathcal{M}$ and the size of φ [21], [22].

V. Safe Learning

In this section, we define the concept of the safety threshold and establish the controller monitor to monitor the system. Based on the above definitions and techniques, we detail the complete method for controlling a hybrid system in an uncertain environment.

Safe Control with Supervisor (SCS) is a risk-sensitive method based on constrained criterion optimization. This method does not concentrate on specific RL algorithms. It is available for most of RL algorithms, from Q-learning to DQN, even Double-DQN [23]. An action is *unsafe* at the current state implies that the action would cause the agent to visit an undesirable state. We hope to guarantee that the probability of performing an unsafe action is very low in $\mathcal{M}$. This amounts shows that the probability of reaching safe states is reachable high, because φ abstracts all the properties desired during learning or executing.

Definition 1: **Safety Threshold** A probability factor $\alpha \in (0, 1]$ is the safety threshold that describes the safe level to guarantee that the optimal solution is upper than the lower safety bound. The safety threshold applied to ptCTL formally defined as:

$$(\mathcal{M}, s, t) \vDash P_{\geq\alpha}(\varphi) \tag{10}$$

It implies the probability of formula φ holding is higher than α. φ can express safety, reliability, and other system properties. Typical assertions that can be checked in this context, e.g., the probability of a failure being less than 5% is equivalent to the formula for verifying that $(\mathcal{M}, s, t) \vDash P_{\geq 0.95}(\varphi_{safe})$ holds, where φ_{safe} expresses the safety property.

A. Generic Safe Control Algorithm with Supervisor

The SCS algorithm assures two properties: safety and interference-minimization. SCS balances between safety and reward via quantifying risk. The supervisor monitors states and actions of the system at runtime and intervenes when the hazard is foreseen.

Definition 2: **Controller Monitor** is a Boolean function that determines whether the chosen action violates the given specification in the current state. The monitor has two inputs: current state $s \in S$, chosen action $a \in A$, and ptCTL formula φ. The function can be written as:

$$(s, a, \varphi) \to bool \tag{11}$$

Algorithm 1 Safe Control with Supervisor

```
Input: MDP model $\mathcal{M} = (S, A, T, R)$, ptCTL formula $\varphi = P_{\geq\alpha}[\psi]$
Output: policy $\pi$
 1: Initilize($\pi$)
 2: $s = s_0;\ a = NOP$
 3: while ($s! = done$) do
 4:     $A' = A; A_\varphi = pmc(A)$;
 5:     $a = choose(A')$;
 6:     while $is\_danger(s, a)$ do
 7:         $A' = A' - a$;
 8:         if $A'! = \emptyset$ then
 9:             $a = choose(A'_\varphi)$;
10:         else
11:             if $model\ is\ accurate$ then
12:                 $a = choose(A)$;
13:             else
14:                 $a = argmax(Pr[\psi](A))$;
15:     $action = a$;
16:     update($\pi, s, action$);
```

In hybrid systems, the monitor converts the continuous variable to the discrete control signal through the hybrid semantics described in Section III-A. Combining the control signal and the specification of the MDP model, the monitor would calculate and output the verification result easily.

Our method takes different strategies for accurate models and inaccurate models. Once the agent chooses an action, it would be judged whether dangerous. If the observed model is accurate, the monitor would verify that the action selected by the agent conforms to the specification and then modify the action based on the verification result. On the contrary, in an inaccurate environment, if the selected action is dangerous, the synthesis controller would replace the originally selected action by the safest action (the probability of being safe is the highest) in the action space.

The content below describes a generic safe control algorithm with supervisor. The inputs of the algorithm are MDP model $\mathcal{M} = (S, A, T, R)$, ptCTL formula φ express the constraints to the system. Finally, the algorithm would output an optimal policy that complies with the given specification.

Algorithm 1 describes a generic safe control algorithm. The inputs of the algorithm are MDP model $\mathcal{M} = (S, A, T, R)$ and ptCTL formula $\varphi = P_{\geq\alpha}[\psi]$ which express the constraints to the system. Finally, the algorithm would output an optimal policy π that complies with the given specification.

We show our strategy of safe control under the uncertainty in Algorithm 1. The supervisor monitors at runtime to ensure that, when the system is accurately modeled, only the safe actions are taken. Starting with the initializing policy π, state s and action a (Line 1-2). The state $done$ means the current state is terminal (Line 3). A' is a copy of A, A_φ is a set that extract actions in A would obeys the specification φ via $pmc(\cdot)$(Line 4). After the agent selecting an action from action set by the fuction $choose(\cdot)$ (Line 5). The controller would verify whether the action follows the specification(Line 6) by a function $is\_danger(\cdot)$ which is the manifestation of controller monitor. The function $is\_danger(\cdot)$ returns $true$ when the action a in the state s would cause damage and cannot avoid in some future states. If it is dangerous and A' is not empty, controller would select a safe action from the set A'_φ that include actions in A' would obeys the specification φ (Line 7-9). If the set A' is empty the model can be modeled accurately, the action would be original choice in Line 5 (Line 11-12).

In the case of an inaccurate modeling environment, we adopt a strategy that is not the same as the one in the accurate model. The method ensures that only the safest actions are taken when the obsevation is not precise. Similarly, as long as the set A' is not empty, SCS would choose an action which is the safest in A'. On the contrary, if whole actions in the set A may not satisfy the specification, the controller would select the safest in A (Line 14) via quantifying risk. $Pr[\varphi](\cdot)$ represents the probability of formula φ holds. On the principle of interference-minimization, if the action is not dangerous, the agent would take original action without a doubt (Line 15). Finally, update the policy π (Line 16).

The SCS algorithm ensures the learned controller is intelligent enough to avoid entering an unsafe state as far as possible. If the model is accurate, only verified safe actions can be taken. If not, the algorithm assures that the safest actions would be taken for the expected number n time steps. The final output of the algorithm is the policy π. After the period of safe learning, the agent would extract a safe and optimal policy.

VI. Experiments

In this section, we implement our safe control algorithm to a simple ACC model we establish. In the first experiment, the agent can observe the environment without error. The second is a challenging environment that the agent cannot perceive the external environment accurately, there are errors between the observed state and the actual state. We validate the robustness of SCS by analyzing the performance of the algorithm in inaccurate models with different $error\_rate$. For each experiment, we accumulate and record rewards obtained by agent with the SCS. Additionally, we study the relationship between the specification and the safety.

Q-leaning is a generic and efficient method for RL, and it is simple enough to validate our approach lightly. The setting of the model is a new environment based on OpenAI gym [24].

A. Adaptive Cruise Control

ACC (shown in Fig. 3) is a dynamic hybrid uncertain environment and is a challenging task due to the complexity to predict behaviors of other cars. ACC requires the car to adjust its action based on state of the leader. The safety property of the system is to avoid collisions between two cars. Other complex properties may also be specified for the system, such as reliability and liveness. The remainder left in this paper as our future work. In this paper, we only consider safety.

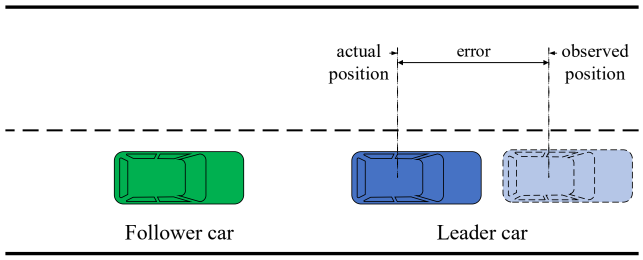

Fig. 3. Adaptive Cruise Control

In our experiments, the envoironment is a MDP model $\mathcal{M} = (S, A, T, R)$. State $s \in S$ is a tuple of relative position and relative velocity between two cars that can be measured by:

$$s := (pos_{rel}, vel_{rel}) \tag{12}$$

$$pos_{rel} := pos_{leader} - pos_{follower} \tag{13}$$

$$vel_{rel} := vel_{leader} - vel_{follower} \tag{14}$$

where pos_{rel} is relative distance between position of leader car and follower car, vel_{rel} is relative velocity between two cars that come from solving the differential equation $\dot{x} = v$,$\dot{v} = a$.

For simplifying action space, there are three actions can be taken: accelerate with the acceleration a, driving at a constant speed, or decelerate with the deceleration d. Action space is:

$$A = \{acc, con, dec\} \tag{15}$$

The relative position of two cars after next time step may be calculated according to current state and action:

$$pos_{rel} := pos_{rel} - \frac{1}{2}a_{rel} * t^2 - vel_{rel} * t \tag{16}$$

$$a_{rel} := a_{follower} - a_{leader} \tag{17}$$

where $t = nT$ is expected time interval of n time steps T, a_{rel} is a relative measure of acceraleration of two cars that represented respectively by $a_{follower}$ and a_{leader}.

Furthermore, we build an inaccurate environment that agent cannot observe the system states precisely. There is an error between actual position and observed position of the leader car. We can manipulate the error magnitude by assigning error rate, and the environment would raise or reduce actual position as observed position randomly, within the range of error rate:

$$\begin{aligned} pos_{leader} :=& pos_{leader} + error \\ :=& pos_{follower} + (pos_{leader} - pos_{follower}) \\ & * random(-error\_rate, +error\_rate) \end{aligned} \tag{18}$$

In every step, the environment would calculate the current state according to the mechanism above fastly. And then the environment would select an action for the leader car. Once the agent enters an unsafe state, it would get a huge penalty and the iteration would be stoped immediately. For encouraging the car to keep up with vehicle ahead, the environment punish the follower if it falls behind, but the penalty is much smaller than collision. We implement penalties by deducting rewards, and the amount of deduction depends on the type of punishment.

B. Experiments Setup and Results

This section presents four experiments. The first experiment validates the safety of learned agent in the accurate environment. The second verifies the validation of SCS in the inaccurate environment. Besides, we study the performance of the algorithm in the environment with different $error\_rate$. Extracted policies are evaluated in the same environments, to measure the safety of the policy learned by SCS. We implement the SCS algorithm using Q-network, which supports a vast number of states and actions. Then, the effect of ptCTL specification is considered. In our experiment, we use a simple Q-network structure that consists of three layers: input layer contains 2 nodes, hidden layer contains 8 nodes, and output layer contains 3 nodes.

When the environment can be modeled accurately, we assign the probability λ in specification $\lambda = 1$, means the constraint munst be satisfied. If not, $\lambda = 0.95$ due to errors between actual state and the observed. Additionally, we evaluate the effect of different count of expected time steps in the specification.

1) SCS in Accurate Environment: The first experiment validates the algorithm by performance of SCS in a precise model. We run training of agent for episode $= 300, 500, 1000, 1500, 2000$. The specification is:

$$P_{\geq 1}(EG_{[0,5T]}(pos > 0 \wedge v \geq 0 \wedge v \leq 100)) \tag{19}$$

Equation 19 is the constraint point that, in the next 5 time steps (time step number determined by experiment in section VI-B3), that relative position must greater than 0 and the velocity of follower car is between 0 and 100.

Then we test the learned policies in the same environments and record the number of collisions. After that, we compare the learning process and test result of SCS with the normal Q-learning for reflecting the performance improvement.

Accumulated reward per episode in early 200 episodes that represent the trend of learning shown in Fig. 4. Accumulated reward is the cumulative sum of the rewards that have been obtained at previous steps. We compare the learning process of SCS (blue, solid) and normal Q-learning (red, dotted). The result implies that the agent with SCS can avoid the collision effectively after enough episodes of training. Although the normal version can avoid the crash to some extent after more episodes, it still has some collisions. The frequency of collisions is not ignorable and is intolerable in the real system. After training, the learned policies would be tested in the environment. Table I shows the comparison of different algorithms' performance.

Table I indicates that the agent never visits any unsafe state after a certain number of SCS training episodes. However, Fig. 4 shows that the follower car falls behind the leader car in a few states. Although the states are not unsafe, it is not desired states, a fly in the ointment. The performance of the normal algorithm is unacceptable and collisions still occur after a sizable number of training episodes. In summary, SCS guarantees the safety of the system effectively in the accurate ACC environment.

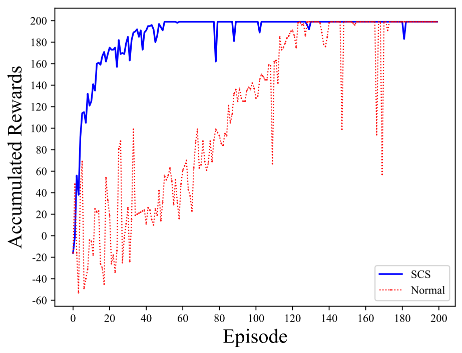

Fig. 4. Accumulated Rewards per episode in Accurate Environment

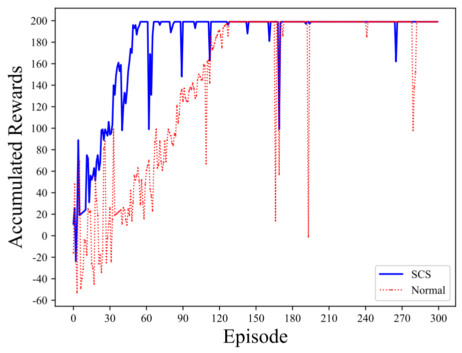

Fig. 5. Accumulated Rewards per episode in Inaccurate Environment ($error\_rate = 0.1$)

TABLE I
SCS vs. Normal Q-learning in Accurate Environment ($test\ episodes = 1000$)

Train episodes	SCS		Normal	
	Crash	Fall-behind	Crash	Fall-behind
300	0	0	63	6
500	0	3	16	28
1000	0	1	12	2
1500	0	0	1	6
2000	0	0	0	12

TABLE II
SCS vs. Normal Q-learning in Inaccurate Environment($test\ episodes = 1000, error\_rate = 0.1$)

Train episodes	SCS		Normal	
	Crash	Fall-behind	Crash	Fall-behind
300	13	98	194	356
500	9	29	93	128
1000	0	1	59	161
1500	0	0	73	134
2000	0	3	24	56

2) SCS in Inaccurate Environment: For validating that the SCS is a robust algorithm that can perform well even if the model is not accurate, this experiment considers an agent with the SCS algorithm to explore the policy in the inaccurate model. Similar to the accurate, we run training of agent for episode $= 300, 500, 1000, 1500, 2000$ and test the learned policy. But the specification is different:

$$P_{\geq 0.95}(EG_{[0,5T]}(pos > 0 \wedge v \geq 0 \wedge v \leq 100)) \quad (20)$$

The specification (Equation 20) requires the probability of constraint satisfying is higher than 0.95. Then we implement the learned policies to the same environment and record the number of accidents. Finally, we compare the learning process and evaluation result of SCS (blue, solid) with the normal Q-learning (red, dotted).

Fig. 5 implies that the agent after training of SCS can avoid collisions efficiently. SCS enables the agent to get more rewards than normal Q-learning. The normal algorithm increases the accumulated rewards obtained by the agent through the training but still crashes after training. After training, the learned strategies would be estimated in the same environment, the result of performance is shown in Table II.

Table II lists number of collisions in testing policies learned by different methods. SCS can avoid most of collisions, yet due to incorrect observations, the agent inevitably enters a few unsafe states occasionally. The normal algorithm exhibits poor performance when observations are not entirely precise.

Besides, we compare the performance of the proposed algorithm in inaccurate environments with different $error\_rate$.

Table III presents that SCS performs well and much better than the normal algorithm when the error is not large. With the error increasing, although the frequency of crash increases, SCS still works much better than the normal, confirms the robustness of the algorithm.

3) SCS with Different Specification: For studying the influence of ptCTL specification to the system, we adopt different ptCTL formula as the constraint to the system. The difference between those formulae is unequal expected time steps n. We use the time steps of $n = 1, 2, ..., 10$ to train the agent, then evaluate the policy in the ACC environment.

Fig. 6 describes the crash numbers of distinct specifications with different expected time steps. When n is too small, there is no enough time to prevent the coming danger, although the supervisor has already realized that danger would appear. With the growing of n, the safety improves. The agent may need to consider more time steps in the model if the environment cannot be observed accurately. However, n does not need to be too large to avoid unnecessary computation and other unexpected problems.

VII. Conclusion

We present an algorithm SCS to guarantee the safety of RL. The method combines reinforcement learning with formal verification to monitor and correct system behavior. The main contributions include proposing ptCTL, a logic to express complex stochastic task specifications and properties. Besides, we establish a simple evaluation environment of safe learning based on OpenAI gym. Through experiments, we validate the

TABLE III
COLLISION OF SCS VS. NORMAL Q-LEARNING IN INACCURATE ENVIRONMENT WITH DIFFERENT $error\_rate$ $(train\ episodes = 500,\ test\ episodes = 1000)$

$error\_rate$	SCS		Normal	
	Crash	Fall-behind	Crash	Fall-behind
0.05	0	1	36	105
0.1	9	29	93	128
0.15	22	95	234	84
0.2	18	47	225	109
0.3	41	72	316	86

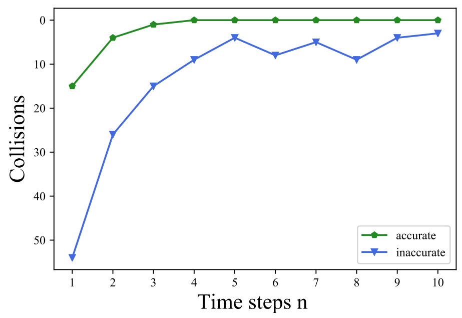

Fig. 6. Relationship between specification and safety $(train\ episodes = 1000, test\ episodes = 1000)$

availability and robustness of the method. SCS ensures the safety of hybrid dynamical systems. The performance of SCS is much better than classical Q-learning regardless of whether the model is accurate or not.

Furthermore, the algorithm may be extended easily to be appropriate for other complex systems and RL algorithms. Even the approach can be extended to implement to other research areas, not limited to RL and safe control.

ACKNOWLEDGEMENT

This work was supported in part by the National Key Research and Development under Project 2019YFA0706404, in part by the NSFC under Project 61972150, and in part by the Shanghai Knowledge Service Platform under Project ZF1213.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[3] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
[4] K. Driessens and S. Džeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.
[5] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1037–1044.
[6] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[7] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019, pp. 6065–6073.
[8] J. Krook, L. Svensson, Y. Li, L. Feng, and M. Fabian, "Design and formal verification of a safe stop supervisor for an automated vehicle," in *2019 International Conference on Robotics and Automation (ICRA), Palais des congres de Montreal, Montreal, Canada*, 2019, pp. 5607–5613.
[9] D. Sadigh and A. Kapoor, "Safe control under uncertainty," *arXiv preprint arXiv:1510.07313*, 2015.
[10] S. Pathak, L. Pulina, and A. Tacchella, "Verification and repair of control policies for safe reinforcement learning," *Applied Intelligence*, vol. 48, no. 4, pp. 886–908, 2018.
[11] M. Heger, "Consideration of risk in reinforcement learning," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 105–111.
[12] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," *arXiv preprint arXiv:1205.4810*, 2012.
[13] R. Noothigattu, D. Bouneffouf, N. Mattei, R. Chandra, P. Madan, K. R. Varshney, M. Campbell, M. Singh, and F. Rossi, "Teaching AI agents ethical values using reinforcement learning and policy orchestration," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019, pp. 6377–6381.
[14] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 1–8.
[15] M. Pecka and T. Svoboda, "Safe exploration techniques for reinforcement learning–an overview," in *International Workshop on Modelling and Simulation for Autonomous Systems*. Springer, 2014, pp. 357–375.
[16] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield synthesis," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 533–548.
[17] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[18] A. Gollu and P. Varaiya, "Hybrid dynamical systems," in *Proceedings of the 28th IEEE Conference on Decision and Control,*. IEEE, 1989, pp. 2708–2712.
[19] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer programming*, vol. 2, no. 3, pp. 241–266, 1982.
[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[21] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.
[22] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 2002, pp. 200–204.
[23] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.
[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.