ARTIFICIAL INTELLIGENCE

# A formal methods approach to interpretable reinforcement learning for robotic planning

Xiao Li[1]*, Zachary Serlin[1], Guang Yang[2], Calin Belta[1,2]



**Growing interest in reinforcement learning approaches to robotic planning and control raises concerns of predictability and safety of robot behaviors realized solely through learned control policies. In addition, formally defining reward functions for complex tasks is challenging, and faulty rewards are prone to exploitation by the learning agent. Here, we propose a formal methods approach to reinforcement learning that (i) provides a formal specification language that integrates high-level, rich, task specifications with a priori, domain-specific knowledge; (ii) makes the reward generation process easily interpretable; (iii) guides the policy generation process according to the specification; and (iv) guarantees the satisfaction of the (critical) safety component of the specification. The main ingredients of our computational framework are a predicate temporal logic specifically tailored for robotic tasks and an automaton-guided, safe reinforcement learning algorithm based on control barrier functions. Although the proposed framework is quite general, we motivate it and illustrate it experimentally for a robotic cooking task, in which two manipulators worked together to make hot dogs.**

## INTRODUCTION

Robotic systems that are capable of learning from experience have recently become more commonplace. These systems have demonstrated success in learning difficult control tasks. However, as tasks become more complex and the number of options to reason about becomes greater, there is an increasing need to be able to specify the desired behavior in a structured and interpretable fashion, guarantee system safety, and conveniently integrate task-specific knowledge with more general knowledge about the world. This paper addresses these problems specifically in the case of reinforcement learning (RL) by using techniques from formal methods.

Experience and prior knowledge shape the way humans make decisions when asked to perform complex tasks. Conversely, robots have had difficulty incorporating a rich set of prior knowledge when solving complex planning and control problems. In RL, the reward offers an avenue for incorporating prior knowledge. However, incorporating such knowledge is not always straightforward using standard reward engineering techniques. This work presents a formal specification language that can combine a base of general knowledge with task specifications to generate richer task descriptions that are interpretable. For example, to make a hot dog at the task level, one needs to grab a sausage, grill it, place the cooked sausage in a bun, apply ketchup, and serve the assembled hot dog. Prior knowledge about the context of the task, e.g., sausages can be damaged if squeezed too hard, should also be taken into account.

Interpretability in RL rewards—easily understanding what the reward function represents and knowing how to improve it—is a key component in understanding the behavior of an RL agent. This property is often missing in reward engineering techniques, which makes it difficult to understand exactly what the implications of the reward function are when tasks become complex. Interpretability of the reward allows for better value alignment between human intent and system objectives, leading to a lower likelihood of reward hacking (*1*) by the system. The formal specification language presented in this work has the added benefit of being easily interpretable from the beginning because the language is very similar to plain English.

Safe RL, guaranteeing that undesirable behaviors do not occur (i.e., collisions with obstacles), is a critical concern when learning and deployment of robotic systems happen in the real world. Safety for these systems not only presents legal challenges to their wide adoption but also raises risks to hardware and users. By using techniques from formal methods and control theory, we provide two main components to ensure safety in the RL agent behaviors. First, the formal specification language allows for explicit definition of undesirable behaviors (e.g., always avoid collisions). Second, control barrier functions (CBFs) are used to enforce the satisfaction of the safety specifications.

### Related work

Value alignment and policy verification are two important use cases of interpretability in RL. Value alignment (*2*) focuses on ensuring that the objectives given to the learning systems align with human's intentions. This is important because RL agents are capable of exploiting faulty objectives to gain a high return. In policy verification, the goal is to understand how and why the learned policy makes certain decisions. This is important for ensuring safe deployment of RL agents.

In value alignment, the objective of a task can be expressed in a reward-based or reward-free fashion. Reward shaping (*3*) has been a popular approach to create dense reward functions. On one hand, for complex tasks, shaping a dense reward function that aligns well with the true task objective can be a challenge. On the other hand, learning from demonstrations allows agents to learn without an explicit reward function. Imitation learning and behavior cloning (BC) are efforts in this direction. However, learning solely from demonstrations faces challenges, such as distribution shift (accumulative error resulting from deviation of state and action distributions from demonstrations). Inverse RL (IRL) aims to alleviate this problem by combining demonstrations with exploration. In IRL, demonstrations are used to learn a reward function that is then fed to an RL agent. One problem with IRL in practice is that the inaccuracy of the learned reward function can result in compound error in the learned policy. In addition, the incomprehensibility of the parameterized reward function makes it

[1]Department of Mechanical Engineering, Boston University, Boston, MA, USA. [2]Division of Systems Engineering, Boston University, Boston, MA, USA. *Corresponding author. Email: xli87@bu.edu

difficult to improve using prior knowledge. More recent efforts in value alignment include iterated amplification (*4*) and cooperative IRL (*5*), where the agent and human work together to learn the desired policy (possibly in an iterative process). The authors of (*6*) provided an up-to-date overview of recent progress in value alignment. Compared with these efforts, the method presented here focuses on a simpler, more structured, and deterministic way of specifying complex behaviors.

In policy verification, the authors of (*7*) used probabilistic model checking to verify policies against constraints specified in probabilistic temporal logic. In their approach, the environment was modeled as an abstract Markov decision process (MDP). The authors of (*8*) proposed a method to learn and verify decision tree policies. In the domain of deep RL, policy verification largely falls into the category of neural network verification. The authors of (*9*) provided an up-to-date survey of this area. Policy verification is commonly used after training as a means to interpret the learned policy. In practice, policy verification can also be used during training to ensure that training stops only when a policy with desirable properties is obtained. However, this can be computationally demanding for policies with a large number of parameters, and state-of-the-art methods in this area can only verify simple properties.

In this work, we use temporal logic to express the task objective and use its corresponding automaton to guide learning. Previous work on using temporal logic in an RL setting includes (*10*) and (*11*), which combined temporal logic and automaton to solve the non-Markovian reward decision process. In (*12*) and (*13*), the authors took advantage of the robustness degree of signal temporal logic (STL) to motivate the learning process. The authors of (*14*) incorporated maximum-likelihood IRL with task constraints in the form of co-safe linear temporal logic. In (*15*), the authors used a finite-state automaton (FSA) as a safety measure that supervises the actions issued by the agent. The authors of (*16*) developed a reward function based on linear temporal logic and deterministic Rabin automaton and converted the RL problem into a nonconvex max-min optimization. Our effort mainly focused on establishing a safe and learnable planning system that extracts necessary context from the TL-based task specification. In (*17*), the authors introduced a variant of linear temporal logic (geometric linear temporal logic) that could be converted to learnable specification MDPs. The authors of (*18*) presented a technique that compiles ω-regular properties into a limit-deterministic Buchi automaton, which resulted in a ω-regular reward that the RL agents can learn from. In (*19*), the authors introduced the reward machine, which closely reassembles an automaton, and demonstrated its use on RL benchmarks. In (*20*), the logic-based value iteration network that incorporates the FSA into value iteration using demonstrations was proposed. More distant work on learning and planning under non-Markovian rewards were presented in (*21*), (*22*), and (*23*).

We did not formally verify the learned policy but instead prevented violation of the task by specifying undesirable properties as temporal logic formulas and enforced them using CBFs (*24*) while minimizing interference with the decisions made by the RL agent. The CBFs effectively added a "shield" to the learned policy with strong safety guarantees and can be implemented with high efficiency and scalability. Combining CBF with RL has been looked at by the authors of (*25*) and (*26*), whereas the authors of (*27*) made an effort to integrate temporal logic with CBF. Here, we present a novel framework for combining all three elements in continuous state and action spaces.

In comparison to many previous work on combining temporal logic/automaton with RL in discrete environments, we put much emphasis on demonstrating the effectiveness of this combination in high-dimensional robotic domains. Specifically, the contributions of our method include (i) designing of an automaton-guided dense reward over continuous state and action spaces, (ii) using the automaton to simultaneously provide safety constraints for the CBF to enforce, (iii) making the distinction between robot controllable and uncontrollable factors in the specification (hence, the reward and constraint design) to improve the agent's environmental awareness, (iv) combining a knowledge base (set of formulas of general truth) with the task specification for high-level task satisfiability validation as well as a better structured automaton and studying the effectiveness of this integration, and (v) evaluating our approach on a complex high-dimensional robot cooking-and-serving task. We present one of the first attempts to learn from high-level specifications in continuous domains while preventing task violation during both exploration and deployment.

### Problem formulation and approach

Here, we considered the following problem: Given (i) a robotic system, (ii) a high-level rich task specification, (iii) a set of safety requirements, and (iv) general knowledge expressed in rich, comprehensible language, generate a motion plan for (i) that satisfies (ii), (iii), and (iv). As an example, we focused on an application in which two robots are required to prepare and serve a hot dog (see Fig. 1).

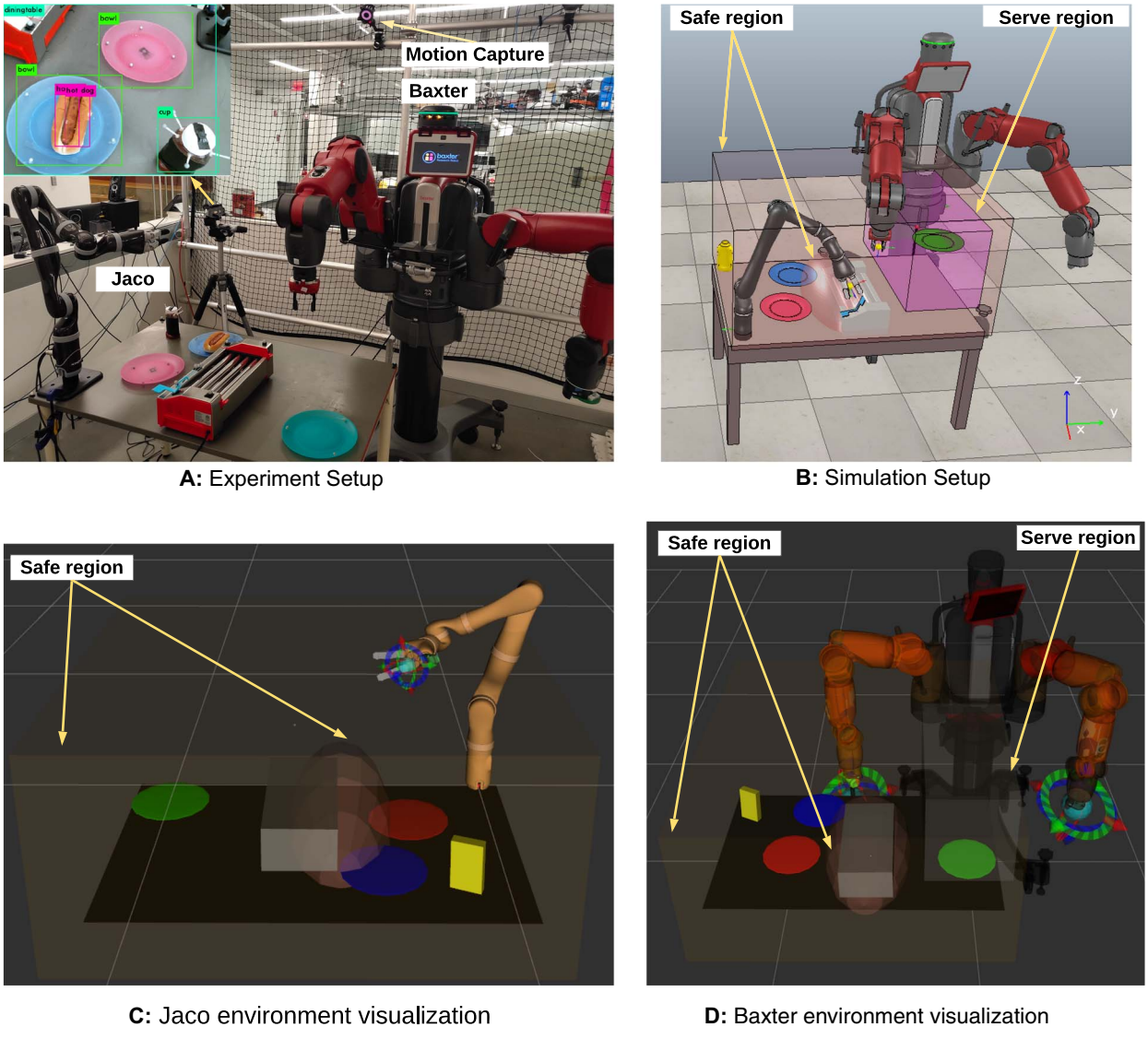
Central to our approach is a predicate temporal logic, called truncated linear temporal logic (TLTL), whose formulas provide a unifying, easily understandable way to specify and combine tasks (e.g., "pick up the hot dog and place it on the grill"), safety requirements ("always avoid collisions"), and general prior knowledge (facts that are known to always be true, e.g., "you cannot pick up an object if you are already holding one"). We converted the overall formula into an automaton called finite-state predicate automaton (FSPA) and proposed an automaton-guided method for RL that creates easily explainable reward functions. Safety constraints are also extracted from the automaton and enforced using CBFs. The resultant motion plan both satisfies the task and guarantees safety.

## RESULTS

In this section, we present a solution to the problem formulated above. We first describe the solution for the general case; then, we focus on a particular cook-and-serve application, in which two manipulators work together to grill and serve a hot dog. Technical details, implementation notes, and experimental results can be found in Materials and Methods and the Supplementary Materials.

### Task specification and general knowledge are temporal logic formulas

We introduce a common formal language to describe both prior knowledge about the world, and task specifications. This is based on an existing predicate temporal logic, called TLTL (*28*). Informally, a formula ($\phi$) in our language is made of standard logical operators, such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$); temporal operators, such as eventually ($\mathcal{F}$), until ($\mathcal{U}$), and next ($\mathcal{X}$); predicates, such as $f(s) > 0$ ($f$ is a scalar function defined on the state space of the system); and derived operators, such as finite time "always" ($\mathcal{G}\phi = \neg\mathcal{F}\neg\phi$) and "then" ($\phi_1 \mathcal{T} \phi_2 = \phi_1 \wedge \mathcal{X}\mathcal{F}\phi_2$). For example, *GripperOpen*($s$) is a




**A:** Experiment Setup

**B:** Simulation Setup

**C:** Jaco environment visualization

**D:** Baxter environment visualization

**Fig. 1. Hot dog cooking and serving task setup.** (**A**) Experiment setup consists of a seven-DOF Jaco arm and a Baxter robot. Objects in the scene are tracked by a motion capture system. The status of a ready-to-serve hot dog was detected using a camera running Darknet. (**B**) Simulation environment in the V-REP simulator setup for training. (**C** and **D**) Information perceived by each robot during experiment (displayed in RViz). Tracked items include the three plates, the grill, and the ketchup. The safe regions and serve region are defined by the task specification.

predicate used in the cook-and-serve application described later (see also table S1). "Eventually *GripperOpen*(*s*)" [$\mathcal{F}$ *GripperOpen*(*s*)] is an example of a simple formula using the same predicate.

The general knowledge base formulas correspond to properties that are always true (e.g., a robot's gripper cannot be both open and closed simultaneously). The knowledge base can contain more complicated concepts, such as obstacle avoidance, which roughly translates to "never leave some safe region." The purpose of the knowledge base is to represent common truths about the world. This allows for the knowledge base to be used across many possible task specifications. The use of TLTL to define the knowledge base allows for both interpretability of the knowledge and for the knowledge base specification set to be conjuncted with the task specification to create a single formula.

Formulas in TLTL have dual semantics, meaning that they can be either true or false or real valued. This means that we can quantify the degree of satisfaction of a formula (it has quantitative semantics). Here, we exploit these quantitative semantics to design rewards for RL.

## Automaton corresponding to TLTL formulas determines the RL rewards

With both the knowledge base and the task specification, a graph structure can be constructed to reflect the required sequence of motions to successfully complete (satisfy) the task. These structures are called FSPA, and they directly capture the syntax and semantics of TLTL through a tuple of states, transitions, guards, initial states, and final states. The guards on the FSPA transitions are predicate Boolean formulas [in disjunctive normal form (DNF)], with robustness (quantitative semantic measure) that can be evaluated at any time. The robustness of a TLTL formula is a metric describing how well the formula is satisfied given a state sequence (a positive robustness value indicates satisfaction, and a negative value indicates violation).

The RL problem is often defined on an MDP, which is a tuple composed of a state space, an action space, a reward function, and a transition function. In this case, both the reward and transition functions are unknown initially; therefore, the information from the FSPA is used to create a reward function, whereas the transition function is learned through experience. To create the reward, we use a product automaton, which combines the MDP and the FSPA, meaning that each state in the product automaton is a pairing of FSPA and MDP states. This allows us to use the FSPA guards as a reward function for the RL task, meaning that we directly generate the rewards from the high-level specification, making the reward synthesis problem fundamentally more formal, explainable, and tractable.

At each state in the product automaton, there exist edges with guards that move the system to other states. In this work, we evaluate each of the edge guards in terms of their robustness and then pick the edge whose guard has the highest robustness. The robustness of this edge guard is then the reward for the RL agent at this particular time step. As the system moves through the product automaton, the reward function is updated on the basis of the available outgoing edge from the agent's current FSPA state.

### CBFs enforce safety

Beyond providing the reward function, the FSPA is also used to ensure safe exploration and deployment of the system. Within the FSPA structure, there are trap states that have guards to explicitly denote conditions that will violate the specification. These trap states have no transition to get the system to an accepting final state; therefore, they are primarily used to denote when the system has entered a state that renders the given task impossible to complete. In this work, these guards are treated as constraints in a CBF-based approach. In the traditional CBF approach (*24*), given a control system, a safety constraint, control limitations, and a cost that involves the control, a safe optimal controller is found as a solution to a linear or quadratic program. In this work, given a control action determined by the RL procedure, we use a CBF method to find a control that, when added to the RL control, makes the system safe while minimizing the control interference.

### Baxter and Jaco cook and serve hot dogs

To test the proposed framework, we present an experiment that uses two robotic manipulators (a Kinova Jaco arm and a Rethink Robotics Baxter) to perform a hot dog cooking and serving task (the simulation and experiment setups are shown in Fig. 1). The Jaco arm was used to prepare the hot dogs, and Baxter was used to serve the hot dog to customers when it detected that one was ready. Most of the objects in the scene were tracked using an OptiTrack motion capture system. A camera was used to identify the existence of a ready-to-serve hot dog as shown in the top left corner of Fig. 1A. Figure 1B shows the simulation environment constructed in the V-REP simulator (*29*). Figure 1 (C and D) shows the environment the robots perceive during the experiment. This includes all tracked objects (three plates, the grill, and the ketchup) as well as the safe regions (the outer box that defines the end effector's outermost region of motion and an ellipsoid that defines the grill's region of collision). In addition, the Baxter robot has a serve region that is used to determine whether there is a customer (i.e., the green plate is detected within the serve region).

### Task specification and prior knowledge

The specification for cooking, preparing, and serving a hot dog in this experiment is given by a combination of knowledge from a general knowledge base and a task-specific "recipe." We begin by introducing a template formula (a specification in TLTL), which is used as a midlevel task definition that can be composed (because of the inductive property of TLTL formulas) with other templates to simplify the specification for more complicated tasks. For example, $Grasped(\boldsymbol{p}_{\text{current}}, \boldsymbol{p}_{\text{goal}})$ can be defined in English as "eventually reach the goal pose ($\boldsymbol{p}_{\text{goal}}$) and with the gripper open, and eventually close the gripper, and do not close the gripper until the goal pose is reached and with the gripper open." The TLTL formula for $Grasped(\cdot)$ can be found in the Supplementary Materials, and its FSPA can be found in fig. S1A. $Grasped(\cdot)$ can be combined with a simple knowledge base to demonstrate its effect on the FSPA. To this end, we specify an example knowledge base as "the gripper cannot be both open and closed at the same time" $[\neg(GripperOpen(\cdot) \wedge GripperClose(\cdot))]$ and "the system state must remain in the safe region" $[InSafeRegion(\cdot)]$. The task specification and knowledge base are from the same logic, TLTL, so they can easily be composed into a single specification using the conjunction operator. The FSPA from the resulting specification is shown in fig. S1B. Note two key changes from the FSPA for just the task specification alone (in fig. S1A); first, the edge from $q_0$ to $q_f$ has been trimmed because the knowledge base states that the gripper cannot be both opened and closed, and second, on each guard, the fact that the end-effector pose needs to stay in the safe region is stipulated.

The hot dog cooking task can be expressed as "eventually turn on the grill, then pick sausage and place on grill, then go to home position, then wait for 5 min, then pick sausage and place on bun, then apply ketchup, then turn off grill, then go to home position"; the hot dog serving task can be described as "eventually serve and do not serve until a hot dog and a customer are detected." The FSPAs for both tasks are shown in Fig. 2 (A and B). For the full definition of the predicates, template formulas, task specifications, and the knowledge base used in this experiment, refer to the Supplementary Materials. Refer also to Materials and Methods for details on evaluating the truth value of the edge guards and transitioning on the FSPA.

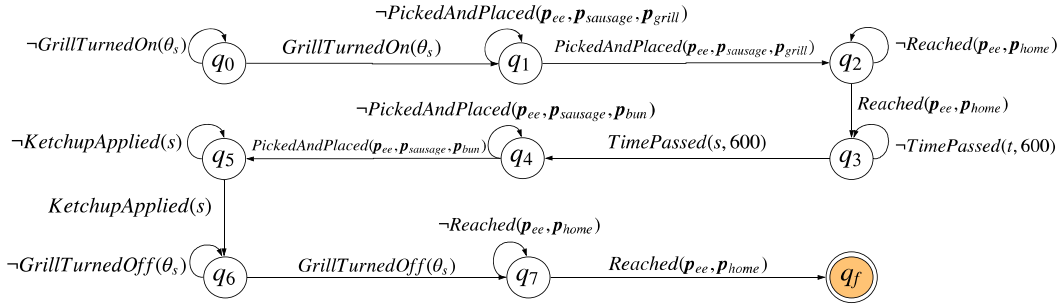
### Training in simulation and executing on the real system

All of the training is performed in the V-REP simulation environment shown in Fig. 1B, and the resultant policy is executed directly on the real robots. This is possible because the learned policy outputs a path that does not take into account the robot's configuration space or dynamics, meaning that the reality gap is minimal and simulation-to-real transfer is possible. Details on the training setup can be found in Materials and Methods.

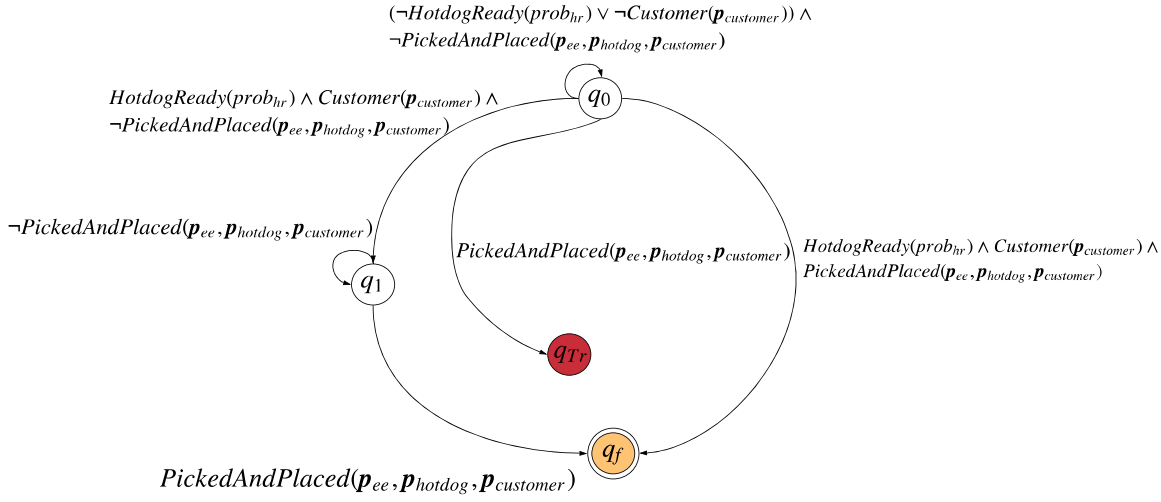
An example execution trace for the cooking and serving task can be found in Fig. 3. Each subfigure represents a change in the predicate truth value for at least one robot's FSPA (accomplishment of a subtask). The colored paths (a gray path for Jaco and a red path for Baxter) in Fig. 3 show times during the execution when a transition in the FSPA has occurred (for at least one robot). Above each path, we show the current FSPA state $q_i$, which corresponds to Fig. 2 (A and B). We also show the guards that are satisfied. Videos of learning and execution can be found in the Supplementary Materials.

### Safety evaluation during training

The system is safe at any given point in time if the value of all CBFs are greater than zero. To investigate the effectiveness of CBFs in safeguarding the learning process, we recorded the CBF values during training and plot their minimum values as a function of training time in Fig. 4 (A and B). For comparison, we also trained a policy without




A




B

**Fig. 2. FSPA for the hot dog cooking and serving tasks.** (**A**) *HotdogCooked*(*s*). (**B**) *HotdogServed*(*s*).

enforcing CBFs (only recording the CBF values). In this case, we terminated and reinitiated an episode when the agent violates a safety specification and penalized the agent with a negative terminal reward equal to the largest negative CBF value. The comparison cases are also illustrated in Fig. 4 (A and B). Because the CBF value is related to entering or leaving a safe region, we can observe that the CBFs are successful in the prevention of safety violation during training (i.e., the minimum value is always greater than zero). In the case without CBFs constraining the agent's motion, the negative terminal reward provides a signal for the RL agent to learn and avoid unsafe regions (the minimum CBF value gradually increases), but safety cannot be guaranteed throughout learning.

## Comparisons of different training setups and imitation learning baselines

Four training setups were used to study the effectiveness of the CBFs and the knowledge base on the performance of the final policy. These studies were divided into FSPA-guided RL without a knowledge base or CBFs (FSPA), FSPA-guided RL with only CBFs (FSPA+CBF), FSPA-guided RL with only knowledge base (FSPA+KB), and FSPA-guided RL with both CBFs and a knowledge base (FSPA+CBF+KB). All four scenarios were trained for the same amount of time with the same training algorithm and hyperparameters.

We also compared our method with imitation learning benchmarks, specifically with BC and generative adversarial imitation learning (GAIL) (*30*). We teleoperatively controlled each robot in simulation to collect expert demonstrations. Two hundred demonstration trajectories were collected for each robot, and the OpenAI Baseline (*31*) implementation of BC and GAIL was used to train the policies with default hyperparameters.

Each resultant policy was executed for 30 trials with a horizon of 400 time steps. The position and orientation of the plates and the ketchup were randomized (within reachable regions) between trials (the position of the grill does not change). The robustness of each collected trajectory with respect to the task specification was calculated, and the task was considered successfully completed if the robustness value was greater than 0. The average success rates are presented in Fig. 4 (C and D).

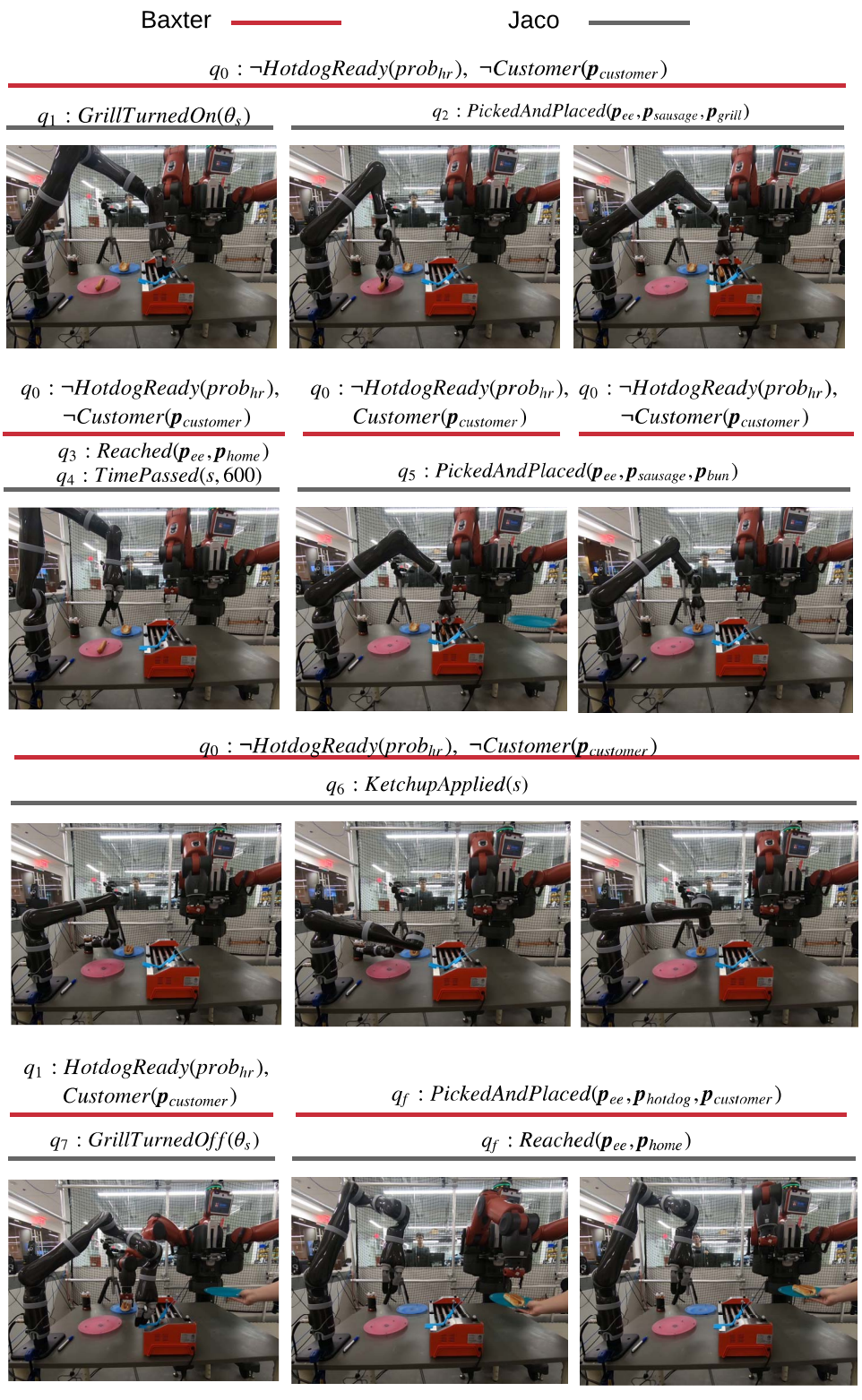



**Fig. 3. Example execution trace of the hot dog cooking and serving task.** Each snapshot represents a change in the FSPA (change of a predicate truth value and/or transition on the FSPA). The gray and red paths trace the steps necessary to trigger a transition on the FSPA (for either robot) shown in Fig. 2 (A and B). The FSPA state along with the corresponding edge guard is provided on top of each path.

Compared with the baseline FSPA-guided RL, we can observe from Fig. 4 (C and D) that the addition of the CBFs noticeably improved the performance of the learned policy. This is because by keeping the system safe during exploration, the RL agent was able to focus on learning to satisfy the specification, whereas in the case without CBF, it also had to learn to avoid unsafe regions. Given that an episode terminates when safety constraints are violated, CBF allows the RL agent to explore longer per episode, which also contributes to the final performance.

The inclusion of the knowledge base also increases the final success rates for both tasks. This improvement is more apparent in the serving task compared with the cooking task because although more steps are necessary to complete the cooking task, its FSPA structure is relatively simple (a linear sequence) when compared with the serving task. In the cooking task case, the knowledge base also does not help much in simplifying the reward structure. The serving task, on the other hand, consists of choices given conditions, which translates to more branching in the FSPA, meaning that the knowledge base helps reduce the complexity of the reward by pruning infeasible edges.

Neither BC nor GAIL was able to succeed, given the number of demonstrations and the evaluation criteria for the hot dog cooking task. In our experiments, we noticed that the trajectories produced by GAIL have higher similarity to the expert trajectories than those of BC. However, low tracking accuracy with respect to the given goals prevents the robustness of these trajectories from evaluating to a positive number. GAIL exhibited a slightly higher success rate in the serving task, whereas BC continued to fail. This set of results show that for tasks with more complex structure and longer horizons, using pure imitation learning without extensive exploration and experience to learn a useful policy can be challenging.

### Evaluation of the learning progress

During training, the FSPA states were randomized between episodes. In doing so, the RL agent was able to experience and learn at different stages of the task without having to follow the order that the task is carried out. This can effectively be seen as a form of curriculum learning

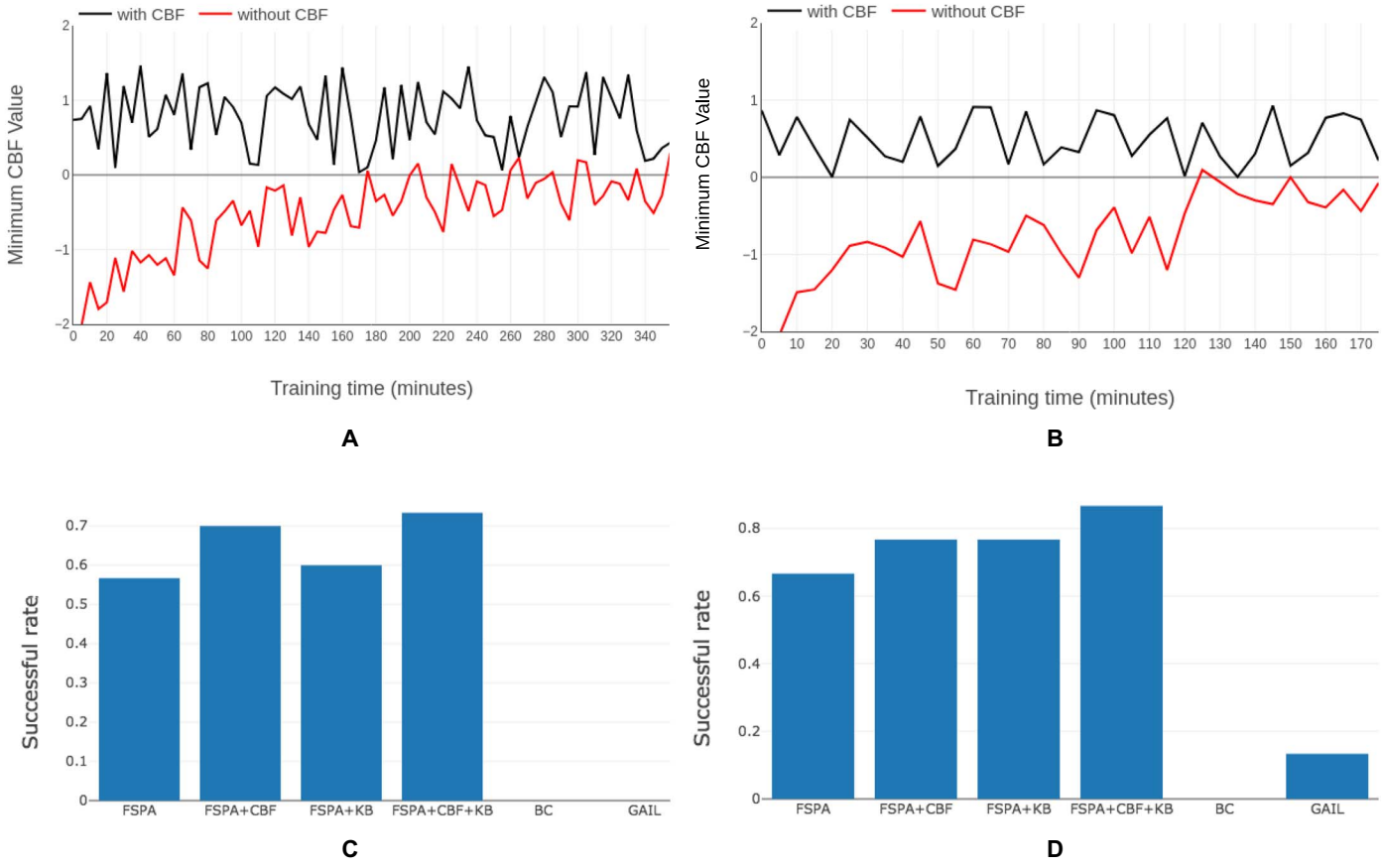



**Fig. 4. Performance evaluation and comparison.** (**A**) The minimum CBF value against training time for Jaco. At each time, the system is safe if the minimum of the CBF values is greater than zero. The red curves present the case where CBFs are not enforced during training but their values are recorded for comparison. In this case, an episode ends when the minimum CBF value is smaller than zero and this value is given to the RL agent as a penalizing terminal reward. (**B**) The minimum CBF value against training time for Baxter. (**C**) Performance comparison of the final policies for Jaco. Here, FSPA represents vanilla FSPA-guided RL. Each policy is executed to produce 30 trajectories. A trajectory satisfies the task (success) if its robustness evaluated against its task specification is greater than zero. Average success rate is reported. (**D**) Performance comparison of the final policies for Baxter.

where the FSPA decomposes a complex task into simpler subtasks. Because of this nature, although the RL policy is trained end to end, we were able to study the internals of the policy by probing different FSPA states during the learning and evaluation process.

At fixed time intervals during training, we took the latest policy and studied its performance at each of the FSPA states. We initialized the policy at each FSPA state and executed it for a certain number of time steps (100 steps for Baxter and 200 steps for Jaco). Thirty evaluation trajectories were obtained at each FSPA state with the plates and ketchup poses randomized between trials. The goal was to obtain the average success rate of the RL agent at transitioning out of the current FSPA state and the average time it takes to do so. Success here was evaluated by calculating the robustness of each trajectory against the disjunction of the outgoing edge guards (a predicate Boolean formula) and determining whether its value is positive. In general, a positive robustness with a low transition time indicates a capable policy at that specific FSPA state. We present the results of this probing as a labeled heat map in Fig. 5 (A and B). In this figure, the $x$ axis shows training time, and the $y$ axis is the FSPA state corresponding to Fig. 2 (A and B). The color of each cell indicates the average number of steps required for the agent to transition out of the current FSPA state. The cell labels show the average success rate over the 30 evaluation trajectories. One exception is state $q_0$ in Fig. 5A because this is the starting state of Baxter's serving task. Transition out of this initial state does not depend on Baxter's actions but on whether there is simultaneously a customer and a ready-to-serve hot dog. Baxter's desired motion at this state is to stay put (minimize control effort). Therefore, for this state, we neglect the meaning of the cell color. The cell label indicates the percentage of trajectories where the maximum control effort is below a certain threshold.

From Fig. 5A, we can observe that learning to stay put ($q_0$) is relatively simple, whereas more effort is required to learn the pick-and-place motion ($q_1$). It also shows that an efficient pick-and-place policy that generalizes across different target poses can be learned. Looking at Fig. 5B, we can see that a *Reached* task ($q_2$ and $q_7$) is the easiest to learn, followed by the *PickedAndPlaced* task ($q_1$ and $q_4$), which consists of a sequence of *Reached*, *GripperOpen*, and *GripperClose* tasks. It takes longer to make progress in learning the *GrillTurnedOn* ($q_0$) and *GrillTurnedOff* ($q_6$) tasks. Compared with the *Reached* and *PickedAndPlaced* tasks where the reward is a distance function, *GrillTurnedOn* and *GrillTurnedOff* provide reward only when the grill switch is flipped, which is a much sparser reward. However, the policy improves quickly once it has learned where the switch is because the position of the grill does not change. The *KetchupApplied* task ($q_5$) takes the most amount of time to complete because it involves

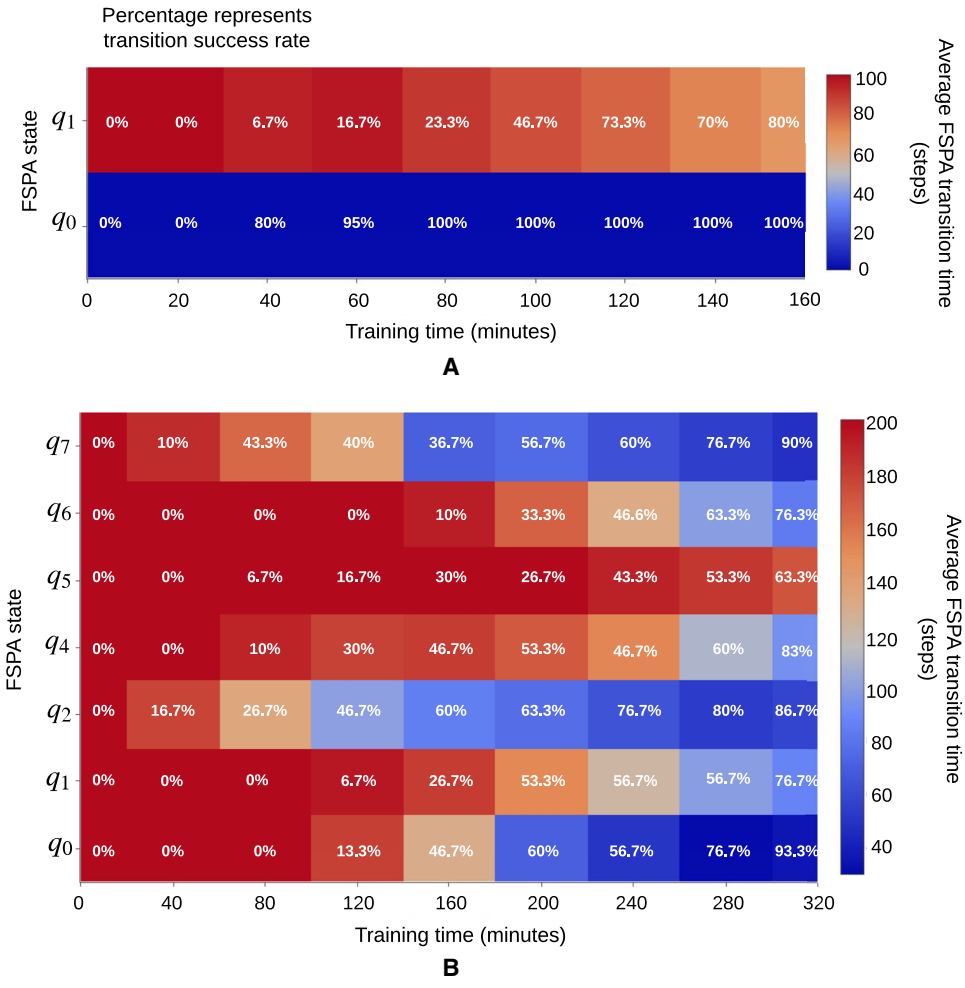



**Fig. 5. Learning progress evaluation at each FSPA state.** (**A**) Baxter. (**B**) Jaco. The *y* axis shows the FSPA states corresponding to Fig. 2 (A and B). For a fixed time interval during training, the latest policy is evaluated for its ability to transition out of the current FSPA state. This is done by setting the FSPA state and executing the policy for a number of time steps (100 for Baxter and 200 for Jaco). Thirty trajectories are collected for each evaluation. The cell color corresponds to the average time for the transition to happen, and the cell labels show the success rates. $q_3$ in (B) is omitted because it corresponds to *TimePassed*(*s*,600), which is not learned.

the most number of substeps, as shown in fig. S1E. The final success rate for the *KetchupApplied* task is also lower compared with the other tasks, which is a result of the large position and orientation changes the end effector needs to undergo to accomplish the task.

## DISCUSSION

Experimental results suggest that the reward generated from the formal specification is able to guide an RL agent to learn a satisfying policy. Safety during exploration can be guaranteed (given that safety is included in the specification). Our method also promotes the general usability of RL frameworks by providing a formal language–based user interface. Specifically, the user only needs to interact with the framework using a rich and high-level language, and the process of finding a satisfying policy can be automated.

The efficacy of the proposed framework results from (i) dense reward feedback, (ii) using the FSPA to break the task into subtasks and being able to learn at different stages of the task by creating a curriculum over the FSPA states, (iii) using the knowledge to make the FSPA more context aware (prune edges that are infeasible, etc.), and (iv) using the CBF to prevent violation to simplify the task for the RL agent and elongate the exploration process (assuming an episode is terminated upon task violation if CBF is not used).

### Advantages of using temporal logic for task and knowledge specification

Our method provides a means of conveniently incorporating domain knowledge in a structured and interpretable fashion. This can be a valuable tool for boosting learning sample efficiency, especially as the task becomes more complex. Flexibility is another merit that our method brings. Learning can be time and resource demanding. Given a knowledge base of general truth, our framework promotes rapid validation of the task against it without having to undergo training. The user can quickly adjust the task if it is proven unsatisfiable. This somewhat resembles the abstract reasoning that humans routinely conduct.

In the domain of deep RL, analyzing the neural network policy can be a challenge. Tools have been developed to verify properties of neural networks, but scalability and efficiency remain areas where more research is necessary. In our framework, because the policy is trained with the FSPA-induced reward and the FSPA state is part of the policy's input, we can easily analyze the policy for its performance on different stages of the task. As shown in Fig. 5, we can probe the policy to observe how learning has progressed at various stages of the task, and because the reward structure is interpretable, we can easily deduce the reason for any bottleneck during training (e.g., learning to flip the switch is slow because of sparse reward). Having found the bottleneck, we can then focus on training certain stages of the task by adjusting the initialization probability of the FSPA states at the start of each episode.

### Benefits of using CBF to prevent violation of specification

Safety during exploration and deployment is paramount to the applicability of learning methods on physical systems. Our results show that learning performance can also benefit from safe exploration even in simulation where collision results in little consequence. This is because guaranteeing safety through external means simplifies the task that the RL agent has to learn while also increasing the length of exploration per episode.

In our formulations, the CBF constraints are generated from the FSPA. This means that the CBF is used not only to guarantee collision

safety but also, more generally, to avoid violation of the task specification. The CBF constraints change with the progression of the task. For example, if given a task of "visit regions A, B, and C in this order and do not revisit regions," then at the beginning of the task, CBF will prevent visitation of regions B and C. After region A has been visited, CBF will then prevent visitation of A and C, etc. In general, RL takes care of learning to reach the acceptance FSPA state, whereas CBF is responsible for prevention of reaching the trap state.

### Limitations and future work

The provided examples generate a path plan in Euclidean space without considering the configuration space of the real system. As a result, the performance of the robot depends largely on how close it is able to track the given path. The method itself, however, can handle higher-dimensional configuration space planning. Currently, we assume linear dynamics, which simplifies the derivation of the CBF constraints. However, one of the strengths of the CBF is its ability to incorporate general nonlinear affine dynamics. One direction of future work is to develop a motion plan variant of the proposed method that takes into account the robot kinematics/dynamics and learns a policy that directly outputs joint level controls. The effectiveness of our method may also be limited by FSPAs with cycles (loops between automaton states that are not self-loops). This issue can be resolved by modifying our current (greedy) reward design to be potential-based rewards (*11*, *32*).

In our formulation, although we specify the task hierarchically using template formulas, the resultant FSPA is nonhierarchical. The sizes of the FSPAs in our tasks are manageable (22 nodes and 43 edges for the cooking task and 8 nodes and 22 edges for the serving task); in general, the size of an FSPA can grow rapidly with the complexity of the TL formula (and knowledge bases can be large), which, in turn, increases the complexity of the reward. This can adversely affect learning. One approach is to maintain multiple simpler FSPAs (for example, one for each template formula) instead of a complex one. This approach adds discrete dimensions to the state space (one for each FSPA) but can notably reduce the complexity of each FSPA. Although not fully developed in this work, we believe that the incorporation of the knowledge base and template formulas present opportunities to extend our framework to a wider set of capabilities, such as high-level (symbolic) task planning/validation, hierarchical learning, and skill composition.

## MATERIALS AND METHODS

### Preliminaries

#### *MDP and RL*

RL is an area of machine learning that focuses on finding an optimal policy that maximizes a cumulative reward by interacting with the environment. We start by defining an MDP (*33*).

Definition 1. An MDP is defined as a tuple $\mathcal{M} = \langle S, A, pr, r\rangle$, where $S \subseteq \mathbb{R}^n$ is the state space; $A \subseteq \mathbb{R}^m$ is the action space ($S$ and $A$ can also be discrete sets); $pr : S \times S \times A \to [0,1]$ is the transition function, with $pr(s_{t+1} \mid s_t, a_t)$ being the conditional probability of being in state $s_{t+1} \in S$ after taking action $a_t \in A$ at state $s_t \in S$; and the reward function is $r : S \times A \times S \to \mathbb{R}$, with $r(s_t, a_t, s_{t+1})$ being the reward obtained by executing action $a_t$ at state $s_t$ and transitioning to $s_{t+1}$.

We assume that the underlying time is discrete: $t = 0,1,2\ldots$. We use $s_t$ and $a_t$ to denote the state and action at time $t$, respectively. An optimal policy $\pi^\star : S \to A$ (or $\pi^\star : S \times A \to [0,1]$ for stochastic policies) is one that maximizes the expected return, that is

$$\pi^\star = \arg\max_\pi \left( E^\pi \left[ \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \right] \right) \tag{1}$$

The horizon (denoted $T$) is defined as the maximum allowable number of time steps of each execution of $\pi$ and, hence, the maximum length of a trajectory. In Eq. 1, $E^\pi[\cdot]$ is the expectation following $\pi$. The state-action value function is defined as

$$Q^\pi(s,a) = E^\pi \left[ \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1}) \mid s_0 = s, a_0 = a \right] \tag{2}$$

which is the expected return of choosing action $a$ at state $s$ and following $\pi$ onward. $Q^\pi$ is commonly used to evaluate the quality of policy $\pi$. For problems with continuous state and action spaces such as robotic control, $Q^\pi$ and $\pi$ usually take the form of parameterized function approximators. Depending on the RL method, one can optimize for $Q^\pi$ and take greedy actions accordingly [such as Deep Q-Network (DQN) (*34*)]. One can also use Eq. 1 for direct policy search [such as policy gradient methods (*35*)]. Another popular approach is to alternate between optimizing $Q^\pi$ and $\pi$, which is referred to as actor-critic methods [such as (*36*)]. In this work, we will use an actor-critic method–proximal policy gradient (*37*) as our RL algorithm.

#### *Truncated linear temporal logic*

We consider tasks specified using TLTL (*28*). Formal definitions for its syntax and semantics are given in the Supplementary Materials. Informally, TLTL is made of predicates [e.g., $f(s) > 0$, where $f : \mathbb{R}^n \to \mathbb{R}$ is a scalar function]; the usual Boolean operators, such as $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), etc.; and temporal operators, such as $\mathcal{X}$ (next), $\mathcal{F}$ (eventually or in the future), $\mathcal{G}$ (bounded always, globally over a finite interval), and $\mathcal{U}$ (until). An example of a valid TLTL formula is (we assume $s \in \mathbb{R}$) $\mathcal{G}(s < 8) \wedge \mathcal{F}(s < 4) \wedge \mathcal{X}\,\mathcal{F}(s > 6)$, which entails that for all times in the considered interval, $s < 8$, and at some point in the future, $s < 4$, and after that, $s > 6$. Another example is $(\phi_a \Rightarrow \mathcal{F}\phi_b)\mathcal{U}\phi_c$, which means that $\phi_c$ becomes eventually true, and until (before) this happens, if $\phi_a$ is satisfied, then $\phi_b$ is eventually satisfied ($\phi_{a,b,c}$ are valid formulas).

The semantics of TLTL formulas is given over finite trajectories in a set $S$, such as the ones produced by the MDP from Definition 1. We use $s_{t:t+k}$ to denote a sequence of states (state trajectory) from time $t$ to $t + k$, i.e., $s_{t:t+k} = s_t s_{t+1} \ldots s_{t+k}$. TLTL has both qualitative (a trajectory either satisfies or violates a formula) and quantitative semantics (details are given in the Supplementary Materials). The latter quantifies the degree of satisfaction of a formula $\phi$ by a trajectory $s_{0:T}$. This measure is also referred to as robustness degree or, simply, robustness [$\rho(s_{0:T}, \phi)$ maps a state trajectory and a formula to a real number], for example (again, we assume $s \in \mathbb{R}$), $\rho(s_{0:3}, \mathcal{F}(s < 4)) = \max(4 - s_0, 4 - s_1, 4 - s_2)$. Because $\mathcal{F}(s < 4)$ requires $s < 4$ to be true at least once in the trajectory, we take the maximum over the time horizon. In general, robustness greater than zero (in the quantitative semantics) is equivalent with $s_{t:t+k}$ satisfying $\phi$ (in the qualitative semantics).

The main differences between TLTL and linear temporal logic (*38*) are as follows: (i) TLTL is evaluated against finite traces (hence, the

term "truncated"), whereas linear temporal logic is over infinite traces. (ii) TLTL is specified over predicates of MDP states compared with atomic propositions in linear temporal logic.

### *Control barrier functions*

Consider a discrete time affine control system

$$s_{t+1} = f(s_t) + g(s_t)a_t \tag{3}$$

where $f: S \to S$ and $g: S \to \mathbb{R}^{n\times m}$ are locally Lipschitz continuous, $s_t \in S \subseteq \mathbb{R}^n$ is the state at time $t$, and $a_t \in A \subseteq \mathbb{R}^m$ is the control input (action) at time $t$. We use the same notation for the states and actions of the control system from Eq. 3 and the MDP from Definition 1. We embed the control systems generating the desired robot trajectories and environmental dynamics into the MDP.

Consider the following set

$$\mathcal{C} = \{s \in S \mid h(s) \geq 0\} \tag{4}$$

where $h : S \to \mathbb{R}$.

A function $h : S \to \mathbb{R}$ is a discrete time exponential CBF (*39*, *40*) that renders $\mathcal{C}$ invariant along the trajectories of the system in Eq. 3 if there exist $\alpha \in [0,1]$ and $a_t \in A$ such that

$$h(s_0) > 0 \tag{5}$$

$$h(s_{t+1}) + (\alpha - 1)h(s_t) > 0, \quad \forall t \in \mathrm{Z}^+ \tag{6}$$

A controller that renders the safe set $\mathcal{C}$ invariant and minimizes the control effort can be found by solving the following optimization problem

$$a_t^\star = \underset{a_t \in A}{\operatorname{argmin}} \|a_t\|^2$$
$$\text{s.t.} \quad h(f(s_t) + g(s_t)a_t) + (\alpha - 1)h(s_t) \geq 0$$

$$a^{min} \leq a_t \leq a^{max} \tag{7}$$

where $a^{min}$ and $a^{max}$ are control bounds.

If $h(s)$ is affine, then the above problem is a quadratic program. If $h(s)$ is quadratic, then the problem is a quadratically constrained quadratic program. Both can be solved efficiently with available solvers such as Gurobi (*41*).

## Formal problem formulation

We define a knowledge base as a set of predicate Boolean formulas over MDP states that are always true within the horizon of the task. Formally, a knowledge base $K$ is defined as

$$K = \{\psi_0, \psi_1, \psi_2, \ldots, \psi_k\} \tag{8}$$

where $\psi_i$, $i = 0, \ldots, k$, are predicate Boolean formulas over the state $s$ of an MDP. We assume that $\forall s_{0:T}$, $\rho(\phi_K, s_{0:T}) > 0$, where $\phi_K = \mathcal{G} \bigwedge_{i=0}^{k} \psi_i$. A knowledge base is integrated into the problem formulation by adding formula $\phi_K$ to the task specification through conjunction.

In this work, a robotic system consists of robot-related states (end-effector pose, gripper position, etc.) and actions (end-effector velocity) as well as environmental states that the robot can perceive (pose of the grill, ketchup, etc.). We are now ready to formulate the general problem that we consider here.

Given a robotic system with states $S \in \mathbb{R}^n$ and actions $A \in \mathbb{R}^m$, a knowledge base $K = \{\psi_0, \psi_1, \psi_2, \ldots, \psi_n\}$, and a TLTL task specification $\phi_{task}$ over $S$, generate a trajectory $s_{0:T}$ that satisfies $\phi_{task} \wedge \phi_K$.

## Approach

An overview of our architecture is presented in Fig. 6. Given a knowledge base $\phi_K$ and a task specification $\phi_{task}$, a feasibility check is first performed to ensure that the task does not conflict with the knowledge base [using packages such as scheck (*42*) and lomap (*43*)]. If the task is feasible, then the specification $\phi_{task} \wedge \phi_K$ is transformed into an FSPA. We then extract information from the FSPA to guide an RL agent toward finding a satisfactory policy. To ensure safety during exploration and deployment, we integrated a CBF into the system that takes constraints from the FSPA and calculates a minimal interfering action that guarantees safety of the system with respect to the task specification. We will use a running example throughout the following sections. As depicted in Fig. 7A, a robot is navigating in two-dimensional (2D) space. The states are its 2D position coordinates, and its actions are $x$, $y$ velocities. There are three regions $A$, $B$, and $C$ and an obstacle $D$. The goal is to find a policy that satisfies the specification $\phi_{ex} = (\mathcal{F}\psi_A \vee \mathcal{F}\psi_B) \wedge \mathcal{F}\psi_C \wedge (\neg\psi_C \,\mathcal{U}(\psi_A \vee \psi_B)) \wedge \mathcal{G}\neg\psi_D$, which indicates "eventually visit regions $A$ or $B$ and eventually visit region $C$, and do not visit region $C$ before $A$ or $B$ has been visited, and always avoid region $D$."

### *Finite-state predicate automaton*

The FSPA is defined as

Definition 2. An FSPA is a tuple $\mathcal{A} = \langle \mathcal{Q}, S, \mathcal{E}, \Psi, q_0, b, F, Tr \rangle$, where $\mathcal{Q}$ is a finite set of automaton states; $S \subseteq \mathbb{R}^n$ is the MDP state; $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{Q}$ is the set of transitions (edges); $\Psi$ is the input alphabet, also called set of guards (a set of predicate Boolean formulas, where the predicates are evaluated over $S$); $q_0 \in \mathcal{Q}$ is the initial state; $b : \mathcal{E} \to \Psi$ maps the transitions of $\mathcal{A}$ to predicate Boolean formulas from $\Psi$; $F \subseteq \mathcal{Q}$ is the set of final (accepting) states; and $Tr \subset \mathcal{Q}$ is a set of trap states.

Although $b(q, q')$ is a Boolean formula, it can be seen as a particular case of a TLTL formula. Its robustness $\rho(s_{t:t+k}, b(q, q'))$ is only evaluated at $s_t$. Therefore, we use the shorthand $\rho(s_t, b(q, q')) = \rho(s_{t:t+k}, b(q, q'))$. At some states $q \in \mathcal{Q}$, there is a transition $(q, q')$ to a trap state $q' \in Tr$. The guard of $(q, q')$ captures all the possible situations that lead to a violation of the specification. As will become clear later, we will use CBFs to make sure transitions to trap states are not possible.

The semantics of an FSPA are defined over finite trajectories $s_{0:T}$ over $S$ (such as the ones produced by the MDP from Definition 1). The motivation for the semantics defined below will become clear later when we give the definition of the FSPA-augmented MDP. At time 0, the automaton is in state $q_0$. The automaton evaluates the formulas assigned to the edges that are connected to $q_0$ [i.e., $b(q_0, q)$, with $(q_0, q) \in \mathcal{E}$] at $s_0$ by calculating $\rho(s_0, b(q_0, q))$. If at least one of the edges has formulas that are satisfied, then the automaton will make a transition. If more than one edge have satisfied formulas, the automaton will choose the edge $(q_0, q)$ that has the largest (positive) robustness $\rho(s_0, b(q_0, q))$, which signifies best satisfaction at that time instant. Given that any nondeterministic finite automaton can be translated into its deterministic counterpart at the cost of a larger automaton, in practice, we prefer to work with a deterministic FSPA that prevents two edge guards to be satisfied at the same time. In the (unlikely) event that there are more than two outgoing edges with exactly the same maximum robustness, the automaton will choose one edge randomly. When the automaton moves to a state $q$, then the process described above is reiterated at $q$. A trajectory is accepted if it

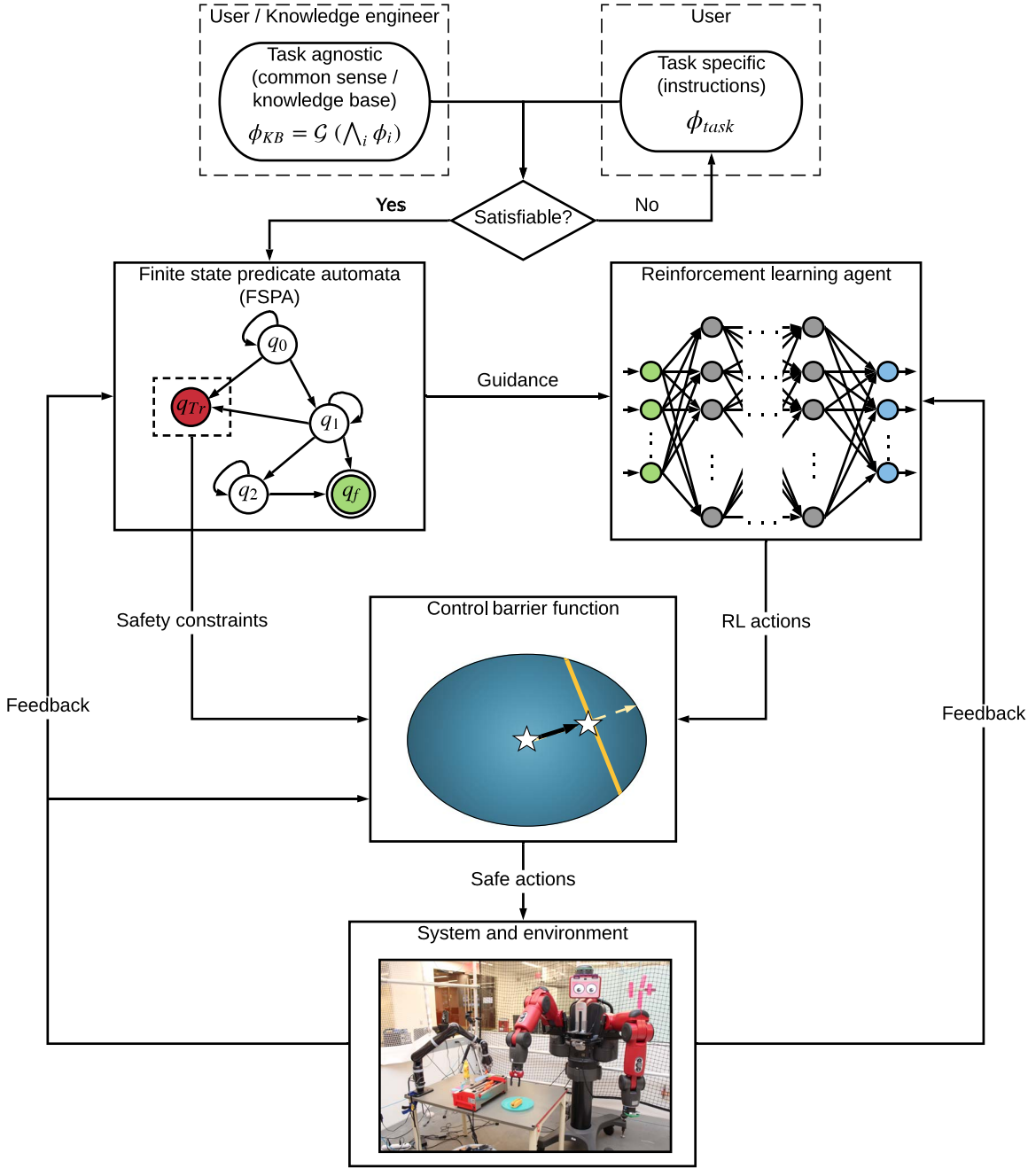



**Fig. 6. Pictorial representation of the overall approach.** The user-provided task specification is first checked symbolically against a knowledge base with generally true statements for task feasibility [using a model checking tool, such as lomap (*43*)]. If feasible, specification $\phi_{task} \wedge \phi_K$ is transformed into an FSPA that provides guidance (reward) to an RL agent that encourages satisfaction of the specification. The FSPA also provides constraints to the FSPA-guided CBF, which safeguards the decisions made by the RL agent from violating the specification while minimally interfering with the RL agent.

ends in a final state of $\mathcal{A}$. An example of transitioning on an FSPA for $\phi_{ex}$ is depicted in Fig. 7 (A to C).

The FSPA resembles a traditional FSA. The main differences are as follows: (i) predicates replace symbols on the edge guards, and (ii) transition on the FSPA depends on the robustness of the predicates at each MDP state instead of truth values of the symbols in FSAs. The decision to use TLTL and FSPA is because we need a formalism to specify finite-horizon, time-dependent (non-Markovian) robotic manipulation tasks over continuous state and action spaces and be able to define dense rewards and constraints on this formalism. Linear temporal logic with FSA does not provide such capabilities. STL (*44*) can be another alternative. However, STL does not have an automaton counterpart, and a reward function defined on STL is non-Markovian (limits the type of RL algorithms that can be used) and sparse (only a terminal reward can be obtained because robustness of an STL formula requires the entire trajectory to calculate).

Given a TLTL formula $\phi$ over predicates in a state set $S$, there exists an FSPA $\mathcal{A}_\phi = \langle \mathcal{Q}_\phi, S, \mathcal{E}_\phi, \Psi_\phi, q_{\phi,0}, b_\phi, F_\phi, Tr_\phi \rangle$ that accepts all the

trajectories over predicates in $S$ that satisfy $\phi$. This can be generated with available packages like lomap (*43*) [refer to (*45*) for details on the generation procedure]. In the following sections, we will drop the subscript $\phi$ for clarity when the context is clear.

***FSPA-augmented MDP***

The FSPA-augmented MDP $\mathcal{M}_\mathcal{A}$ establishes a connection between the FSPA (hence, a TLTL formula) and the standard RL problem (arrow labeled "Guidance" in Fig. 6). A policy learned using $\mathcal{M}_\mathcal{A}$ has implicit knowledge of the FSPA through the automaton state $q \in \mathcal{Q}$.

We define $\mathcal{P}$ to be a set of predicates with each of its elements $p \in \mathcal{P}$ in the form $f(s_t) > 0$ [$f: S \to \mathbb{R}$ is referred to as the predicate function; we assume that $f(s_t)$ is bounded and refer to it as the predicate function of $p$]. We classify the predicates into two categories: actionable and nonactionable. An actionable predicate is one such that the agent can execute actions to increase its robustness, i.e., $\exists a_t \in A$ s.t. $\rho(s_{t+1}, p) > \rho(s_t, p)$ [an example of an actionable predicate is *Grasped*$(\cdot)$]. A nonactionable predicate is one such that the agent cannot actively effect its robustness, essentially predicates associated with the environment [i.e., *Customer*$(\cdot)$, whether there is a customer]. Define $L: \mathcal{P} \to \{0, 1\}$ to be a labeling function. A predicate is actionable if $L(p) = 1$ and nonactionable if $L(p) = 0$.

Definition 3. Given FSPA $\mathcal{A} = \langle \mathcal{Q}, S, \mathcal{E}, \Psi, q_0, b, F, Tr \rangle$ and MDP $\mathcal{M} = \langle S, A, pr, r \rangle$, an FSPA-augmented MDP is defined as $\mathcal{M}_\mathcal{A} = \langle \tilde{S}, A, \tilde{pr}, \tilde{r}, \mathcal{E}, \Psi, q_0, b, F, Tr \rangle$, where $\tilde{S} \subseteq S \times \mathcal{Q}$ is the product state space. $\tilde{pr}(\tilde{s}_{t+1} \mid \tilde{s}_t, a_t)$ is the transition function defined by

$$\tilde{pr}(\tilde{s}_{t+1} \mid \tilde{s}_t, a_t) = \begin{cases} (1/C) pr(s_{t+1} \mid s_t, a_t) 1(\rho(s_t, b(q_t, q_{t+1}))) & (q_t, q_{t+1}) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where $C$ is a normalization constant and $1: \mathbb{R} \to \{0,1\}$ is the indicator function that returns value 1 if its input is greater than zero and returns 0 otherwise. Define $\Omega_{q_t} = \{q_t' \mid (q_t, q_t') \in \mathcal{E}, q_t' \neq q_t, q_t' \notin Tr\}$ to be the set of non-trap FSPA states that are connected with $q_t$ and not equal to $q_t$. Let $D(q_t) = \vee_{q_t' \in \Omega_{q_t}} b(q_t, q_t')$ represent the disjunction of all edge guards for edges $(q_t, q_t') \in \mathcal{E}, q_t' \in \Omega_{q_t}$. We can write $D(q_t)$ in its DNF

$$D(q_t) = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m_i} (\neg) p_{q_t}^{ij} \quad (10)$$

where $(\neg)$ represents possible negations in the DNF. $p_{q_t}^{ij}$ is the indexed predicate of $D(q_t)$ in its DNF form. The reward function $\tilde{r}: \tilde{S} \times \tilde{S} \to IR$ is defined as

$$\tilde{r}(\tilde{s}_t, \tilde{s}_{t+1}) = \begin{cases} \max\limits_{i \in [0,\ldots,n)} \left[ \min\limits_{j \in [0,\ldots,m_i)} (L(p_{q_t}^{ij}) \rho(s_{t+1}, (\neg) p_{q_t}^{ij})) \right] & \exists i, j \text{ s.t.} L(p_{q_t}^{ij}) = 1 \\ -\|a_t\| & \text{otherwise} \end{cases} \quad (11)$$

Note that in Eq. 11, $s_{t+1}$ is used to calculate $\tilde{r}$, which incorporates the consequence of applying action $a_t$ at state $s_t$. Action $a_t$ does not directly appear in the reward definition. We have found, in practice, that minimizing control effort can improve the performance of the learned policy, but we do not explicitly use it here.

Definition of the reward in Eq. 11 follows the robustness definition for $D(q_t)$ [$\min(\cdot)$ corresponds to the robustness of the inner conjunctive clause and $\max(\cdot)$ corresponds to the robustness of the outer disjunctive clause] while filtering out nonactionable predicates. Intuitively, the reward function encourages the system to exit the current automaton state and move on to the next non-trap state [by maximizing the robustness of the most probable outgoing edge guard at each state, hence the satisfaction of $D(q_t)$] and, by doing so, eventually reaches the final state $q_f$, which satisfies the TL specification (property of FSPA). The labeling function $L(\cdot)$ is used to filter out the actionable predicates in the edge guards. At a particular state, if the transition on the FSPA depends only on environmental factors (nonactionable predicates), then the agent is encouraged to stay put by minimizing its actions.

For the example shown in Fig. 7 (A to C), $D(q_0) = (\psi_A \wedge \neg \psi_D) \vee (\psi_B \wedge \neg \psi_D)$, $D(q_1) = \psi_C \wedge \neg \psi_D$. At $q_0$, the reward in Eq. 11 guides the robot to region $A$ or $B$, whichever is closer (with higher robustness). At $q_1$, the reward encourages the robot to visit region $C$.

***FSPA-guided safety constraint generation***

The aim of this section is to develop a method that extracts safe sets from the FSPA (arrow labeled "Safety constraints" in Fig. 6). At state $(s_t, q_t)$, if $(q_t, q_{Tr}) \in \mathcal{E}$, then we would like to avoid activating $b(q_t, q_{Tr})$. That is, we need to ensure that $b(q_t, q_{Tr})$ is always false or, in other words, is always true $[\rho(s_t, \neg b(q_t, q_{Tr})) > 0]$. Using ideas similar to that in the reward definition, we first write $b(q_t, q_{Tr})$ in its DNF form

$$b(q_t, q_{Tr}) = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m_i} (\neg) p_{q_t}^{ij} \quad (12)$$

Let

$$i^\star(s_t, q_t) = \arg\max_{i \in \{0,\ldots,n\}} \rho(s_t, \bigwedge_{j=0}^{m_i} (\neg) p_{q_t}^{ij}) \quad (13)$$

We define the FSPA-based safe set at state $q_t$ as

$$\mathcal{C}(q_t) = \{s_t \mid h_i(s_t, q_t) \rangle 0, h_i \in H(s_t, q_t)\}, \text{where}$$

$$H(s_t, q_t) = \begin{cases} \left\{ -\rho\left(s_t, (\neg) p_{q_t}^{i^\star j}\right) \mid j \in \{0, \ldots, m_{i^\star}\}, L\left(p_{q_t}^{i^\star j}\right) = 1 \right\} & (q_t, q_{Tr}) \in \mathcal{E} \\ \varnothing & (q_t, q_{Tr}) \notin \mathcal{E} \end{cases} \quad (14)$$

In the above equation, $i^\star$ is used as a shorthand for $i^\star(s_t, q_t)$. At each time step, if there is an edge connecting $q_t$ to $q_{Tr}$, then Eq. 13 finds the conjunctive clause in Eq. 12 with the highest robustness (hence, most likely to be satisfied). Equation 14 then sets the negative robustness (negation) of the actionable predicates of conjunctive clause $i^\star$ as the CBFs. In the event that there are more than one conjunctive clause with the same (highest) robustness, then all their predicates are used to define $H$.

It is also valid to assign the negative robustness of all $(\neg) p_{q_t}^{ij}$ in Eq. 12 as CBFs. Doing so increases the number of constraints and may be too restrictive at times; however, it prevents violation in a global sense.

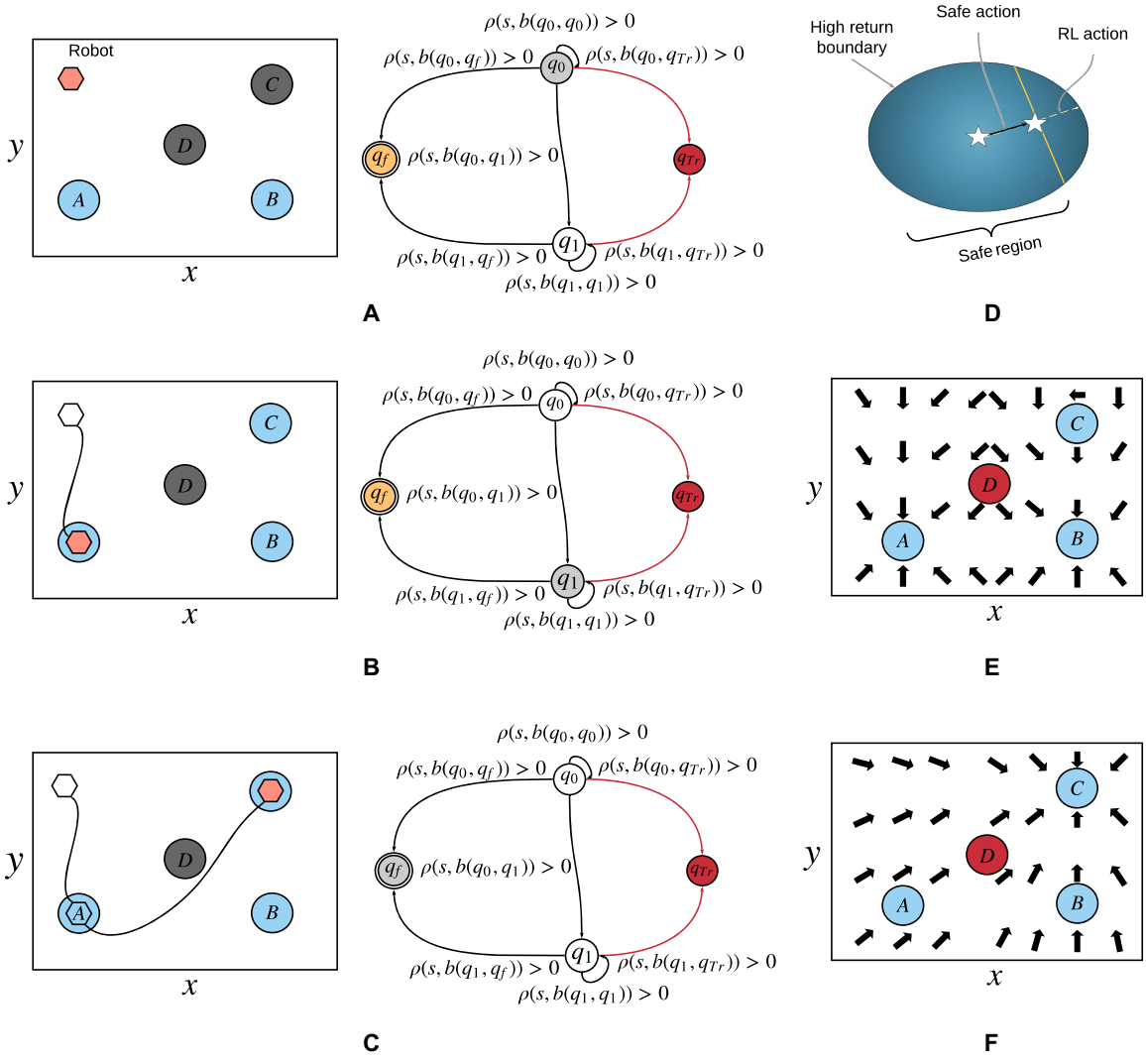



**Fig. 7. A robot navigation example to illustrate the use of the FSPA-augmented MDP and FSPA-guided CBF.** The robot moves in a 2D world with regions *A*, *B*, and *C* and obstacle *D*. The robot's states are its positions denoted $s = \boldsymbol{p}_{robot}$. The robot is to satisfy the task $\phi_{ex} = (\mathcal{F}\ \psi_A \vee \mathcal{F}\ \psi_B) \wedge \mathcal{F}\ \psi_C \wedge (\neg\psi_C\ \mathcal{U}\ (\psi_A \vee \psi_B)) \wedge \mathcal{G}\neg\psi_D$ where $\psi_i = \|\boldsymbol{p}_{\text{robot}} - \boldsymbol{p}_i\| < \delta$, $i \in \{A, B, C, D\}$, and $\boldsymbol{p}_i$ is the center coordinate of region *i*. (**A**) The robot's initial position and automaton state, $q_0$ (shown in light gray). The CBF blocks out regions *C* and *D* (shown in dark gray) to prevent violation of the specification. (**B**) The robot visits region *A* as directed by the FSPA-augmented MDP reward and transitions to $q_1$. The CBF now opens up *C* but still blocks *D*. (**C**) The robot visits region *C* to complete the task. The edge guards in the automaton above are defined as $b(q_0, q_0) = \neg\psi_A \wedge \neg\psi_B \wedge \neg\psi_C \wedge \neg\psi_D$, $b(q_0, q_1) = (\psi_A \wedge \neg\psi_C \wedge \neg\psi_D) \vee (\psi_B \wedge \neg\psi_C \wedge \neg\psi_D)$, $b(q_0, q_f) = (\psi_A \wedge \psi_C \wedge \neg\psi_D) \vee (\psi_B \wedge \psi_C \wedge \neg\psi_D)$, $b(q_0, q_{Tr}) = \psi_C \vee \psi_D$, $b(q_1, q_f) = \psi_C \wedge \neg\psi_D$, and $b(q_1, q_{Tr}) = \psi_D$. (**D**) Pictorial depiction of the FSPA-guided CBF. (**E**) The resulting policy at $q_0$, i.e., $\pi(s, q_0)$. (**F**) The resulting policy at $q_1$, i.e., $\pi(s, q_1)$.

In comparison, Eq. 14 defines local safe sets that change as the system evolves.

The goal of the CBF is to prevent the agent from entering the trap state. For the example in Fig. 7, at $q_0$, the robot can violate the specification by entering either region *C* or *D*. $C(q_0)$ is, therefore, any location outside of *C* and *D* (areas that are not grayed out). In practice, Eq. 14 blocks out the closer of *C* or *D* to the robot at each time step. Once the robot reaches *A* or *B* and transitions to $q_1$, the CBF allows for entering *C* but prevents entering *D*.

### *FSPA-guided CBF*

In this section, we take the output of the RL policy and the constraints (CBF) extracted from the FSPA in the previous section to design a shield that always keeps the system safe from violating the task specification (the CBF block in Fig. 6). Here, we divide the state space of the MDP into two parts, $S = S^r \times S^e$, where $S^r$ is the robot-related states (i.e., end-effector position and gripper position) and $S^e$ is the environment-related states (i.e., grill switch angle, scene object poses, etc.). Because our goal is to find a path, we abstract away the dynamics of the robot and use a simple linear dynamics to control the motion of the robot's end-effector frame

$$s^r_{t+1} = W_1 s^r_t + W_2(\pi_{rl}(s_t, q_t) + \bar{a}_t) \tag{15}$$

where $W_1$ and $W_2$ are constant matrices. In Eq. 15, the action consists of two parts: the action provided by the RL policy $\pi_{rl}: S \rightarrow A$ and the CBF action $\bar{a} \in A$. This known linear dynamics combined with the unknown environmental dynamics form the MDP transition function $pr(\cdot \mid \cdot, \cdot)$. The linear dynamics is only used in the CBF and

not in learning the RL policy. Notice that the RL policy takes as input $s_t \in S$; this way, the RL agent can learn a policy that takes into account the unknown dynamics, whereas we use CBF to steer the resultant path away from unsafe regions.

Given the set of CBF constraints $H(s_t, q_t)$ from Eq. 14, we solve the following optimization problem for $\bar{a}_t$

$$\bar{a}_t^{\star}(s_t, q_t) = \arg\min_{\bar{a}_t} \|\bar{a}_t\|_2$$

$$\text{s.t.} h^0(W_1 s_t^r + W_2(\pi(s_t, q_t) + \bar{a}_t), q_t) + (\alpha - 1)h^0(s_t, q_t) \geq 0$$

$$\vdots$$

$$h^i(W_1 s_t^r + W_2(\pi(s_t, q_t) + \bar{a}_t), q_t) + (\alpha - 1)h^i(s_t, q_t) \geq 0$$

$$\bar{a}^{\min} \leq \bar{a} \leq \bar{a}^{\max} \quad (16)$$

where $h_i \in H$.

In our experiments, given that we are generating a path by controlling the robot's end-effector frame $\boldsymbol{p}_{ee}$, we assume that collision is only position dependent and independent of orientation. Let $\boldsymbol{p}_{ee} = (\boldsymbol{p}_{ee}^l, \boldsymbol{p}_{ee}^o)$ (where $\boldsymbol{p}_{ee}^l$ is the 3D position and $\boldsymbol{p}_{ee}^o$ is the quaternion of the robot's end-effector frame). The linear dynamics in Eq. 15 have states $s^r = (\boldsymbol{p}_{ee}^l, g)$ where $g \in [0,1]$ is the robot's gripper position (0 for fully open and 1 for fully closed). The CBF calculated actions $\bar{a} = (\dot{\boldsymbol{p}}_{ee}^l, \dot{g})$ include the linear velocities of the end-effector frame as well as the open/closed velocity of the gripper. Let $a_{rl} = \pi_{rl}(s_t) = (a_{rl}^l, a_{rl}^o, a_{rl}^g)$, where $a_{rl}^l$ and $a_{rl}^o$ are the end effector's linear and angular velocities and $a_{rl}^g$ is the gripper velocity output by the RL policy. $(a_{rl}^l, a_{rl}^g)$ is then included in Eq. 15 as the RL component. Having calculated $\bar{a}^{\star}$, the next position on the path can be numerically integrated using actions $\bar{a}^{\star} + (a_{rl}^l, a_{rl}^g)$ and the current position $\boldsymbol{p}_{ee}^l$. The next orientation (quaternion) can be obtained by $\exp((\Delta t a_{rl}^o)/2) \star \boldsymbol{p}_{ee}^o$, where $\star$ denotes the quaternion inner product and $\Delta t$ is the time step.

The effect of the FSPA-guided CBF is pictorially depicted in Fig. 7D. Here, the agent's current state is at the middle of the ellipse, and the RL action aims to bring the agent to the boundary where return is highest while CBF keeps the agent within the safe region in a minimally interfering way (hence, the objective in Eq. 16).

An illustration of the final policy for our running example is provided in Fig. 7 (E and F). Figure 7E shows the policy at $q_0$. We can see that the policy guides the robot toward $A$ or $B$ (whichever is closer to its current position) while avoiding $C$ and $D$. Once $A$ or $B$ is reached (transition to $q_1$ occurs on the FSPA), the policy shown in Fig. 7F takes the agent to $C$ while avoiding $D$, hence satisfying the specification.

### *Training setup and experiment details*

Training is performed solely in simulation using the V-REP simulation environment (Fig. 1B). Given that the learned policy outputs a path in Euclidean space (no robot or environmental dynamics involved), simulation-to-real transfer can be achieved with accuracy.

The state of the system $S \subseteq \mathbb{R}^{45}$ consists of $\boldsymbol{p}_{ee}$, the 7D end-effector pose (position and quaternion); $g \in [0,1]$, the 1D gripper state, where $g = 0$ is fully open and $g = 1$ is fully closed; $\theta_s \in \mathbb{R}$, the angle of grill switch, in which a value of greater than $\pi/6$ is considered grill turned on and 0 is turned off; $\text{prob}_{hr} \in [0,1]$ is the probability of a ready-to-serve hot dog provided by an object detector; and $\boldsymbol{P}_o = \{\boldsymbol{p}_{\text{grill}}, \boldsymbol{p}_{\text{ketchup}}, \boldsymbol{p}_{\text{red}}, \boldsymbol{p}_{\text{blue}}, \boldsymbol{p}_{\text{green}}\}$ is the set of poses of all tracked items in the scene (red, blue, and green are the color of the tracked plates). We do not directly track the pose of the hot dog or its components (sausage and bun), but assume that we know their pose relative to the tracked objects (know their initial pose and where they are placed during the task).

We use proximal policy optimization (*37*) as the RL algorithm. Our policy is a feed-forward neural network with four hidden layers. The hidden layers have decreasing number of ReLU units from 400 to 100. We use the same architecture for the value function. The episode horizon is 500 for the hot dog cooking task and 300 for the serving task. Each policy and value update consists of five epochs on a trajectory batch of size 50 using a minibatch of 128 experience tuples. A learning rate of $3e^{-4}$ is used.

Randomization is important for the learned policy to generalize. At initialization of each episode, we randomize the poses of the ketchup and the three plates. The initial configuration of the robot is kept fixed, but the FSPA state is randomized to promote exploration at later stages of the task without having to first learn to complete early stages. For the serving task, the existence of a ready-to-serve hot dog in the scene and a customer is also randomized to help Baxter learn to make the right decisions.

The OptiTrack motion capture system was used to track object poses. A Logitech HD webcam running Darknet (*46*) was used for hot dog detection. Training in the simulated environment was performed on a Google Cloud n1-standard-8 instance running Ubuntu 16.04 [eight virtual central processing units (CPUs) with 30-gigabyte memory]. The hot dog cooking task was trained for 6 hours, and the serving task was trained for 3 hours. Experiment on the physical system was performed on a CyberPower PC with Intel i7 8 core CPU (4.2 GHz) and 31-gigabyte memory.

## SUPPLEMENTARY MATERIALS

robotics.sciencemag.org/cgi/content/full/4/37/eaay6276/DC1
Text
Fig. S1. FSPA for example Grasp task.
Fig. S2. FSPA for template formulas.
Table S1. Predicate definitions.
Movie S1. Demonstration of proposed technique in simulation.
Movie S2. Execution of learned policy on the physical robots.

## REFERENCES AND NOTES


1. D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in AI safety. arXiv:1606.06565 [cs.AI] (2016).
2. T. Arnold, D. Kasenberg, M. Scheutz, Value Alignment or Misalignment—What Will Keep Systems Accountable? *AAAI Workshops* (2017).
3. A. Y. Ng, D. Harada, S. J. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in *Proceedings of the Sixteenth International Conference on Machine Learning* (ICML, 1999), pp. 278–287.
4. P. F. Christiano, M. Abate, D. Amodei, Supervising strong learners by amplifying weak experts. arXiv:1810.08575 [cs.LG] (2018).
5. D. Hadfield-Menell, S. J. Russell, P. Abbeel, A. Dragan, Cooperative inverse reinforcement learning, in *Proceedings of Advances in Neural Information Processing Systems* (NeurIPS, 2016), pp. 3909–3917.
6. J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, S. Legg, Scalable agent alignment via reward modeling: A research direction. arXiv:1811.07871 [cs.LG] (2018).
7. G. Mason, R. Calinescu, D. Kudenko, A. Banks, Assured reinforcement learning with formally verified abstract policies, in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, Porto, Portugal, pp. 105–117.
8. O. Bastani, Y. Pu, A. Solar-Lezama, Verifiable reinforcement learning via policy extraction, in *Proceedings of Advances in Neural Information Processing Systems* (NeurIPS, 2018), pp. 2494–2504.
9. A. Adadi, M. Berrada, Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access* **6**, 52138–52160 (2018).

10. G. De Giacomo, L. Iocchi, M. Favorito, F. Patrizi, Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications, in *Proceedings of the International Conference on Automated Planning and Scheduling* (ICAPS, 2019), pp. 128–136.
11. A. Camacho, O. Chen, S. Sanner, S. A. McIlraith, Non-Markovian rewards expressed in LTL: Guiding search via reward shaping, in *The 10th Annual Symposium on Combinatorial Search* (SoCS, 2017), pp. 159–160.
12. D. Aksaray, A. Jones, Z. Kong, M. Schwager, C. Belta, Q-learning for robust satisfaction of signal temporal logic specifications, in *Proceedings of the IEEE 55th Conference on Decision and Control* (CDC, 2016), pp. 6565–6570.
13. A. Balakrishnan, J. Deshmukh, Structured reward functions using STL, in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (HSCC, 2019), pp. 270–271.
14. M. Wen, I. Papusha, U. Topcu, Learning from demonstrations with high-level side information, in *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (IJCAI, 2017), pp. 3055–3061.
15. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, U. Topcu, Safe reinforcement learning via shielding, in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (AAAI, 2018), pp. 2669–2678.
16. Q. Gao, D. Hajinezhad, Y. Zhang, Y, Kantaros, M, Zavlanos, Reduced variance deep reinforcement learning with temporal logic specifications, in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems* (ICCPS, 2019), pp. 237–248.
17. M. L. Littman, U. Topcu, J. Fu, C. I. Isbell, M. Wen, J. MacGlashan, Environment-independent task specifications via GLTL. arXiv:1704.04341 [cs.AI] (2017).
18. E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, D. Wojtczak, Omega-regular objectives in model-free reinforcement learning, in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS, Springer, 2019), pp. 395–412.
19. R. Toro Icarte, T. Q. Klassen, R. A. Valenzano, S. A. McIlraith, Using reward machines for high-level task specification and decomposition in reinforcement learning, in *International Conference on Machine Learning* (ICML, 2018), pp. 2112–2121.
20. B. Araki, K. Vodrahalli, T. Leech, C. I. Vasile, M. Donahue, D. Rus, Learning to plan with logical automata, in *Robotic: Science and Systems* (RSS, 2019), pp. 1–9.
21. S. Thiébaux, C. Gretton, J. K. Slaney, D. Price, F. Kabanza, Decision-theoretic planning with non-Markovian rewards. *J. Artif. Intell. Res.* **25**, 17–74 (2006).
22. F. Bacchus, C. Boutilier, A. J. Grove, Structured solution methods for non-Markovian decision processes, in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence* (AAAI, 1997), pp. 112–117.
23. F. Bacchus, C. Boutilier, A. Grove, Rewarding behaviors, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI, 1996), pp. 1160–1167.
24. A. D. Ames, J. W. Grizzle, P. Tabuada, Control barrier function based quadratic programs with application to adaptive cruise control, in *Proceedings of the 53rd IEEE Conference on Decision and Control* (IEEE, 2014), pp. 6271–6278.
25. M. Ohnishi, L. Wang, G. Notomista, M. Egerstedt, Barrier-certified adaptive reinforcement learning with applications to brushbot navigation. *IEEE Trans. Robot.* **35**, 1186–1205 (2019).
26. R. Cheng, G. Orosz, R. M. Murray, J. W. Burdick, End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks, in *The Thirty-Third AAAI Conference on Artificial Intelligence* (AAAI, 2019), pp. 3387–3395.
27. P. Nilsson, A. D. Ames, Barrier functions: Bridging the gap between planning from specifications and safety critical control, in *IEEE Conference on Decision and Control* (CDC, 2018), pp. 765–772.
28. X. Li, C.-I. Vasile, C. Belta, Reinforcement learning with temporal logic rewards, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS, 2017), pp. 3834–3839.
29. M. F. E. Rohmer, S. P. N. Singh, V-REP: A versatile and scalable robot simulation framework, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS, 2013), pp. 1321–1326.
30. J. Ho, S. Ermon, Generative adversarial imitation learning, in *Proceedings of Advances in Neural Information Processing Systems* (NeurIPS, 2016), pp. 4565–4573.
31. P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, P. Zhokhov, Openai baselines (2017); https://github.com/openai/baselines.
32. X. C. Ding, C. Belta, C. G. Cassandras, Receding horizon surveillance with temporal logic specifications, in *Proceedings of the 49th IEEE Conference on Descision and Control* (CDC, 2010), pp. 256–261.
33. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998).
34. V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
35. R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Proceedings of the 12th International Conference on Neural Information Processing Systems* (NeurIPS, 1999), pp. 1057–1063.
36. T. Degris, M. White, R. S. Sutton, Off-policy actor-critic. arXiv:1205.4839 [cs.LG] (2012).
37. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms. arXiv:1707.06347 [cs.LG] (2017).
38. C. Baier, J. P. Katoen, *Principles of Model Checking* (MIT Press, 2008).
39. R. Cheng, G. Orosz, R. M. Murray, J. W. Burdick, End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. arXiv:1903.08792 [cs.LG] (2019).
40. A. Agrawal, K. Sreenath, Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation, in *Robotics: Science and Systems* (2017).
41. Gurobi Optimization, Gurobi optimizer reference manual (2018).
42. T. Latvala, Efficient model checking of safety properties, in *International SPIN Workshop on Model Checking of Software* (Springer, 2003), pp. 74–88.
43. C. Vasile, A. Ulusoy, LTL Optimal Multi-Agent Planner (LOMAP), *Github repository*, (2017); https://github.com/wasserfeder/lomap.
44. O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* (Springer, 2004), pp. 152–166.
45. K. Y. Rozier, "Explicit or symbolic translation of linear temporal logic to automata," thesis, Rice University (2013).
46. M. Bjelonic, YOLO ROS: Real-time object detection for ROS, *Github repository*, (2016–2018); https://github.com/leggedrobotics/darknet_ros.


**Acknowledgments:** We thank R. Khurshid for giving us access to her Kinova Jaco Robotic Manipulator. **Funding:** This work was supported by the National Science Foundation under grants IIS-1723995 and CMMI-1400167. **Author contributions:** X.L. created the theory, software, simulation, and implementation and wrote large portions of the paper. Z.S. worked on the composition of the task specification and the knowledge graphs, wrote portions of the paper, and edited the paper. G.Y. worked on the CBF software and the hardware implementation and experiments and generated the video. C.B. provided the funding and wrote and edited the paper. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data needed to evaluate the conclusions in the paper are present in the paper or the Supplementary Materials or have been deposited in the database https://xli4217@bitbucket.org/xli4217.robot_cooking_data.

Submitted 5 July 2019
Accepted 4 November 2019
Published 18 December 2019
10.1126/scirobotics.aay6276

**Citation:** X. Li, Z. Serlin, G. Yang, C. Belta, A formal methods approach to interpretable reinforcement learning for robotic planning. *Sci. Robot.* **4**, eaay6276 (2019).

Science Robotics

# A formal methods approach to interpretable reinforcement learning for robotic planning

Xiao Li, Zachary Serlin, Guang Yang, and Calin Belta

*Sci. Robot.* **4** (37), eaay6276. DOI: 10.1126/scirobotics.aay6276



*Science Robotics* (ISSN 2470-9476) is published by the American Association for the Advancement of Science. 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.