Science Robotics
AAAS

robotics.sciencemag.org/cgi/content/full/4/37/eaay6276/DC1

Supplementary Materials for

A formal methods approach to interpretable reinforcement learning for robotic planning

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

*Corresponding author. Email: xli87@bu.edu

Published 18 December 2019, *Sci. Robot.* **4**, eaay6276 (2019)
DOI: 10.1126/scirobotics.aay6276

The PDF file includes:

Other Supplementary Material for this manuscript includes the following:

(available at robotics.sciencemag.org/cgi/content/full/4/37/eaay6276/DC1)

Movie S1 (.mp4 format). Demonstration of proposed technique in simulation.
Movie S2 (.mp4 format). Execution of learned policy on the physical robots.

Supplementary Text

TLTL syntax snd semantics

blueWe consider tasks specified with *Truncated Linear Temporal Logic* (TLTL) *(20)*. The syntax of TLTL is defined inductively in backus-naur form as

$$\phi := \top \mid f(s) < c \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathcal{F}\phi \mid \phi \mathcal{U} \psi \mid \mathcal{X}\phi \ ,$$

where $\top$ is the True Boolean constant. $s \in S$ is an MDP state in Definition 1; $f(s) < c$ is a predicate over the MDP states where $c \in \mathbb{R}$; $\neg$ (negation/not) and $\wedge$ (conjunction/and), $\vee$ (disjunction/or) are Boolean connectives. $\mathcal{F}$ (eventually), $\mathcal{U}$ (until), $\mathcal{X}$ (next), are temporal operators. Here we also define a set of derived temporal operators. A finite time *Always* operator by $\mathcal{G}\phi = \neg\mathcal{F}\neg\phi$. *Imply* is defined as $\phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$. The *Then* operator is defined as $\phi_1 \ \mathcal{T} \ \phi_2 = \phi_1 \mathcal{X} \mathcal{F} \phi_2$. Concatenation of the *Then* operator follows $\phi_1 \ \mathcal{T} \ \phi_2 \ \mathcal{T} \phi_3 = \phi_1 \mathcal{X} \mathcal{F}(\phi_2 \mathcal{X} \mathcal{F} \phi_3)$.

We denote $s_t \in S$ to be the MDP state at time t, and $s_{t:t+k}$ to be a sequence of states (state trajectory) from time t to $t+k$, i.e., $s_{t:t+k} = s_t s_{t+1} ... s_{t+k}$. The Boolean semantics of TLTL are defined as

$$\begin{aligned}
s_{t:t+k} \models f(s) < c \quad &\Leftrightarrow \quad f(s_t) < c, \\
s_{t:t+k} \models \neg\phi \quad &\Leftrightarrow \quad \neg(s_{t:t+k} \models \phi), \\
s_{t:t+k} \models \phi \Rightarrow \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \Rightarrow (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \phi \wedge \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \wedge (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \phi \vee \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \vee (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \mathcal{X}\phi \quad &\Leftrightarrow \quad (s_{t+1:t+k} \models \phi) \wedge (k > 0), \\
s_{t:t+k} \models \mathcal{F}\phi \quad &\Leftrightarrow \quad \exists t' \in [t, t+k) \ s_{t':t+k} \models \phi, \\
s_{t:t+k} \models \phi \ \mathcal{U} \ \psi \quad &\Leftrightarrow \quad \exists t' \in [t, t+k) \ s.t. \ s_{t':t+k} \models \psi \\
&\qquad \wedge (\forall t'' \in [t, t') \ s_{t'':t'} \models \phi).
\end{aligned}$$

A trajectory $s_{0:T}$ is said to satisfy formula ϕ if $s_{0:T} \models \phi$.

The quantitative semantics (also referred to as robustness of a sequence of states/state trajectory with respect to a TLTL formula) is defined recursively as

$$
\begin{aligned}
\rho(s_{t:t+k}, \top) &= \rho_{max}, \\
\rho(s_{t:t+k}, f(s_t) < c) &= c - f(s_t), \\
\rho(s_{t:t+k}, \neg\phi) &= -\rho(s_{t:t+k}, \phi), \\
\rho(s_{t:t+k}, \phi \Rightarrow \psi) &= \max(-\rho(s_{t:t+k}, \phi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \phi_1 \wedge \phi_2) &= \min(\rho(s_{t:t+k}, \phi_1), \rho(s_{t:t+k}, \phi_2)), \\
\rho(s_{t:t+k}, \phi_1 \vee \phi_2) &= \max(\rho(s_{t:t+k}, \phi_1), \rho(s_{t:t+k}, \phi_2)), \\
\rho(s_{t:t+k}, \mathcal{X}\phi) &= \rho(s_{t+1:t+k}, \phi) \ (k > 0), \\
\rho(s_{t:t+k}, \mathcal{F}\phi) &= \max_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \phi)), \\
\rho(s_{t:t+k}, \phi \,\mathcal{U}\, \psi) &= \max_{t' \in [t, t+k)} (\min(\rho(s_{t':t+k}, \psi), \\
&\qquad \min_{t \in [t, t')} \rho(s_{t:t'}, \phi))).
\end{aligned}
$$

where ρ_{max} represents the maximum robustness value. A robustness of greater than zero implies that $s_{t:t+k}$ satisfies ϕ and vice versa ($\rho(s_{t:t+k}, \phi) > 0 \Rightarrow s_{t:t+k} \models \phi$ and $\rho(s_{t:t+k}, \phi) < 0 \Rightarrow s_{t:t+k} \not\models \phi$). The robustness is used as a measure of the level of satisfaction of a trajectory $s_{0:T}$ with respect to a TLTL formula ϕ.

Predicate definitions

Here we provide the definitions of predicates used in specifying the hotdog cooking and serving tasks. The predicates are used as a medium to ground the high-level task specification with low-level state features. We write all predicates p in the form $f(s) > 0$ where $f(s)$ is the predicate function. We classify predicates into two categories - actionable and non-actionable. Actionable predicates are ones that the robot can execute actions to to effect the value of the predicate function. None-actionable predicates are environment predicates which function val-

ues can not be effected by the robot. In our formulations, actionable predicate functions are mostly thresholded distance functions whereas non-actionable predicate functions are indicator function $\mathbf{1} : \mathbb{R} \to \{0, 1\}$ that outputs 0 if the the input is smaller than 0 and outputs 1 if the input is greater than zero. We then scale the non-actionable predicate functions to take values $\{-k, k\}$ (k usually takes larger value than the upper bound of the actionable predicate robustness degree). This is to enforce the idea that when environmental conditions are met (non-actionable predicates have large positive robustnesses), the robustness of an edge guard (with both actionable and non-actionable predicates) is the robustness of the actionable predicate (i.e. the agent can take actions to increase the edge guard robustness). However, if when environmental conditions are not met (non-actionable predicates have large negative robustnesses), there is nothing the agent can do to change the edge guard robustness and hence transition from that edge. The distance function $dist : \mathbb{R}^3 \times S^3 \to \mathbb{R}$ (S^3 is the unit circle) takes in pose vector consisting of position coordinates and quaternions and output a distance metric. In our experiments, this metric is the sum of the Euclidean distance between two 3D coordinates and the quaternion distance defined by $\arccos(2(q_1 \star q_2)^2 - 1)$ where $\star$ is the quaternion inner product.

In Table S1, $\boldsymbol{p}_{ee}$ is the end-effort pose (position and quaternion), $\boldsymbol{p}_g$ is a goal pose. g is the gripper position with value 0 when the gripper is fully open and value 1 when fully close. θ_s is the position of the grill switch with 0 when turned off and value $\pi/6$ when turned on. $prob_{hr}$ is the probability of a read-to-serve hotdog in sight provided by an object detector. t is a timer (in seconds) that starts at the beginning of the task and t_d is a time duration (in seconds). $\boldsymbol{p}_c \in \mathbb{R}^3$ and $\boldsymbol{l}_a \in \mathbb{R}^3$ define an ellipsoid centered at $\boldsymbol{p}_c$ with axis $\boldsymbol{l}_a$.

Example task specification and knowledge base

In this section, we provide the TLTL formulas of the $Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$ example in the Results section.

$$Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) = \mathcal{F}\left((Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) \wedge GripperOpen(g)) \wedge \mathcal{F}\, GripperClose(g)\right) \wedge \left(\neg GripperClose(g)\ \mathcal{U}\ (Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) \wedge GripperOpen(g))\right), \tag{1}$$

where $\boldsymbol{p}_g$ is the grasping pose of certain object. In English, the formula indicates that to pick object g, the robot's end-effector needs to *eventually reach $\boldsymbol{p}_g$ with gripper open. And then eventually close gripper. And do not close gripper until $\boldsymbol{p}_g$ is reached at a gripper open state*. The FSPA for $Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$ is shown in Figure S1 (A).

The knowledge base is defined as

$$K^{grasped} = \{\neg(GripperOpen(g) \wedge GripperClose(g)), InSafeRegion(\boldsymbol{p}_{ee})\}. \tag{2}$$

Here $InSafeRegion(\boldsymbol{p}_{ee}) = \left(\bigwedge_{i\in\{x,y,z\}}(\boldsymbol{p}^i_{ee} - \boldsymbol{r}^i_{min} > 0)\right) \wedge \left(\bigwedge_{i\in\{x,y,z\}}(\boldsymbol{r}^i_{max} - \boldsymbol{p}^i_{ee} > 0)\right)$ defines a cuboid region that the end-effector has to stay within ($\boldsymbol{r}_{max}, \boldsymbol{r}_{min}$ are the max and min coordinates that define the cuboid). The knowledge base contains two pieces of information - the gripper can not be open and close at the same time; the end-effector needs to always be in the safe region. The final task specification is as follows

$$Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) \wedge \mathcal{G}\left(\neg(GripperClose(g) \wedge GripperOpen(g)) \wedge\ InSafeRegion(\boldsymbol{p}_{ee})\right). \tag{3}$$

The FSPA for the combined task specification is provided in Figure S1 (B).

Template formulas

We define the template formulas used in Equations (10) and (11) as follows

$$Picked(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) = (Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) \wedge GripperOpen(g))\ \mathcal{T}\ GripperClose(g), \tag{4}$$

$$Placed(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) = (Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_g) \wedge GripperClose(g))\ \mathcal{T}\ GripperOpen(g), \tag{5}$$

$$PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{g_1}, \boldsymbol{p}_{g_2}) = Picked(\boldsymbol{p}_{ee}, \boldsymbol{p}_{g_1})\ \mathcal{T}\ Placed(\boldsymbol{p}_{ee}, \boldsymbol{p}_{g_2}), \tag{6}$$

$$
\begin{aligned}
KetchupApplied(s) = & Picked(\boldsymbol{p}_{ee}, \boldsymbol{p}_{ketchup}) \ \mathcal{T} \ Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}'_{bun}) \ \mathcal{T} \ GripperSqueeze(g) \ \mathcal{T} \\
& Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}''_{bun}) \ \mathcal{T} \ (GripperClose(g) \wedge \neg GripperSqueeze(g)) \ \mathcal{T} \\
& Placed(\boldsymbol{p}_{ee}, \boldsymbol{p}_{table}),
\end{aligned} \tag{7}
$$

$$
GrillTurnedOn(\theta_s, g) = GripperClose(g) \ \mathcal{T} \ GrillSwitchOn(\theta_s), \tag{8}
$$

$$
GrillTurnedOff(\theta_s, g) = GripperClose(g) \ \mathcal{T} \ GrillSwitchOff(\theta_s). \tag{9}
$$

All of the template formulas used here are describe sequential movements with appropriate gripper positions. As expected, Their FSPAs (shown in Figure S2) have a linear structure. In Equation (7), $\boldsymbol{p}_{ketchup}$ is the tracked ketchup position. $\boldsymbol{p}'_{bun}$ and $\boldsymbol{p}''_{bun}$ are start and end poses for squeezing ketchup onto the hotdog (these poses are given relative to the tracked blue plate). $\boldsymbol{p}_{table}$ is the pose relative to the table where the ketchup should be placed (table pose is static and known a priori). The reason that $KetchupApplied(s)$ is defined in this "waypoint" form is because we currently do not have an effective way to evaluate whether applying ketchup is successful (e.g. using camera images). If such mechanism is readily available, $KetchupApplied(s)$ can simply be defined as a predicate similar to $HotdogReady(prob_{hr})$.

The main reason for using template formulas in this work is for the convenience of presentation and explanation. During learning, the non-hierarchical specifications and their full FSPAs are fed to the RL agent. Writing the task in a hierarchical fashion however presents the opportunity for hierarchical learning using the skill templates (and their individual FSPAs) which can improve sample efficiency. In the case where the low level skills are learned individually using the skill templates, composing a task using these templates also promotes skill reuse. These extension will be explored in future work.

Hotdog cooking and serving tasks

In this section, we formally define the hotdog cooking task and the hotdog serving task. Formula $HotdogCooked$ describes a sequence of conditions that needs to be met for a hotdog to be considered successfully made, much like how a recipe is written. The formula is as follows

$$\begin{aligned} HotdogCooked(s) = &\mathcal{F}(GrillTurnedOn(\theta_s, g)\ \mathcal{T}\ PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{sausage}, \boldsymbol{p}_{grill})\ \mathcal{T} \\ &Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_{home})\ \mathcal{T}\ TimePassed(t, 600)\ \mathcal{T} \\ &PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{sausage}, \boldsymbol{p}_{bun})\ \mathcal{T} KetchupApplied(s)\ \mathcal{T} \\ &GrillTurnedOff(\theta_s, g)\ \mathcal{T}\ Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_{home})), \end{aligned} \tag{10}$$

where $\boldsymbol{p}_{home}$ is the robot's home position. $TimePassed(t, 600)$ is included here for the completeness of the specification, it is neglected during training and evaluation, To simplify the formula and promote reusability of sub-formulas, we do not specify the task directly from base predicates but in a hierarchical fashion. In Equation (10), $GrillTurnedOn$, $GrillTurnedOff$, $PickedAndPlaced$ and $KetchupApplied$ are formulas by themselves. Their definitions and corresponding FSPAs are provided in the supplementary material. We refer to these lower-level formulas as *template formulas*. Every TLTL formula can be a template formula that can be used by a higher level task specification.

Equation (11) specifies the hotdog serving task. It defines a set of conditions that need to be met for the robot to serve the hotdog - it needs to detect both a ready-to-serve hotdog and a customer "asking" for a hotdog.

$$\begin{aligned} HotdogServed(s) = &\mathcal{F}\ PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{hotdog}, \boldsymbol{p}_{green}) \wedge \\ &(\neg PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{hotdog}, \boldsymbol{p}_{green})\, \mathcal{U} \\ &(HotdogReady(prob_{hr}) \wedge Customer(\boldsymbol{p}_{customer}))). \end{aligned} \tag{11}$$

Here $\boldsymbol{p}_{customer} = \boldsymbol{p}_{green}$ which is the customer's plate that is being tracked. There is a customer if his/her plate is detected to be within a cuboid serving zone (shown in Figure 1 D). The detection formula is

$$Customer(\boldsymbol{p}_{customer}) = \big(\bigwedge_{i \in \{x,y,z\}} (\boldsymbol{p}^i_{customer} - \boldsymbol{cr}^i_{min} > 0) \big) \wedge \big(\bigwedge_{i \in \{x,y,z\}} (\boldsymbol{cr}^i_{max} - \boldsymbol{p}^i_{customer} > 0) \big), \tag{12}$$

where $\boldsymbol{cr}_{min}, \boldsymbol{cr}_{max}$ are min and max coordinates defining the serving zone.

In Equations (10) and (11), $\boldsymbol{p}_{sausage}, \boldsymbol{p}_{bun}, \boldsymbol{p}_{hotdog} \in \{\boldsymbol{p}_{grill}, \boldsymbol{p}_{red}, \boldsymbol{p}_{blue}, \boldsymbol{p}_{green}\}$. We do not directly identify and track the sausage, bun and hotdog, but we know where they initially are and where the robot has moved them.

We use the following knowledge base

$$\begin{aligned} K = &\{InSafeRegion(\boldsymbol{p}_{ee}), \neg(GripperClose(g) \wedge GripperOpen(g))\} \cup \\ &\{\neg(Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}^i_g) \wedge Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}^j_g) \mid i, j \in \{grill, ketchup, red, blue, green\}, i \neq j\}, \end{aligned} \tag{13}$$

where $InSafeRegion$ is defined similarly to that in the previous section as

$$\begin{aligned} InSafeRegion(\boldsymbol{p}_{ee}) = &\big(\bigwedge_{i \in \{x,y,z\}} (\boldsymbol{p}^i_{ee} - \boldsymbol{sr}^i_{min} > 0) \big) \wedge \big(\bigwedge_{i \in \{x,y,z\}} (\boldsymbol{sr}^i_{max} - \boldsymbol{p}^i_{ee} > 0) \big) \wedge \\ &\neg InEllipsoidRegion(\boldsymbol{p}_{ee}, \boldsymbol{p}_{grill}, \boldsymbol{l}_{grill}), \end{aligned} \tag{14}$$

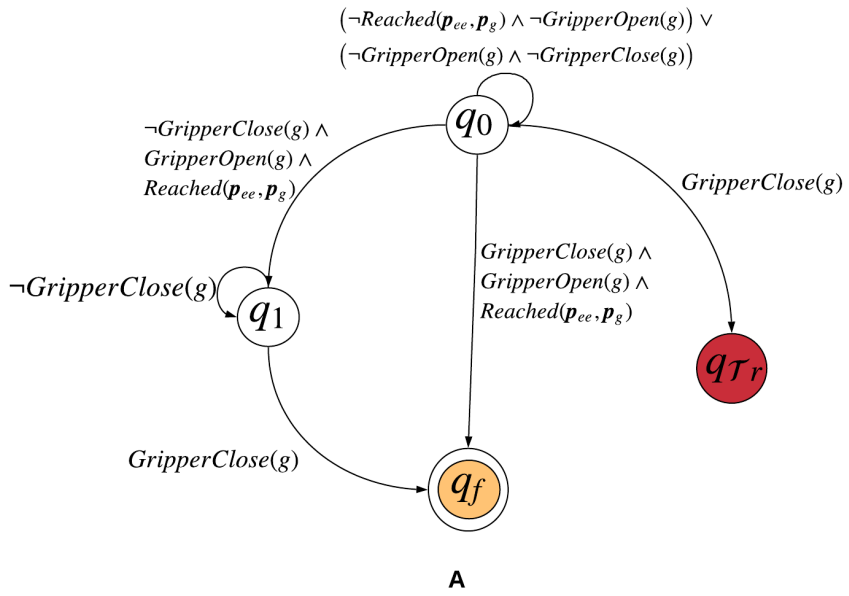
which requires the robot to stay within a cuboid safe region while avoiding collision with the grill (represented by a fitted ellipsoid). All predicates in ***InSafeRegion***($\boldsymbol{p}_{ee}$) are treated as non-actionable due to the fact that safety is guaranteed by the CBFs and thus does not need to be learned. Both the serving zone and safe region are depicted in Figure 1. Notice in Figures 1 (C) and (D) that the safe region for the grill is different for Baxter and Jaco. This is because Jaco needs access to the grill panel which means we can not have the ellipsoid cover the entire grill. Instead, the ellipsoid covers the front half of the grill which is most likely to collide with the arm during its motion.

The knowledge base contains three types of constraints - the end-effector needs to in the safe region; the robot's gripper can not be open and close at the same time; the robot can not reach any two places at the same time. The final task specifications with be in the form $HotdogServed(s) \wedge \mathcal{G} \bigwedge_{\psi_i \in K} \psi_i$ and $HotdogCooked(s) \wedge \mathcal{G} \bigwedge_{\psi_i \in K}^{n} \psi_i$. As discussed in the previous section, the incorporation of the knowledge base will effect the resulting FSPA by pruning infeasible edges and adding constraints to make it more context aware.

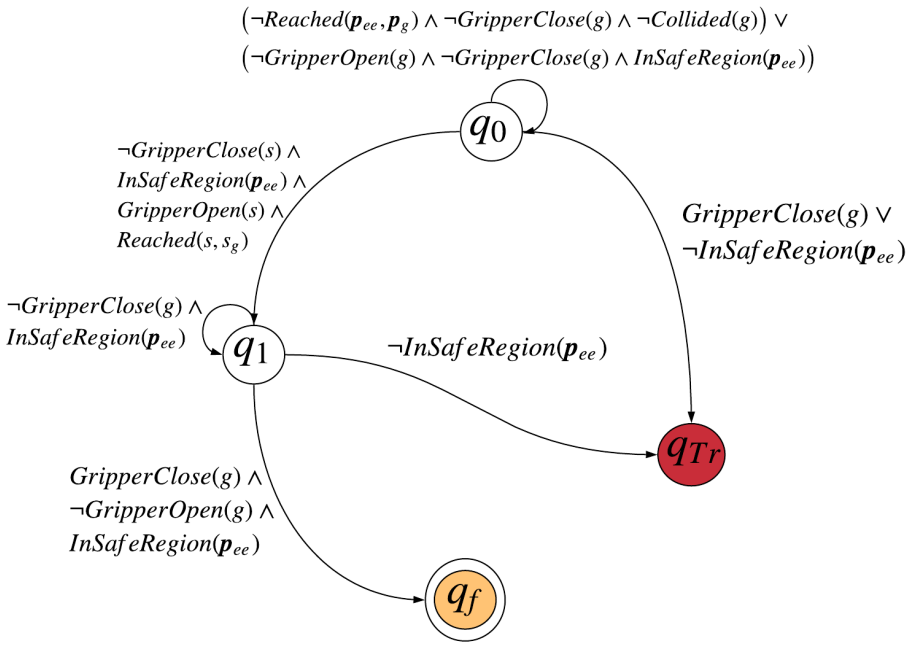
The FSPAs of $HotdogCooked$ and $HotdogServed$ (without the knowledge base) are shown in Figures 3 (C) and (D). Note that if we substitute all template formulas with their base predicate alternatives (which is what is used in later planning and learning).

Table S1. Predicate definitions. The predicates are true when the corresponding predicate function evaluates to a positive number. Actionable predicates are ones that the robot can execute actions to effect their truth value. Non-actionable predicates describe environmental factors that the robot has no control over. The non-actionable predicate functions are scaled to take value $\{-k, k\}$.

Predicate	**Predicate Function**
Actionable	
$Reached(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$	$\epsilon_0 - dist(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$
$GripperOpen(g)$	$\epsilon_1 - g$
$GripperClose(g)$	$g - \epsilon_2$
$GripperSqueeze(g)$	$g - \epsilon_3$, $\epsilon_3 > \epsilon_2$
$GrillSwitchOn(\theta_s)$	$\theta_s - \epsilon_4$
$GrillSwitchOff(\theta_s)$	$\epsilon_5 - \theta_s$
Non-actionable	
$HotdogReady(prob_{hr})$	$k(2 \times \mathbf{1}(prob_{hr} - \epsilon_5) - 1)$
$TimePassed(t, t_d)$	$k(2 \times \mathbf{1}(t - t_d) - 1)$
$InEllipsoidRegion(\boldsymbol{p}_{ee}, \boldsymbol{p}_c, \boldsymbol{l}_a)$	$k(2 \times \mathbf{1}(1 - \sum_{i \in \{x,y,z\}}((\boldsymbol{p}^i_{ee} - \boldsymbol{p}^i_c)/\boldsymbol{l}^i_a)^2) - 1)$



A



B

Fig. S1. FSPA for example $Grasp$ **task.** **(A)** $Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$. **(B)** $Grasped(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$ with knowledge base.

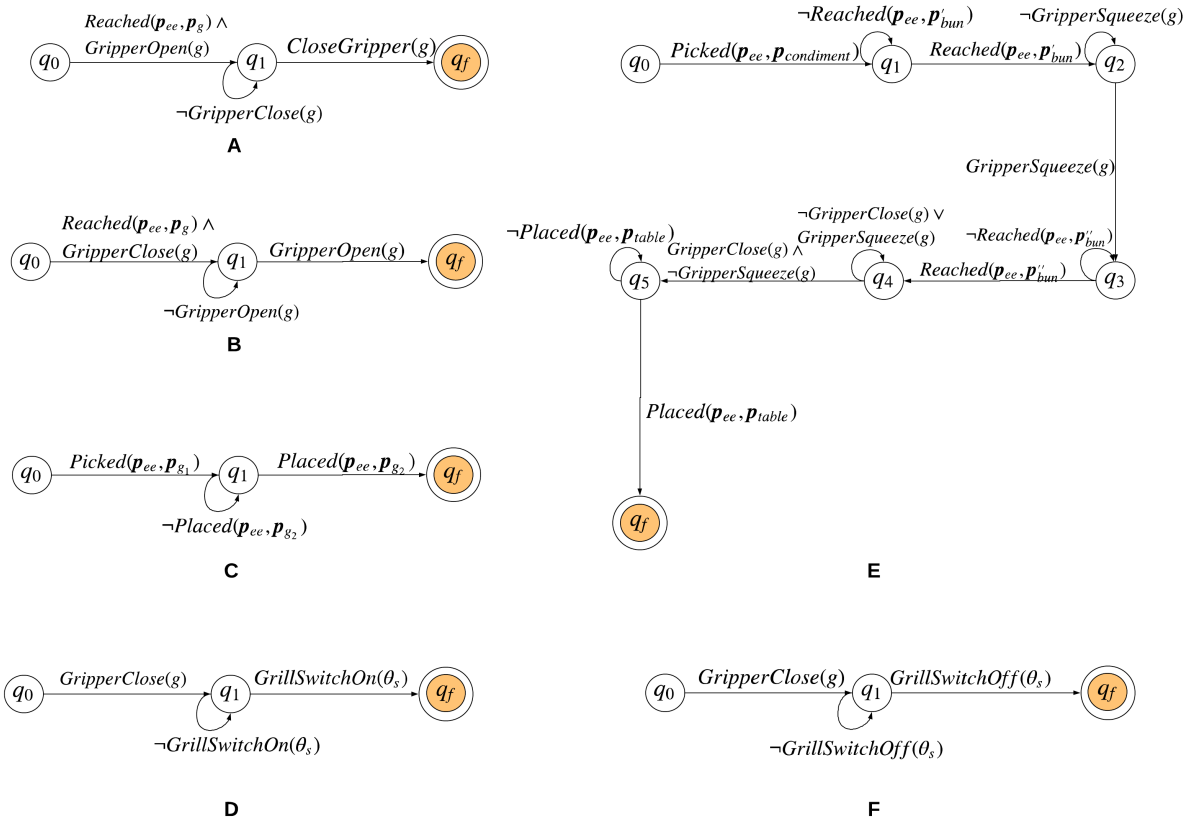


Fig. S2. FSPA for template formulas. **(A)** $Picked(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$. **(B)** $Placed(\boldsymbol{p}_{ee}, \boldsymbol{p}_g)$. **(C)** $PickedAndPlaced(\boldsymbol{p}_{ee}, \boldsymbol{p}_{g_1}, \boldsymbol{p}_{g_2})$. **(D)** $GrillTurnedOn(\theta_s, g)$. **(E)** $KetchupApplied(s)$. **(F)** $GrillTurnedOff(\theta_s, g)$