# Automatic Encoding and Repair of Reactive High-Level Tasks with Learned Abstract Representations

Adam Pacheck[1], George Konidaris[2], and Hadas Kress-Gazit[1]

[1] Cornell University, Ithaca NY 14850, USA,
{akp84,hadaskg}@cornell.edu
[2] Brown University, Providence RI 02912, USA,
gdk@cs.brown.edu

**Abstract.** We present a framework that, given a set of skills a robot can perform, abstracts sensor data into symbols that are used to automatically encode the robot's capabilities in Linear Temporal Logic (LTL). We specify reactive high-level tasks based on these capabilities, for which a strategy is automatically synthesized and executed on the robot, if the task is feasible. If a task is not feasible given the robot's capabilities, our framework automatically suggests additional skills for the robot that would make the task feasible. We demonstrate our framework on a Baxter robot manipulating blocks on a table.

## 1 Introduction

One of the central challenges in robotics is to create robots that can autonomously perform complex tasks in different environments. Adding to the challenge is the desire to specify the task at a high-level – the "what" and not the "how". For example, we would like to specify a task for a robotic waiter as "we have five people sitting at the table, make sure the table is set and that their water glasses are constantly refilled" instead of providing a sequence of commands of the form "place a fork by plate 1, then place a knife by plate 1", etc. High-level tasks may be reactive, requiring the robot to change its behavior based on environment input, for example, the choice of whether to pour more water in a cup will depend on the water level in the cup and on the person's requests. Additionally, the robot actions, referred to here as skills, may have nondeterministic outcomes – a fork may fall on the plate, or off the table – requiring care in the choice of actions. It is also possible that the robot does not have the right skills to perform the requested task. In these cases, we would like feedback to be provided to the user.

We automatically abstract and encode the robot's capabilities in Linear Temporal Logic (LTL). We do so with abstractions created from sensor data [1]. We take into account unmodeled nondeterminism in robot skills, such as a fork slipping from a gripper and landing on the plate or floor instead of next to the plate. Our framework allows a user to specify a reactive high-level task that takes into account the robot's skills, learned symbols, and additional user defined symbols. If the task the user specifies is infeasible, we automatically suggest additional skills for the robot that would enable it to complete the task.

**Contributions:** Given a set of skills a robot is able to perform, we present a framework for (1) automatically encoding into LTL, from sensor data, the robot capabilities that

are then used to automatically synthesize high-level robot behaviors and (2) if a task is not feasible due to a missing skill, automatically suggesting skills that repair the task (i.e. make it executable by the robot). We demonstrate our approach on a Baxter robot manipulating blocks.

**Related work:** Typically, to enable robots to perform high-level tasks, the robot's capabilities, state, and environment are abstracted into predicates that include the robot's skills and their effects on the robot's state and environment. These predicates are often abstractions of the state space [2,3,4]. Automated creation of abstractions has been studied in various domains [1], including the manipulation domain [5]. In He, et al. [5], the authors sample the state space and plan trajectories to automatically create abstractions for a manipulation domain before executing a reactive task. We use sensor information from repeated executions of a given set of skills to create abstractions [1].

Planning algorithms and frameworks use abstractions to find a sequence of commands to reach a goal state [6,7,8]. If there is uncertainty in the outcome of skills, planners exist that will return the sequence of skills that is most likely to accomplish the task, but often require replanning when an unexpected effect occurs (e.g. [9]). When there is uncertainty in sensing, initial state, and actuation, conditional planning can return a plan that will take these into account [8]. If the robot can not observe the environment at all, conformant planning generates a plan for a robot to accomplish its goal [8]. However, these goals are typically defined as a desired end state, while we consider more complex tasks. Additionally, if a planner fails to find a plan, to the best of our knowledge, planners are unable to suggest new skills that result in a valid plan.

Work in synthesis for robotics [2] allows us to specify a reactive high-level task for a robot and produce a strategy guaranteed to succeed, or a proof that the task cannot be accomplished (e.g. [10,5,11,12,13,14]). Using synthesis, robots can find a strategy that accounts for all possible outcomes of their skills and changes in the environment. If completion of a task cannot be guaranteed, synthesis algorithms can provide explanations as to what caused the problem (e.g. [15,16,17]). This allows repair suggestions and automated fixes to be made to specifications [18,19]. In prior work, these suggestions either restricted the environment for which the behavior could be guaranteed or removed robot goals. In this work, we suggest changes that extend the capabilities of the robot through new skills that are grounded in the sensor-based abstract representation.

**Approach:** In this paper, we automatically use sensor data to encode robot capabilities into LTL formulas for use with synthesis algorithms. The abstractions are automatically created from sensor data using recent work [1]. If a specified task can be completed, we find a strategy to do so and execute it on the robot. If the task cannot be performed, we suggest skills that would enable the robot to complete the task.

## 2 Background

To automatically encode the robot's capabilities, allow a user to specify reactive high-level tasks, and offer skill suggestions to repair tasks that cannot be completed, we use work in symbol generation [1] and Linear Temporal Logic (LTL) synthesis [2].

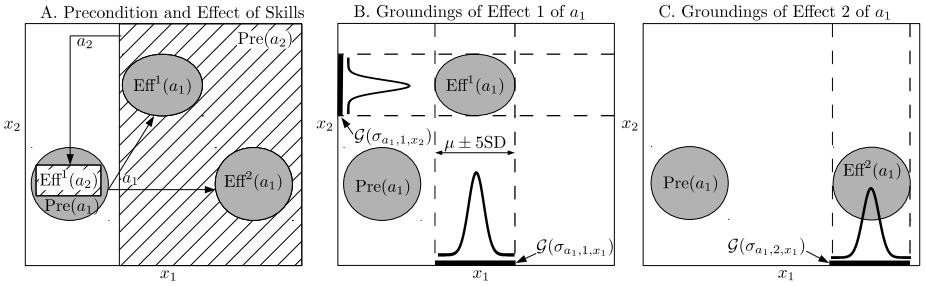


Fig. 1: Depiction of Example 1, demonstrating the symbol generation process. (A) Two skills $a_1$ and $a_2$ and their precondition and effect sets. (B,C) The grounding sets of the symbols generated from skill $a_1$. The robot considers the value of $x_1$ and $x_2$ when deciding if it can apply $a_1$, so pre-mask$(a_1) = \{\top, \top\}$. It only considers the value of $x_1$ when deciding if it can apply $a_2$, so pre-mask$(a_2) = \{\top, \bot\}$. The application of $a_1$ either changes $x_1$ and $x_2$ or only $x_1$, so eff-mask$^1(a_1) = \{\top, \top\}$ and eff-mask$^2(a_1) = \{\top, \bot\}$. In effect 1 of $a_1$, $\sigma^{\top}_{\text{eff}^1(a_1)} = \{\sigma_{a_1,1,x_1}, \sigma_{a_1,1,x_2}\}$ become $\top$ and $\sigma^{\bot}_{\text{eff}^1(a_1)} = \{\sigma_{a_1,2,x_1}, \sigma_{a_2,1,x_1}, \sigma_{a_2,1,x_2}\}$ become $\bot$. In effect 2 of $a_1$, $\sigma^{\top}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,2,x_1}\}$ becomes $\top$, $\sigma^{\bot}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,1,x_1}, \sigma_{a_2,1,x_1}\}$ becomes $\bot$, and $\sigma^{\text{stay}}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,1,x_2}, \sigma_{a_2,1,x_2}\}$ do not change.

### 2.1 Symbol Generation

The process of symbol generation [1] automatically constructs a set of symbols which are used for planning. To illustrate the main ideas of symbol generation, we introduce Example 1, shown in Figure 1. In this two dimensional space, a robot has two skills $a_1$ and $a_2$ that allow it to move between regions, as shown by the arrows.

The input to the symbol generation process is a set of skills $A$ operating over a world with a continuous state space $(x_1, \ldots, x_n) \in X \subseteq \mathbb{R}^n$. Each skill $a \in A$ has a region from which it is applicable, termed the *precondition* of $a$, $\text{Pre}(a) \subseteq X$. The application of $a$ will result in the state being in one of $j \in \{1, \ldots, k(a)\}$ possible *effect sets*, denoted by $\text{Eff}^j(a) \subseteq X$. In Figure 1, $a_1$ has a nondeterministic outcome, resulting in either $\text{Eff}^1(a_1)$ or $\text{Eff}^2(a_1)$. We create a finite set of propositional symbols $\Sigma$ representing the effect sets of $a \in A$. Each $\sigma \in \Sigma$ is grounded via the *grounding operator* $\mathcal{G}$ to the state space $X$.

In determining if a skill can be applied, the values of some $x_i$ may matter, while the values of others may not. We denote this in the *precondition mask* of $a$, pre-mask$(a) \in \mathbb{B}^n$, where pre-mask$(a)(i) = \top$ if the value of $x_i$ influences if $a$ can be applied and $\bot$ otherwise. We create a classifier to test inclusion in $\text{Pre}(a)$, which is defined for $x_i$ for which pre-mask$(a)(i) = \top$ . Similarly, when a skill is applied, it may change some or all of the state variables. We record this in the *effect mask*, eff-mask$^j(a) \in \mathbb{B}^n$, where eff-mask$^j(a)(i) = \top$ if the value of $x_i$ is modified by the application of $a$ in the $j^{th}$ outcome and $\bot$ otherwise. In Figure 1, to apply $a_1$, the values of both $x_1$ and $x_2$ matter, so pre-mask$(a_1) = \{\top, \top\}$. We only consider the value of $x_1$ to determine

if $a_2$ can be applied, so pre-mask$(a_2) = \{\top, \bot\}$. Effect 1 of $a_1$ changes the value of $x_1$ and $x_2$ so eff-mask$^1(a_1) = \{\top, \top\}$, while effect 2 only changes the value of $x_1$, so eff-mask$^2(a_1) = \{\top, \bot\}$.

A separate $\sigma$ is created[1] for each $a$, $j$, and $x_i$ when eff-mask$^j(a)(i) = \top$. We add subscripts to $\sigma$ and say each $\sigma_{a,j,x_i}$ grounds to a set over one state variable $x_i$. In our work, this is done by fitting a Gaussian over the raw data in $x_i$ and considering $\mathcal{G}(\sigma_{a,j,x_i})$ to be the set of states spanned by five standard deviations from the mean. If the raw data for two symbols is found to be the same by a two sample Kolmogorov-Smirnov test [20], the two symbols are merged into one symbol. The set of symbols referring to a single state variable $x_i$ is $\Sigma_{x_i} = \{\sigma_{a,j,x_i} | a \in A, j \in \{1, \ldots, k(a)\}\}$. The set of all symbols is $\Sigma = \bigcup_{x_i \in X} \Sigma_{x_i}$. In Figure 1B, Eff$^1(a_1)$ results in two symbols, $\sigma_{a_1,1,x_1}$ and $\sigma_{a_1,1,x_2}$, because eff-mask$^1(a_1) = \{\top, \top\}$. Only one symbol, $\sigma_{a_1,2,x_1}$ is generated from Eff$^2(a_1)$ as eff-mask$^2(a_1) = \{\top, \bot\}$.

### 2.2 Linear Temporal Logic (LTL)

Let $AP$ be a set of atomic propositions and $\pi \in AP$ be a Boolean variable. A formula in Linear Temporal Logic (LTL) [21] is constructed as:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where negation ($\neg$,"not") and disjunction ($\vee$,"or") are Boolean operators and $\bigcirc$ ("next") and $\mathcal{U}$ ("until") are temporal operators. True is defined as $\top = \varphi \vee \neg\varphi$ and False as $\bot = \neg\top$. Given these operators one can derive conjunction ($\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$), implication ($\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$), equivalence ($\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$), eventually ($\Diamond\varphi \equiv \top\mathcal{U}\varphi$), and always ($\Box\varphi \equiv \neg\Diamond\neg\varphi$).

The semantics of an LTL formula $\varphi$ are defined over an infinite sequence $w = w_1 w_2 \ldots$[21]. Each $w_i$ corresponds to the set of $\pi$ that are $\top$ at step $i$. We denote that a sequence $w$ satisfies an LTL formula at instance $i$ by $w, i \models \varphi$. Intuitively, $w, i \models \bigcirc\varphi$ if $\varphi$ is $\top$ at step $i+1$, $w, i \models \Box\varphi$ if $\varphi$ holds at every step after and including $i$ in $w$, and $w, i \models \Diamond\varphi$ if $\varphi$ holds at some step on or after $i$ in $w$.

We consider the generalized reactivity(1) (GR(1)) fragment of LTL [22]. Let $AP = \mathcal{X} \cup \mathcal{Y}$ be the set of atomic propositions, where $\mathcal{X}$ is the state of the world as represented by $\Sigma$ and additional user defined symbols $E$, and $\mathcal{Y}$ refers to the activation of robot skills, $A$. In GR(1), formulas are of the form:

$$\varphi = (\varphi_e \rightarrow \varphi_s),\ \varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e,\ \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s \tag{1}$$

where $\varphi_e$ are assumptions about the environment's behavior and $\varphi_s$ are guarantees for the robot, also referred to as the system, and:

- $\varphi_i^e$ and $\varphi_i^s$ are constraints on the initial environment and system states, respectively.
- $\varphi_t^e$ and $\varphi_t^s$ are environment and system safety constraints, respectively, that must always be satisfied, of the form $\bigwedge_j \Box\phi_j$ where $\phi_j$ are Boolean formulas over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}' \cup \mathcal{Y}'$. Here $\mathcal{X}'$ and $\mathcal{Y}'$ represent variables in $\mathcal{X}$ and $\mathcal{Y}$ prefixed with $\bigcirc$.

[1] Note that $\sigma$ are only generated from effect sets in Konidaris, et al. [1], future work will consider generating $\sigma$ from precondition sets as well.

- $\varphi_g^e$ and $\varphi_g^s$ are the liveness guarantees of the environment and system, respectively. These correspond to goals that should be repeatedly achieved and are of the form $\bigwedge_k \square\Diamond\psi_k$ where $\psi_k$ are Boolean formulas over $\mathcal{X} \cup \mathcal{Y}$.

### 2.3 Synthesis and Counter Strategies

Synthesis is the process of creating a finite state controller for the system such that Equation 1 is $\top$ for every execution [22]. This means that either the environment assumptions hold and the system guarantees are satisfied ($\varphi_e$ and $\varphi_s$ are $\top$) or the environment assumptions are not satisfied ($\varphi_e$ is $\bot$). We say that Equation 1 is realizable if such a controller exists and unrealizable otherwise. We define a controller as $F = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta, L)$, where:

- $\mathcal{X}$ and $\mathcal{Y}$ are the environment and system propositions, respectively, defined above
- $Q$ is the set of states
- $Q_0 \subseteq Q$ is the set of initial states
- $\delta : Q \times 2^{\mathcal{X}} \to Q$ is the transition function
- $L : Q \to 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ is a labeling function that returns which $\mathcal{X}$ and $\mathcal{Y}$ are $\top$

for $q \in Q,\ x \in 2^{\mathcal{X}}$, $q' = \delta(q, x)$ is the next state and $L(q)$ is the set of symbols that are $\top$ in state $q$. Here, $\delta$ depends on $\mathcal{X}$ as the system reacts to the environment state.

If Equation 1 is unrealizable, meaning that there does not exist a controller $F$, the synthesis algorithm can provide a *counter-strategy* that represents the behavior of the environment that will cause the system to fail [17,16]. We define such a strategy as $F_{cs} = (\mathcal{X}, \mathcal{Y}, Q, Q_0, Q_{nt}, \delta_{cs}, L_t, L_{nt})$, where $\mathcal{X}, \mathcal{Y}, Q, Q_0$ are the same as in $F$ and:

- $Q_{nt} \subseteq Q$ is the set of states from which there are no outgoing transitions
- $\delta_{cs} : Q \setminus Q_{nt} \times 2^{\mathcal{X}} \to Q$ is the transition function
- $L_t : Q \setminus Q_{nt} \to 2^{\mathcal{X}} \times 2^{\mathcal{Y}}$ is the labeling function for states with outgoing transitions
- $L_{nt} : Q_{nt} \to 2^{\mathcal{X}}$ is the labeling function for states with no outgoing transitions. The system has no valid transitions from $Q_{nt}$, so only $\mathcal{X}$ is needed to label $Q_{nt}$.

In section 5, we use the states with no outgoing transitions, $Q_{nt}$, to narrow the search for skills to repair unrealizable specifications.

## 3 Problem Formulation

Our goal is to automatically encode the capabilities of the robot in a Linear Temporal Logic (LTL) formula and find a strategy for a reactive high-level task. If no strategy can be found, we find additional skills that would allow the robot to complete the task.

**Problem 1:** Given a set of skills $A$, automatically abstract and encode the capabilities of the robot in an LTL formula, $\varphi_{\text{fixed}}$. Allow a user to specify a reactive high-level task and find a strategy to fulfill it.

**Problem 2:** Given an unrealizable specification $\varphi_{\text{unreal}}$ and counter-strategy $F_{cs}$, find an additional skill, $a_{\text{new}}$, such that constructing $\varphi_{\text{fixed}}$ with $A \cup a_{\text{new}}$ makes the specification $\varphi_{\text{unreal}}$ realizable, if such a skill exists.

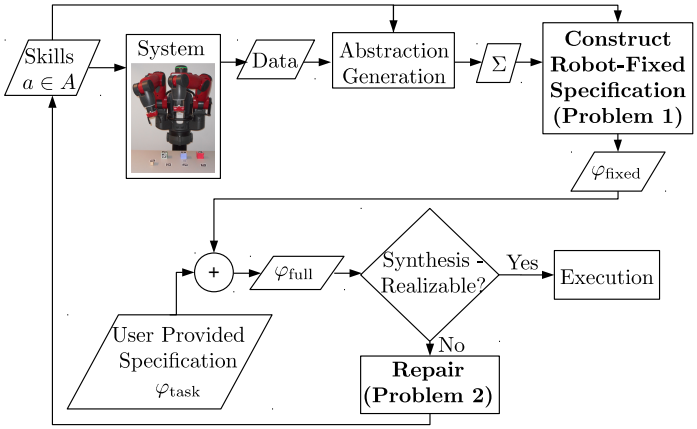


Fig. 2: Framework for automatically encoding robot capabilities, executing tasks, and repairing unrealizable tasks. Novel contributions are in bold.

## 4 Symbols, Specifications, and Synthesis

In Problem 1, we automatically encode the robot's capabilities in $\varphi_{\text{fixed}}$ using learned abstractions. The robot-fixed specification, $\varphi_{\text{fixed}}$, can be reused for different tasks performed by the same robot. The user then writes the task specific specification, $\varphi_{\text{task}}$, over $\Sigma \cup E \cup A$, which is combined with $\varphi_{\text{fixed}}$ to create $\varphi_{\text{full}}$. The symbols in $\Sigma$ are learned from the low-level sensor information [1], $E$ are additional user defined environment propositions, and $A$ are the skills of the robot. Using a synthesis algorithm, such as Slugs [23], we either find a strategy for accomplishing $\varphi_{\text{full}}$ or a counter-strategy. An overview of the framework is depicted in Figure 2.

### 4.1 Robot-fixed Specification ($\varphi_{\text{fixed}}$)

Given a set of skills $A$, we first create symbols $\sigma \in \Sigma$, representing the effects of $a \in A$ [1]. We slightly abuse notation and use $a$ as a proposition that is $\top$ when the skill $a$ is active, and $\bot$ otherwise.

The robot-fixed specification ($\varphi_{\text{fixed}}$) is composed of the system safety ($\varphi^{\text{s}}_{\text{t,fixed}} = \varphi^{\text{s}}_{\text{t,pre}} \wedge \varphi^{\text{s}}_{\text{t,mut\_exc}}$) and environment safety ($\varphi^{\text{e}}_{\text{t,fixed}} = \varphi^{\text{e}}_{\text{t,eff}} \wedge \varphi^{\text{e}}_{\text{t,no\_act}}$) specifications. The system safety specification includes constraints on when the system is allowed to perform skills ($\varphi^{\text{s}}_{\text{t,pre}}$) and optionally the mutual exclusion of skills ($\varphi^{\text{s}}_{\text{t,mut\_exc}}$). The environment safety specification includes how each $\sigma$ is allowed to change with the application of a skill ($\varphi^{\text{e}}_{\text{t,eff}}$) and the effect of no skill being performed ($\varphi^{\text{e}}_{\text{t,no\_act}}$).

**System Safety ($\varphi^{\text{s}}_{\text{t,fixed}}$):** For each action, we find all possible combinations of symbols that overlap with the precondition mask and determine which combinations fall within the precondition set [1]. We define $\sigma_{\text{pre}(a)} = \{\sigma_p \in \Sigma_{\text{pre-mask}(a)} | \mathcal{G}(\sigma_p) \subseteq \text{Pre}(a)\}$, where $\Sigma_{\text{pre-mask}(a)} = \prod_{x_i \in X, \text{pre-mask}(a)(i)=\top} \Sigma_{x_i}$. The set $\sigma_{\text{pre}(a)}$ contains all the combinations of $\sigma$ that satisfy the precondition of $a$. When none of the preconditions in $\sigma_{\text{pre}(a)}$ are satisfied, the robot is not allowed to perform $a$. We encode this as:

$$\varphi^{\text{s}}_{\text{t,pre}} = \bigwedge_{a \in A} \square \left[ \neg \left( \bigvee_{\sigma_p \in \sigma_{\text{pre}(a)}} \left( \bigwedge_{\sigma \in \sigma_p} \sigma \right) \right) \rightarrow \neg a \right]. \tag{2}$$

Equation 2 states that skill $a$ cannot be executed if no combinations of symbols $\sigma_p \in \sigma_{\text{pre}(a)}$, are $\top$. Therefore, only when some combination of symbols $\sigma_p \in \sigma_{\text{pre}(a)}$ is $\top$ can $a$ be performed. In Figure 1, $\sigma_{\text{pre}(a_2)} = \{\{\sigma_{a_1,1,x_1}\}, \{\sigma_{a_1,2,x_1}\}\}$.

We can encode mutual exclusion of skills in $\varphi^{\text{s}}_{\text{t,mut\_exc}}$. In the examples presented, skills are mutually exclusive, although in general they do not have to be.

**Environment Safety ($\varphi^{\text{e}}_{\text{t,fixed}}$):** To encode a skill's nondeterministic effects, we consider the skill outcome as part of the uncontrolled environment.

We denote the symbols which become $\top$ with the application of a skill $a$ as $\sigma^{\top}_{\text{eff}^j(a)} = \cup_i \, \sigma_{a,j,x_i}[1]$. In Figure 1, $\sigma^{\top}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,2,x_1}\}$.

When $a$ is applied, symbols whose grounding sets do not overlap with those in $\sigma^{\top}_{\text{eff}^j(a)}$ become $\perp$ due to mutual exclusion. We denote this set of symbols as $\sigma^{\perp}_{\text{eff}^j(a)} = \cup_{x_i | \text{eff-mask}^j(a)(i)=\top} \{\sigma \in \Sigma_{x_i} | \; \mathcal{G}(\sigma) \cap \mathcal{G}(\sigma_{a,j,x_i}) = \varnothing\}$. In Figure 1, $\sigma^{\perp}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,1,x_1}, \sigma_{a_2,1,x_1}\}$.

When performing synthesis [2], if a symbol is not constrained, it can be set to any value. We must therefore consider the "frame problem" [24] and constrain symbols that are not modified by the current skill to stay the same. The set $\sigma^{\text{stay}}_{\text{eff}^j(a)} = \cup_{x_i | \text{eff-mask}^j(a)(i)=\perp} \Sigma_{x_i}$ contains the $\sigma$ not modified by skill $a$ in the $j^{th}$ outcome. In Figure 1, because $x_2$ is not modified in effect 2 of $a_1$, $\sigma^{\text{stay}}_{\text{eff}^2(a_1)} = \{\sigma_{a_1,1,x_2}, \sigma_{a_2,1,x_2}\}$.

We encode how the truth values for $\sigma$ can change when a skill is applied in $\varphi^{\text{e}}_{\text{t,eff}}$:

$$\begin{aligned} \varphi^{\text{e}}_{\text{t,eff}} = \bigwedge_{a \in A} \square \Bigg[ a \rightarrow \bigvee_{j \in \{1,\ldots,k(a)\}} \Bigg( \Bigg( \bigwedge_{\sigma \in \sigma^{\top}_{\text{eff}^j(a)}} \bigcirc \sigma \Bigg) \wedge \\ \Bigg( \bigwedge_{\sigma \in \sigma^{\perp}_{\text{eff}^j(a)}} \bigcirc \neg \sigma \Bigg) \wedge \Bigg( \bigwedge_{\sigma \in \sigma^{\text{stay}}_{\text{eff}^j(a)}} (\sigma \leftrightarrow \bigcirc \sigma) \Bigg) \Bigg) \Bigg]. \end{aligned} \tag{3}$$

Equation 3 states that when skill $a$ is performed, it leads to one of $j$ nondeterministic outcomes with $\sigma \in \sigma^{\top}_{\text{eff}^j(a)}$ becoming $\top$, $\sigma \in \sigma^{\perp}_{\text{eff}^j(a)}$ becoming $\perp$, and $\sigma \in \sigma^{\text{stay}}_{\text{eff}^j(a)}$ not changing. Symbols whose grounding sets overlap with those in $\sigma^{\top}_{\text{eff}^j(a)}$ and are therefore not in $\sigma^{\top}_{\text{eff}^j(a)}$, $\sigma^{\perp}_{\text{eff}^j(a)}$, or $\sigma^{\text{stay}}_{\text{eff}^j(a)}$ are not constrained. In the examples presented in this work, there are no such symbols.

When no skill is performed, no $\sigma$ can change as encoded in $\varphi^{\text{e}}_{\text{t,no\_act}}$:

$$\varphi^{\text{e}}_{\text{t,no\_act}} = \square \left[ \left( \bigwedge_{a \in A} \neg a \right) \rightarrow \left( \bigwedge_{\sigma \in \Sigma} (\sigma \leftrightarrow \bigcirc \sigma) \right) \right]. \tag{4}$$

### 4.2 Task-Specific Specification, Synthesis, and Execution

The user writes the task-specific specification, $\varphi_{\text{task}}$, which may include adding environment propositions $\pi_{env} \in E$. The task-specific specification includes the initial state of the system, initial state of the environment, system liveness, and environment liveness in $\varphi^{\text{s}}_{\text{i,task}}$, $\varphi^{\text{e}}_{\text{i,task}}$, $\varphi^{\text{s}}_{\text{g,task}}$, and $\varphi^{\text{e}}_{\text{g,task}}$, respectively. Additional system safety constraints can be added in $\varphi^{\text{s}}_{\text{t,task}}$. We give examples of these formulas in Section 6.

The full specification is then written as:

$$\varphi_{\text{full}} = \varphi^{\text{e}}_{\text{i,task}} \wedge \varphi^{\text{e}}_{\text{t,fixed}} \wedge \varphi^{\text{e}}_{\text{g,task}} \rightarrow \varphi^{\text{s}}_{\text{i,task}} \wedge \varphi^{\text{s}}_{\text{t,fixed}} \wedge \varphi^{\text{s}}_{\text{t,task}} \wedge \varphi^{\text{s}}_{\text{g,task}} \tag{5}$$

We generate a strategy for satisfying $\varphi_{\text{full}}$ using a synthesis algorithm, such as Slugs [23]. If $\varphi_{\text{full}}$ is realizable, the resulting automaton $F = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta, L)$, where $\mathcal{X} = \Sigma \cup E$ and $\mathcal{Y} = A$, is used to control the robot.

To assist the user in writing $\varphi_{\text{task}}$, we visualize the grounding of the symbols and combinations of symbols. Figures 4E and 4F show examples of individual symbol groundings. Figures 4A-D, 5C, and 5F visualize the combination of multiple symbols. To visualize each combination of symbols, 10 samples are drawn from the intersection of the grounding sets of the symbols.

## 5 Specification Repair through Finding Additional Skills

We address Problem 2 by searching for a skill, $a_{\text{new}}$, to make an unrealizable task realizable when $\varphi_{\text{fixed}}$ is constructed with $A \cup \{a_{\text{new}}\}$. We assume the robot has all the symbols it needs to define the task and that our new skill will consist of a precondition set and effect mask we have already seen, reducing the search space of the new skill.

We leverage the structure of $F_{cs}$ to focus the repair process. The counter strategy, $F_{cs}$, contains the environment behaviors that make a specification unrealizable. In general, a generalized reactivity(1) (GR(1)) specification is unrealizable either because the robot violates safety constraints or gets stuck in a loop when trying to satisfy its liveness goals. When the robot can only satisfy its liveness goals by violating safety constraints, the counter strategy contains states with no successors (i.e. $Q_{nt} \neq \varnothing$). We find the skills that lead to these states, and use their precondition sets to narrow the search space for $a_{\text{new}}$. We generate new effect sets, based on existing effect masks, and combine them with the precondition sets to create new skills.

Algorithm 1 shows our procedure for repairing unrealizable specifications. In line 2, we create new effect sets based on existing effect masks. We only consider effect masks that already exist, making the assumption that the new skills will change similar states as current skills. For each existing effect mask, we find all the state variables that are changed. We then compute all possible combinations of $\sigma_{a,j,x_i}$ that ground to those state variables, regardless of which skill they were originally generated from. In line 3, we find $A_{test}$, the set of skills whose preconditions were satisfied that lead to states with no outgoing transitions. In lines 5-8, we construct new skills, $a_{\text{new}}$, each one consisting of the precondition set of a skill in $A_{test}$ and a new effect set found in $\Sigma^+$. In line 9, we check that the $a_{\text{new}}$ under consideration does not already exist. We then construct $\varphi_{\text{fixed}}$ with the proposed skill $a_{\text{new}}$ in line 10 and synthesize a strategy for $\varphi_{\text{full}}$ in line 11. If

**Algorithm 1** Specification Repair

1: **procedure** REPAIR($F_{cs}, A, \Sigma$)
2: $\Sigma^+ \leftarrow \bigcup_{a\in A, j\in\{1,\ldots,k(a)\}} \prod_{x_i\in X, \text{eff-mask}^j(a)(i)=\top} \Sigma_{x_i}$
3: $A_{test} \leftarrow \{a \in A | \exists \sigma_p \in \sigma_{\text{pre}(a)},\ q \in Q,\ x \in 2^{\mathcal{X}} \text{s.t. } \delta(q,x) \in Q_{nt},\ \sigma_p \in \mathcal{L}_t(q)\}$
4: $A_n \leftarrow \varnothing$
5: **for** $a \in A_{test}$ **do**
6: $\sigma_{\text{pre}(a_{\text{new}})} \leftarrow \sigma_{\text{pre}(a)}$
7: **for** $\sigma_{test} \in \Sigma^+$ **do**
8: $\sigma^{\top}_{\text{eff}(a_{\text{new}})} \leftarrow \sigma_{test}$
9: **if** $\nexists\ a \in A,$ s.t. $\sigma_{\text{pre}(a_{\text{new}})} = \sigma_{\text{pre}(a)}$ and $\sigma^{\top}_{\text{eff}(a_{\text{new}})} = \sigma^{\top}_{\text{eff}(a)}$ **then**
10: Write $\varphi_{\text{fixed}}$ with $A \cup a_{\text{new}}$
11: Synthesize $\varphi_{\text{full}}$ with $\varphi_{\text{fixed}}$ and $\varphi_{\text{task}}$
12: **if** Realizable **then**
13: $A_n \leftarrow A_n \cup (\sigma_{\text{pre}(a_{\text{new}})},\ \sigma^{\top}_{\text{eff}(a_{\text{new}})})$
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **return** $A_n$
19: **end procedure**

the new specification is realizable, $a_{\text{new}}$ is added to the set of suggested skills in line 13. All $a_{\text{new}}$ which make $\varphi_{\text{full}}$ realizable are returned to the user, enabling them to select the skill they deem easiest to physically implement.

## 6 Robot Demonstrations

We use the following example with a Baxter to demonstrate the process of automatically creating $\varphi_{\text{fixed}}$, writing and executing task specifications, and the repair process.

**Example 2**: Consider a robot manipulating blocks. The robot has skills that enable it to move the blocks between different locations as shown in Figure 3A. In this domain, the location AT is one block height above location A and location C is slightly elevated. Skills are referenced by the starting and ending locations in the format $a_{\texttt{start-to-end}}$. For example, the skill moving a block from location D to A is referred to as $a_{\texttt{d-to-a}}$. One skill takes blocks from locations A and AT to D and another takes blocks from A and AT to E. There is no robot skill which attempts to place a block directly in location G. The skill moving block 3 from location F to C, $a_{\texttt{f-to-c}}$, may result in block 3 ending up in either location C or G. Likewise, the skill moving block 3 from location G to C, $a_{\texttt{g-to-c}}$, may result in block 3 ending up in C or G. In this scenario, the nondeterminism was due to location C being elevated and block 3 having a chance of falling into G. Given these skills, we want the robot to configure the blocks as shown in Figure 5A when a user defined environmental variable $switch = \top$ and as shown in Figure 5D

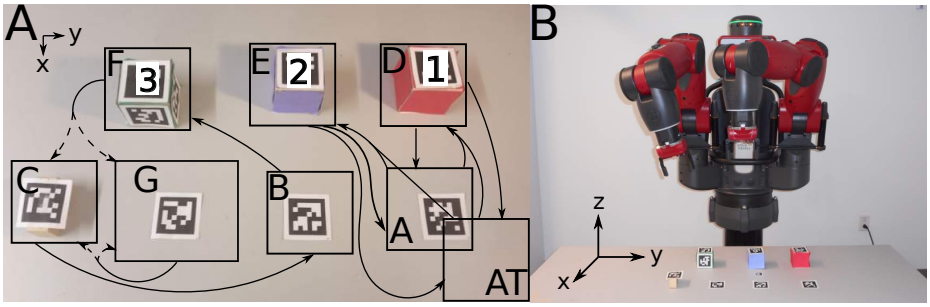


Fig. 3: (A) Example 2: Block manipulation domain. The arrows show the skills given to the robots to move blocks between lettered locations. Dashed arrows represent skills with nondeterministic outcomes. One skill takes blocks from A and AT to D and another from A and AT to E. (B) Initial setup of the Baxter robot and blocks.

when $switch = \perp$. The robot must react to the variable $switch$ and the outcome of skills moving block 3 to location C.

### 6.1 Environment Setup

Figure 3 shows the environment, a table with seven locations and three blocks on it. The blocks are 6.4 cm cubes. AprilTags [25] denote each location and block. The origin of the coordinate system is in the Baxter robot, with the the $x$ and $y$ positions of the AprilTags (in meters) at approximately (0.7,.0.2), (0.7,0), (0.7,-.1), (0.4,0.2), (0.4,0), and (0.4,-.2) for locations A, B, D, E, F, and G, respectively, and $z$ = -0.12. The location of region C was on top of a wooden block at approximately (0.7,-0.3,-.08).

### 6.2 Skills

We defined 10 robot skills, each of which involved moving a block from one location to another, shown by arrows in Figure 3A. Blocks 1 and 2 (red and blue, respectively) were manipulated by the left arm of the Baxter and block 3 (green) by the right arm. The robot was allowed to execute each skill when the center of the block is within 6 cm of the AprilTag representing the starting location, with the exception of location G, where the distance is 15 cm.

The sensor data are the coordinates of the AprilTags, identified by cameras at the ends of the Baxter's arms. Each skill consists of a robot arm moving from the resting position, shown in Figure 3B, to above the block, picking up the block, moving above the goal location, releasing the block, and returning to the resting position. We use MoveIt [26] to generate the commands to move the robot arms between the waypoints that make up the skill.

### 6.3 Symbol learning

The continuous state space is the $x$, $y$, and $z$ positions of each of the blocks. We collected position data from between 60 and 137 executions of each skill.

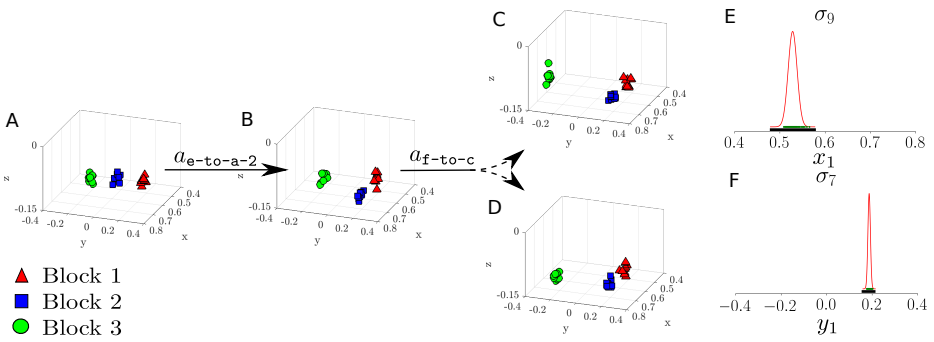


Fig. 4: Visualization of symbol combinations (A): $\sigma_9 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_{11} \wedge \sigma_{13} \wedge \sigma_{12} \wedge \sigma_{17} \wedge \sigma_{18} \wedge \sigma_{16}$, (B): $\sigma_9 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_{12} \wedge \sigma_{17} \wedge \sigma_{18} \wedge \sigma_{16}$, (C): $\sigma_9 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_{12} \wedge \sigma_0 \wedge \sigma_1 \wedge \sigma_2$, (D): $\sigma_9 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_{12} \wedge \sigma_0 \wedge \sigma_{18} \wedge \sigma_{16}$. All other symbols were $\perp$. Ten samples were drawn from the intersection of the grounding sets of each symbol combination. Possible transitions are shown between the subfigures, corresponding to transitions in Equations 7 and 8. Applying skill $a_{\texttt{e-to-a-2}}$ to (A) results in (B). Applying skill $a_{\texttt{f-to-c}}$ in (B) results in (C) or (D). Examples of symbol groundings are shown in (E) and (F) in black. The raw data is shown in green and the Gaussian fit to it in red.

The symbol learning process created 19 symbols after automatically merging symbols with similar effect sets, as found by a two sample Kolmogorov-Smirnov test [20]. Block 3 has three distinct sets of values in the $y$ dimension, corresponding to locations C, B, and G, therefore the framework created three symbols referring to the $y_3$ state variable. For every other block and dimension, there are only two distinct sets of values, and therefore two symbols.

The learning process created 20 abstract skills from the 10 initial robot skills. This is due to the learning process partitioning skills that could move either block 1 or 2 into separate skills [1]. For example, skill $a_{\texttt{e-to-a}}$ was partitioned into two skills, $a_{\texttt{e-to-a-1}}$ and $a_{\texttt{e-to-a-2}}$. Skills $a_{\texttt{d-to-a}}$, $a_{\texttt{e-to-at}}$, and $a_{\texttt{d-to-at}}$ were also partitioned into two skills. Skills $a_{\texttt{a-to-d}}$, and $a_{\texttt{a-to-e}}$ were partitioned into four skills each, corresponding to blocks 1 and 2 starting in either A or AT. Skills involving block 3 were not partitioned.

### 6.4 Robot-fixed Specification ($\varphi_{\text{fixed}}$)

We automatically encoded the symbols $\{\sigma_0, \ldots, \sigma_{18}\} \in \Sigma$ and skills $\{a_{\texttt{f-to-c}}, \ldots, a_{\texttt{d-to-at-2}}\} \in A$ in $\varphi_{\text{fixed}}$. In Equation 6, part of the robot-fixed system safety formula $\varphi^{\text{s}}_{\text{t,fixed}}$ is shown. Part of the environment safety formula $\varphi^{\text{e}}_{\text{t,eff}}$ is shown in Equations 7 and 8. Figure 4A-D visualize the result of applying skills $a_{\texttt{e-to-a-2}}$ and $a_{\texttt{f-to-c}}$.

The precondition requirements of $a_{\texttt{c-to-b}}$ are encoded in $\varphi^{\text{s}}_{\text{t,fixed}}$ as:

$$\square(\neg\sigma_1 \rightarrow \neg a_{\texttt{c-to-b}}) \tag{6}$$

Based on the data the robot has seen, it determines that it only needs to consider the value of $y_3$ in deciding if skill $a_{\texttt{c-to-b}}$ can be performed. There is only one symbol

falling inside the precondition set so $\sigma_{\text{pre}(a_{\texttt{c-to-b}})} = \{\{\sigma_1\}\}$. Therefore, equation 6 states that if $\sigma_1$ is not $\top$, i.e. block 3 is not at approximately $y = -0.3$m, skill $a_{\texttt{c-to-b}}$ can not be applied.

The part of $\varphi^{\text{e}}_{\text{t,fixed}}$ referring to the effect of skill $a_{\texttt{e-to-a-2}}$ is:

$$\square(\, a_{\texttt{e-to-a-2}} \rightarrow (\bigcirc(\sigma_3 \wedge \sigma_4) \wedge \bigcirc(\neg\sigma_{11} \wedge \neg\sigma_{13}) \bigwedge_{\sigma \in \sigma^{\text{stay}}_{\text{eff}^1(a_{\texttt{e-to-a-2}})}} (\sigma \leftrightarrow \bigcirc\sigma))) \tag{7}$$

where $\sigma^{\text{stay}}_{\text{eff}^1(a_{\texttt{e-to-a-2}})} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_5, \ldots, \sigma_{10}, \sigma_{12}, \sigma_{14}, \ldots, \sigma_{18}\}$. This corresponds to block 2 moving from location E to A and blocks 1 and 3 not moving. A potential outcome of applying skill $a_{\texttt{e-to-a-2}}$ is visualized in Figure 4B.

The part of $\varphi^{\text{e}}_{\text{t,fixed}}$ referring to the nondeterministic effects of skill $a_{\texttt{f-to-c}}$ is:

$$\begin{aligned}\square((a_{\texttt{f-to-c}}) \rightarrow ((\bigcirc\sigma_0 \wedge \bigcirc\neg\sigma_{17} \bigwedge_{\sigma \in \sigma^{\text{stay}}_{\text{eff}^1(a_{\texttt{f-to-c}})}} (\sigma \leftrightarrow \bigcirc\sigma)) \vee \\ (\bigcirc(\sigma_0 \wedge \sigma_1 \wedge \sigma_2) \wedge \bigcirc(\neg\sigma_{15} \wedge \neg\sigma_{17} \wedge \neg\sigma_{16} \wedge \neg\sigma_{18}) \bigwedge_{\sigma \in \sigma^{\text{stay}}_{\text{eff}^2(a_{\texttt{f-to-c}})}} (\sigma \leftrightarrow \bigcirc\sigma))))\end{aligned} \tag{8}$$

where $\sigma^{\text{stay}}_{\text{eff}^1(a_{\texttt{f-to-c}})} = \{\sigma_1, \ldots, \sigma_{16}, \sigma_{18}\}$ and $\sigma^{\text{stay}}_{\text{eff}^2(a_{\texttt{f-to-c}})} = \{\sigma_3, \ldots, \sigma_{14}\}$. Equation 8 encodes that when skill $a_{\texttt{f-to-c}}$ is applied, either $\sigma_0$ becomes $\top$ and $\sigma_{17}$ becomes $\bot$ with symbols in $\sigma^{\text{stay}}_{\text{eff}^1(a_{\texttt{f-to-c}})}$ not changing (block 3 ends in G), or $\sigma_0$, $\sigma_1$, and $\sigma_2$ become $\top$ and $\sigma_{15}$, $\sigma_{17}$, $\sigma_{16}$, and $\sigma_{18}$ become $\bot$ with symbols in $\sigma^{\text{stay}}_{\text{eff}^2(a_{\texttt{f-to-c}})}$ not changing (block 3 ends in C). This is visualized in Figures 4C and 4D.

### 6.5 Task Specification ($\varphi_{\text{task}}$)

We introduce an additional environment variable $switch \in E$. The task liveness specification shown in Figure 5 is:

$$\square\lozenge(\text{Switch} \rightarrow (\sigma_6 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_5 \wedge \sigma_{17} \wedge \sigma_{18} \wedge \sigma_{16})) \tag{9}$$

$$\square\lozenge(\neg\text{Switch} \rightarrow (\sigma_6 \wedge \sigma_7 \wedge \sigma_8 \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_{12} \wedge \sigma_0 \wedge \sigma_1 \wedge \sigma_2)) \tag{10}$$

Equation 9 encodes that when $switch = \top$, block 1 should eventually be in A, block 2 in AT, and block 3 in F (Figure 5A-C). Equation 10 encodes that when $switch = \bot$, block 1 should be in AT, block 2 in A, and block 3 in C (Figure 5D-F).

We include a fairness assumption on the environment (Equation 11) that block 3 will eventually be placed in location C when $a_{\texttt{g-to-c}}$ is applied. Without this, the specification is unrealizable because in the worst case, skill $a_{\texttt{g-to-c}}$ always results block 3 ending in G.

$$\varphi^{\text{e}}_{\text{g,task}} = \square\lozenge(a_{\texttt{g-to-c}} \rightarrow (\sigma_0 \wedge \sigma_1 \wedge \sigma_2)) \tag{11}$$

$$\square\lozenge(\text{Switch} \rightarrow (\sigma_6 \wedge \sigma_7 \wedge \sigma_{10} \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_5 \wedge \sigma_{17} \wedge \sigma_{18} \wedge \sigma_{16}))$$


$$\square\lozenge(\neg\text{Switch} \rightarrow (\sigma_6 \wedge \sigma_7 \wedge \sigma_8 \wedge \sigma_3 \wedge \sigma_4 \wedge \sigma_{12} \wedge \sigma_0 \wedge \sigma_1 \wedge \sigma_2))$$


Fig. 5: (A,B,C) Where the blocks should be when $switch = \top$. (D,E,F) Where the blocks should be when $switch = \bot$. (A,D) As seen by an observer. (B,E) As seen by the Baxter's left hand camera. (C,F) As represented by sampling 10 points from the grounding sets of the $\top$ symbols. Above each row is the liveness formula.

### 6.6 Synthesis and Execution

We generated an automaton $F$, with 58 states, containing a strategy to fulfill the specification $\varphi_{\text{full}}$ using Slugs [23] in 12 seconds on a desktop machine running Ubuntu 14.04 with 8 GB RAM. We controlled the Baxter using $F$ and provided the $switch$ input through a user interface. We sampled the current state $x \in X$ to find out which symbols were $\top$. A symbol $\sigma_{a,j,x_i}$ was $\top$ if the state was in the grounding set for the symbol, $\mathcal{G}(\sigma_{a,j,x_i})$. All other symbols were $\bot$. We show an example execution of $F$ in Figure 6.

### 6.7 Repair

We demonstrate the repair process by finding skill suggestions for four unrealizable specifications. For two specifications we find $A_{test} \neq \varnothing$, allowing us to narrow the search space for new skills to those with the same preconditions as $a \in A_{test}$. We find $A_{test} = \varnothing$ for the other two specifications, requiring us to perform an exhaustive search for new skills over all preconditions sets.

The two unrealizable specifications for which we can narrow the search space of possible skills have the same $\varphi_{\text{fixed}}$ and $\varphi_{\text{task}}$ as in Section 6.4 and 6.5, with the addition of $\varphi^{\text{s}}_{\text{t,task}}$, as shown in Figure 7 and described below.

**Unrealizable Specification 1**: In Figure 7A, we show the added constraint that block 3 never be in location B, $\varphi^{\text{s}}_{\text{t,task}} = \square\neg \bigcirc (\sigma_0 \wedge \sigma_{15})$. This type of scenario could occur if there was an obstacle in location B.

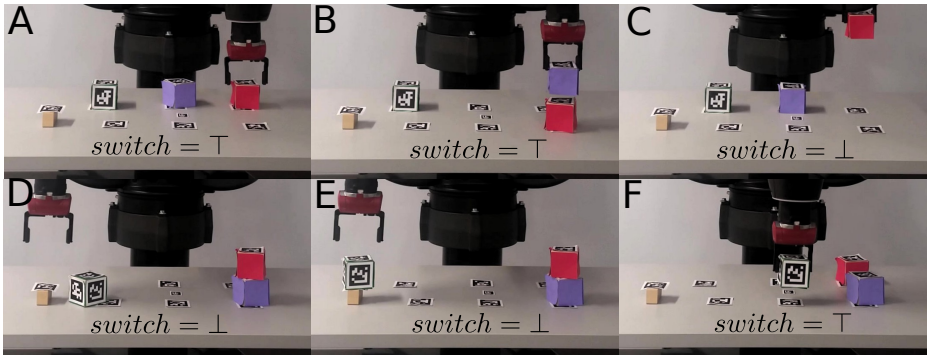


Fig. 6: The Baxter reacting to the value of $switch$ (shown at the bottom of each figure) and the nondeterministic effects of its skills to fulfill the task in Equations 9 and 10.

**Unrealizable Specification 2**: In Figure 7B, we show the added constraint that the robot never use skill $a_{\texttt{c-to-b}}$, $\varphi^{s}_{\text{t,task}} = \square\neg a_{\texttt{c-to-b}}$. This type of scenario could occur if a motor enabling skill $a_{\texttt{c-to-b}}$ was damaged and the skill could not be performed.

For both Unrealizable Specifications 1 and 2, the repair process found $A_{test} = \{a_{\texttt{c-to-b}}\}$, corresponding to the precondition that block 3 be in location C. The repair process searched through 63 possible skills and tested 62 with Slugs to find six skill suggestions for Unrealizable Specification 1 and three skill suggestions for Unrealizable Specification 2 in 9.9 minutes and 23.1 minutes, respectively. Some suggestions were not physically possible, such as a suggestion with $\sigma_{\text{pre}(a_{\text{new}})} = \{\{\sigma_1\}\}$ and $\sigma^{\top}_{\text{eff}(a_{\text{new}})} = \{\sigma_{17}, \sigma_1, \sigma_{16}\}$, corresponding to moving block 3 to the $x$ position of location F and the $y$ and $z$ position of to location C, which would leave the block floating in the air. One skill suggestion, with $\sigma_{\text{pre}(a_{\text{new}})} = \{\{\sigma_1\}\}$ and $\sigma^{\top}_{\text{eff}(a_{\text{new}})} = \{\sigma_{17}, \sigma_{18}, \sigma_{16}\}$, corresponding to moving block 3 from location C to F, is physically possible. When this skill is added to Unrealizable Specifications 1 and 2, the task is realizable.

**Unrealizable Specification 3**: We removed skill $a_{\texttt{c-to-b}}$ from $A$ before writing the specification, using the same set of symbols $\Sigma$ as in Section 6.4. The user defined task was the same as in Equations 9 and 10.

**Unrealizable Specification 4**: We removed all data pertaining to skill $a_{\texttt{c-to-b}}$ before the symbol generation process. This resulted in a different set of symbols, $\Sigma$. The user defined task was the same as represented in Figure 5. There were no longer symbols corresponding to block 3 being in location B, as symbols are only generated from effect sets, so the subscripts in Equations 9 and 10 were different.

For both Unrealizable Specification 3 and 4, the repair process found $A_{test} = \varnothing$, requiring an exhaustive search of the skill space. For Unrealizable Specification 3, the repair process searched through 1197 new skills, tested 1172 of them with Slugs, and found 68 possible new skills in 325 minutes. For Unrealizable Specification 4, the repair process searched through 810 new skills, tested 788 of them, and found 17 possible skills to repair the specification in 111 minutes. The repair process suggested a skill that would move block 3 from both locations C and G to location F for both specifications. With the fairness assumption in Equation 11, this has the same result as giving the

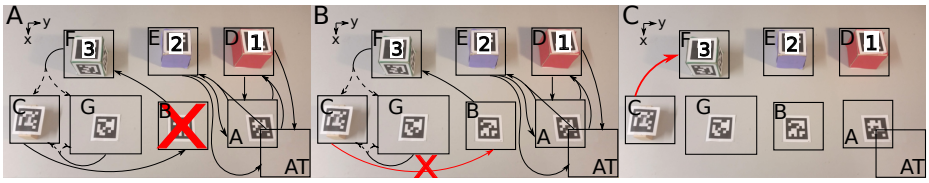


Fig. 7: Additional constraints for: (A) Unrealizable Specification 1 (the block cannot move into location B, $\square\neg\bigcirc(\sigma_0 \wedge \sigma_{15})$), (B) Unrealizable Specification 2 (skill $a_{\texttt{c-to-b}}$ is not allowed). (C) The skill, $a_{\texttt{c-to-f}}$, used to repair Unrealizable Specifications 1-4.

robot a skill moving block 3 from C to F. When we added a skill from C to F, both specifications were realizable.

Figure 7C shows the additional skill $a_{\texttt{c-to-f}}$. We collected data from 57 executions of skill $a_{\texttt{c-to-f}}$ and relearned the symbols, precondition sets, and effect sets. When we rewrote the previously unrealizable specifications after incorporating this new data, all the specifications were realizable.

## 7 Conclusion

We have shown a framework for automatically encoding robot capabilities using abstractions generated from sensor data in Linear Temporal Logic (LTL). We allow a user to specify reactive high-level tasks for the robot to perform, which can involve additional user defined symbols. We provide skill suggestions to repair unrealizable specifications. Our framework is demonstrated on a Baxter robot manipulating blocks.

In future work, we will demonstrate our framework on additional domains, and consider abstract representations that are grounded locally [27]. We will develop methods of refining skill suggestions for repair.

## Acknowledgments

This work is supported by the ONR PERISCOPE MURI award N00014-17-1-2699.

## References

1. Konidaris, G., Kaelbling, L.P., and Lozano-Perez, T. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.
2. Kress-Gazit, H., Lahijanian, M., and Raman, V. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:211–236, 2018.
3. Mazo, M., Davitian, A., and Tabuada, P. PESSOA: A Tool for Embedded Controller Synthesis. In *International Conference on Computer Aided Verification*, pages 566–569, Berlin, Heidelberg, 2010. Springer-Verlag.
4. Finucane, C., Jing, G., and Kress-Gazit, H. LTLMoP: Experimenting with language, Temporal Logic and robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1988–1993, 2010.

5. He, K., Lahijanian, M., Kavraki, L., and Moshe, V. Automated Abstraction of Manipulation Domains for Cost-Based Reactive Synthesis. *IEEE Robotics and Automation Letters*, 2018.
6. Fikes, R.E. and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.
7. Fox, M. and Long, D. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.
8. Ghallab, M., Nau, D., and Traverso, P. *Automated Planning and Acting*. Cambridge University Press, 2016.
9. Yoon, Sungwook and Fern, Alan and Givan, Robert. FF-Replan: A baseline for probabilistic planning. In *ICAPS*, number 7, pages 352–359, 2007.
10. Lahijanian, M., Andersson, S.B., and Belta, C. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, 2012.
11. Wongpiromsarn, T., Topcu, U., and Murray, R.M. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM international conference on Hybrid systems: Computation and Control*, pages 101–110. ACM, 2010.
12. DeCastro, J.A. and Kress-Gazit, H. Nonlinear controller synthesis and automatic workspace partitioning for reactive high-level behaviors. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 225–234. ACM, 2016.
13. Kress-Gazit, H., Fainekos, G.E., and Pappas, G.J. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.
14. He, K., Wells, A., Kavraki, L.E., and Vardi, M.Y. Efficient Symbolic Reactive Synthesis for Finite-Horizon Tasks. pages 8993–8999, 2019.
15. Raman, V. and Kress-Gazit, H. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*, 29(1):94–104, 2013.
16. Chatterjee, K., Henzinger, T.A., and Jobstmann, B. Environment assumptions for synthesis. In *International Conference on Concurrency Theory*, pages 147–161. Springer, 2008.
17. Könighofer, R., Hofferek, G., and Bloem, R. Debugging formal specifications using simple counterstrategies. In *Formal Methods in Computer-Aided Design*, pages 152–159. IEEE, 2009.
18. Alur, R., Moarref, S., and Topcu, U. Counter-strategy guided refinement of GR (1) temporal logic specifications. In *Formal Methods in Computer-Aided Design*, pages 26–33. IEEE, 2013.
19. Li, W., Dworkin, L., and Seshia, S.A. Mining assumptions for synthesis. In *Proceedings of the Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 43–50. IEEE Computer Society, 2011.
20. Hollander, M., Wolfe, D.A., and Chicken, E. *Nonparametric statistical methods*. John Wiley & Sons, Inc., 3rd edition, 2014.
21. Pnueli, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
22. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Saar, Y. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.
23. Ehlers, R. and Raman, V. Slugs: Extensible GR (1) Synthesis. In *International Conference on Computer Aided Verification*, pages 333–339. Springer, 2016.
24. Ghallab, M., Nau, D., and Traverso, P. *Automated Planning: Theory and Practice*. Elsevier, 2004.
25. Olson, E. AprilTag: A robust and flexible visual fiducial system. In *International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, 2011.
26. Sucan, I.A. and Chitta, S. Moveit! http://moveit.ros.org.
27. James, S., Rosman, B., and Konidaris, G. Learning to Plan with Portable Symbols. In *ICML/IJCAI/AAMAS 2018 Workshop on Planning and Learning*, 2018.