Towards Minimal Explanations of Unsynthesizability for High-Level Robot Behaviors

Vasumathi Raman[1] and Hadas Kress-Gazit[2]

***Abstract*— High-level robot control has recently seen the application of formal methods to the automatic synthesis of correct-by-construction controllers from user-defined specifications. When a specification fails to yield a corresponding controller, existing techniques provide feedback on portions of the specification that cause the failure, but at a coarse granularity. This work provides techniques for extracting minimal explanations of such failures. The approach is shown to provide refinement of the feedback on several example specifications.**

I. Introduction

Recent advances in the use of formal methods for robot control have enabled non-expert users to command robots to perform high-level tasks using a specification language instead of programming the robot controller (e.g., [1], [3], [8], [10], [12], [18]). Several approaches automatically synthesize correct-by-construction controllers from a description of the desired robot behavior and assumptions on its operating environment [12], [18]. If an automaton implementing the specification exists, one is returned. However, for specifications that have no implementation (i.e. the specification is *unsynthesizable*), the process of pin-pointing the cause of the problem can be a frustrating and time-consuming process.

There are two ways in which a specification can be unsynthesizable – it is either *unsatisfiable*, in which case the specified robot behavior cannot be achieved in *any* environment, or it is *unrealizable*, in which case there exists an admissible environment (satisfying the specified assumptions) that prevents the robot from achieving its specified behavior. While previous work identified sub-portions of the specification that contribute to the problem [16], [17], the feedback provided was of a coarse granularity, determined by the structure of the specification. The work presented in this paper builds upon these approaches to identify *unrealizable cores* – minimal subsets of the desired robot behavior that cause it to be unrealizable. The analysis makes use of the counterstrategy (the adversarial environment strategy that prevents the robot from succeeding), and covers unsatisfiable specifications as a special case of unrealizability.

II. Preliminaries

The high-level tasks considered in this work involve a robot operating in a known workspace. The robot reacts to events in the environment, which are captured by its sensors, by choosing from a set of actions including moving between adjacent locations. The tasks specified may also include infinitely repeated behaviors such as patrolling a set of locations. Examples of such high-level tasks include search and rescue missions and the DARPA Urban Challenge.

A. Controller Synthesis

Using formal methods to construct a controller for a robot operating in a continuous domain requires a discrete abstraction and a description of the task in a formal specification language. The discrete abstraction in this work consists of a set of propositions $\mathcal{X}$ whose truth value is controlled by the environment and read by the robot's sensors, and a set of action and location propositions $\mathcal{Y}$ controlled by the robot; the set of all propositions $AP = \mathcal{X} \cup \mathcal{Y}$. The value of each $\pi \in AP$ is the abstracted binary state of a low-level black box component. More details on the discrete abstraction used in this work can be found in [12].

The formal language used for high-level specifications in this work is *Linear Temporal Logic (LTL)* [15]. LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following recursive grammar:

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

where $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is a strong "until". Conjunction ($\wedge$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$), "eventually" ($\diamondsuit$) and "always" ($\square$) are derived from these operators. The truth of an LTL formula is evaluated over sequences of truth assignments to the propositions in AP. Informally, formulas $\bigcirc\varphi$, $\square\varphi$ and $\diamondsuit\varphi$ express that φ is true in the next position, every position, and some position in the sequence, respectively; formula $\square\diamondsuit\varphi$ is thus satisfied if φ is true infinitely often. For a formal definition of the semantics, see [6].

The task specifications in this work are expressed as LTL formulas of the form $\varphi = \varphi_e \Rightarrow \varphi_s$, with $\varphi_p = \varphi_p^i \wedge \varphi_p^t \wedge \varphi_p^g$, where φ_p^i, φ_p^t and φ_p^g for $p \in \{e, s\}$ represent the initial conditions, transition relation and goals for the environment (e) and the robot (s) respectively. LTL is appropriate for specifying robotic behaviors because it provides the ability to describe changes in the truth values of propositions over time. To allow users who may be unfamiliar with LTL to define specifications, tools like LTLMoP [7] include a parser that automatically translates English sentences from a pre-defined grammar into LTL formulas. There are two primary types of properties allowed in a specification – *safety*

*This work was supported by NSF CAREER CNS-0953365, NSF EXCAPE and DARPA N66001-12-1-4250.

[1]V. Raman is with the Department of Computer Science, Cornell University, Ithaca, NY 14853, USA vraman at cs.cornell.edu

[2]H. Kress-Gazit is with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA hadaskg at cornell.edu

properties, which guarantee that "something bad never happens", and *liveness* properties, which state that "something good (eventually) happens". These correspond naturally to LTL formulas with operators "always" ($\Box$) and "eventually" ($\Diamond$). Formulas φ_s^t and φ_e^t above are referred to as *safety* formulas, and consist of a conjunction of formulas of the form $\Box A_i$, where each A_i is a boolean formula over AP and $\bigcirc AP = \{\bigcirc \pi \mid \pi \in AP\}$. Conversely, φ_g^e and φ_g^s are called *liveness* formulas, and consist of conjunctions of formulas $\Box\Diamond B_i$, where each B_i is a boolean formula over AP.

An LTL formula φ is *realizable* if, for every time step, given a truth assignment to the environment propositions for the next time step, there is an assignment of truth values to the robot propositions such that the resulting infinite sequence of truth assignments satisfies φ. The synthesis problem is to find an automaton that provides this assignment. For a synthesizable specification φ, synthesis produces an implementing automaton, enabling the construction of a hybrid controller producing the desired high-level, autonomous robot behavior. The reader is referred to [14] and [12] for details of the synthesis procedure, and to [7], [12] for a description of how the extracted discrete automaton is transformed into low-level robot control.

B. Environment Counterstrategy

When controller synthesis fails, the specification is called *unsynthesizable*. As described in Section I, unsynthesizable specifications are either *unsatisfiable* (e.g., if the task requires patrolling a disconnected workspace), or *unrealizable* (e.g., if in the above task, the environment can disconnect an otherwise connected workspace, such as by closing a door). In either case, the robot can fail in one of two ways: either it ends up in a state from which it has no valid moves (termed *deadlock*), or the robot is always able to change state according to the specified safety requirements, but one of its goals (livenesses) is unreachable (termed *livelock*).

For unsynthesizable specifications, the counterstrategy synthesis algorithm in [11] can be used to extract a strategy for the environment, which provides sequences of environment actions that prevent the specified robot behavior. The counterstrategy takes the form of a finite state machine:

Definition 1 *An* environment counterstrategy *for LTL formula φ is a tuple $A_\varphi^e = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta_e, \delta_s, \gamma_\mathcal{X}, \gamma_\mathcal{Y}, \gamma_{goals})$ where*

- *Q is a set of states.*
- *$Q_0 \subseteq Q$ is a set of initial states.*
- *$\mathcal{X}$ is a set of inputs (sensor propositions).*
- *$\mathcal{Y}$ is a set of outputs (location and action propositions).*
- *$\delta_e : Q \to 2^\mathcal{X}$ is the deterministic input transition relation, which provides the input propositions that are true in the next time step given the current state q, and satisfies φ_e^t.*
- *$\delta_s : Q \times 2^\mathcal{X} \to 2^Q$ is the (nondeterministic) robot transition relation. If $\delta_s(q,x) = \emptyset$ for some $x \in 2^\mathcal{X}, q \in Q$, then there is no next-step assignment to the set of outputs that satisfies the robot's transition relation φ_s^t, given the next set of environment inputs x and the current state q.*
- *$\gamma_\mathcal{X} : Q \to 2^\mathcal{X}$ is a transition labeling, which associates with each state the set of environment propositions that are true over incoming transitions for that state (note that this set is the same for all transitions into a given state). Note that if $q' \in \delta_s(q,x)$ then $\gamma_\mathcal{X}(q') = x$.*
- *$\gamma_\mathcal{Y} : Q \to 2^\mathcal{Y}$ is a state labeling, associating with each state the set of robot propositions true in that state.*
- *$\gamma_{goals} : Q \to \mathbb{Z}^+$ labels each state with the index of a robot goal that is prevented by that state. During the counterstrategy extraction, every state in the counterstrategy is marked with some robot goal [11].*

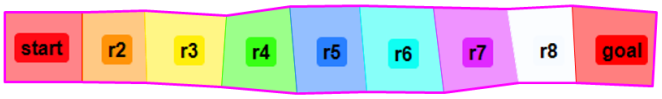


Fig. 1: Map of robot workspace in Specification 1

The counterstrategy provides truth assignments to the input propositions (according to the transition function δ_e) that cause either livelock or deadlock. The inputs provided by δ_e in each state satisfy φ_e^t, meaning that for all $q \in Q$, the truth assignment sequence $(\gamma_\mathcal{X}(q) \cup \gamma_\mathcal{Y}(q), \gamma_\mathcal{X}(\delta_e(q)))$ satisfies A_i for each conjunct A_i in φ_e^t (note that A_i is a formula over two consecutive time steps).

III. Problem Statement

Previous work produced explanations of unrealizability in terms of combinations of the specification components (i.e., initial conditions, safeties and goals). However the true conflict often lies in small subformulas of these components.

Specification 1 Unrealizable specification – livelock

1) *Robot starts in start with camera* ($\pi_{start} \wedge \pi_{camera}$, part of φ_s^i))
2) *If you are sensing a person then do not r5* ($\Box(\bigcirc \pi_{person} \Rightarrow \neg \bigcirc \pi_{r5})$, part of φ_s^t)
3) *Always activate the camera* ($\Box \bigcirc \pi_{camera}$, part of φ_s^t)
4) *Visit the goal* ($\Box\Diamond \pi_{goal}$, part of φ_s^g)

Consider Specification 1, in which the robot is operating in the workspace depicted in Fig. 1. The robot starts at the left hand side of the hallway (line 1), and must visit the goal on the right (4). The robot should not pass through region $r5$ if it senses a person (2), and should always activate its camera (3). The environment can prevent the goal in (4) by always activating the "person" sensor (π_{person}), because of the initial condition in (1) and the safety requirement in (2); this is an example of livelock. The safety requirement in (3) is irrelevant, and should be excluded from any explanation of why this specification is unrealizable.

The algorithm presented in [17] will narrow down the cause of unsynthesizability to the goal in 4. However, it will also highlight the entirely of φ_s^t, declaring that the environment can prevent the goal because of some subset of the safeties; the exact subset is not identified.

This motivates the identification of small, minimal, "core" explanations of the unsynthesizability. In the robotics literature, [9] find minimal revisions of specification automata, by removing the minimum number of constraints from the unsatisfiable specification. The work presented in this paper differs in its objective, which is to provide feedback on existing specifications, not rewrite them. Moreover, this work deals with reactive specifications.

There has also been considerable prior work on unrealizable cores for LTL in the formal methods literature. The authors of [5] propose definitions for helpful assumptions and guarantees, and compute minimal explanations of unrealizability by iteratively expelling unhelpful constraints. Their algorithm assumes an external realizability checker, and performs iterated realizability tests. The main advantage over previous approaches of the work presented in this paper is that it reduces the number of realizability checks required for most specifications, as detailed in Section IV.

To identify and eliminate the source of unrealizability, some works like [4], [13] provide a minimal set of additional environment assumptions that, if added, would make the specification realizable; on the other hand, the work presented in this paper takes the environment assumptions as fixed, and the goal is to compute a minimal subset of the robot guarantees that is unrealizable.

Let $\varphi_1 \preceq \varphi_2$ denote that φ_1 is a subformula of φ_2.

Definition 2 *Given a specification $\varphi = \varphi_e \Rightarrow \varphi_s$, an* unrealizable core *is a subformula $\varphi_s^* \preceq \varphi_s$ such that $\varphi_e \Rightarrow \varphi_s^*$ is unrealizable, and for all $\varphi_s' \prec \varphi_s^*$, $\varphi_e \Rightarrow \varphi_s'$ is realizable.*

Problem 1 *Given an unrealizable formula φ, return an unrealizable core φ_s^*.*

Note that unrealizable cores of φ may not be unique.

IV. Unrealizable Cores

In this section, unrealizable components of the robot specification φ_s are analysed based on the environment counterstrategy, narrowing down the cause of unrealizability for both deadlock and livelock. Consider a counterstrategy $A_\varphi^e = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta_e, \delta_s, \gamma_\mathcal{X}, \gamma_\mathcal{Y}, \gamma_{goals})$ for formula φ. It allows the following characterizations of deadlock and livelock:

- **Deadlock** There exists a state in the counterstrategy such that there is a truth assignments to inputs, for which no truth assignment to outputs satisfies the robot transition relation. Formally,

$$\exists q \in Q | \delta_s(q, \delta_e(q)) = \emptyset$$

- **Livelock** There exist a set of states $\mathcal{C}$ in the counterstrategy such that the robot is trapped in $\mathcal{C}$ no matter what it does, and there is some robot liveness B_k in φ_s^g that is not satisfied by any state in $\mathcal{C}$. Formally,

$$\exists \mathcal{C} \subseteq Q, B_k \preceq \varphi_s^g | \forall q \in \mathcal{C}, q \not\models B_k, \delta_s(q, \delta_e(q)) \subseteq \mathcal{C}$$

A. Unrealizable Cores for Deadlock

Consider Specification 2 on the workspace in Fig. 1. The robot starts in $r5$ with the camera on (1). The safety conditions specify that the robot should not pass through the region marked $r5$ if it senses a person (2). However, the robot must stay in place if it senses a person (3). Finally, the robot should always activate its camera (4). Here, the environment can force the robot into deadlock by activating the "person" sensor (π_{person}) when the robot is in $r5$, because there is then no way the robot can fulfil both (2) and (3).

The environment counterstrategy A_φ^e is as follows:

- $Q = Q_0 = \{q_1\}$
- $\mathcal{X} = \{\pi_{person}\}$, $\mathcal{Y} = \{\pi_{r5}, \pi_{camera}\}$
- $\delta_e(q_1) = \{\pi_{person}\}$, $\delta_s(q_1, \{\pi_{person}\}) = \emptyset$, $\delta_s(q_1, \emptyset) = \emptyset$
- $\gamma_\mathcal{X}(q_1) = \{\pi_{person}\}$, $\gamma_\mathcal{Y}(q_1) = \{\pi_{r5}, \pi_{camera}\}$
- $\gamma_{goals}(q_1) = 1$

Notice that q_1 is a deadlocked state, because given the input π_{person} in the next time step, there is a conflict between safety conditions 2 and 3, and the robot has no valid move (hence $\delta_s(q_1, \{\pi_{person}\}) = \emptyset$).

For $q \in Q$, define the *propositional-representation of q* as:

$$\psi_{state}(q) = \bigwedge_{x \in \gamma_\mathcal{X}(q)} x \wedge \bigwedge_{x \in \mathcal{X} \setminus \gamma_\mathcal{X}(q)} \neg x \wedge \bigwedge_{y \in \gamma_\mathcal{Y}(q)} y \wedge \bigwedge_{y \in \mathcal{Y} \setminus \gamma_\mathcal{Y}(q)} \neg y$$

In the example above, $\psi_{state}(q_1) = \pi_{person} \wedge \pi_{r5} \wedge \pi_{camera}$.

Let π^i represent the value of $\pi \in AP$ at time step i, and $AP^i = \{\pi^i \mid \pi \in AP\}$. For example, in Specification 2, $AP^0 = \{\pi^0_{person}, \pi^0_{r5}, \pi^0_{camera}\}$ and $AP^1 = \{\pi^1_{person}, \pi^1_{r5}, \pi^1_{camera}\}$. Given LTL specification φ, $q \in Q$ such that $\delta_s(q, \delta_e(q)) = \emptyset$, construct a propositional formula over $AP^0 \cup AP^1$ as follows:

$$\psi_{dead}(q, \varphi) = \psi_{state}(q)[\pi/\pi^0] \wedge \bigwedge_{z \in \delta_e(q)} z^1 \wedge \bigwedge_{z \in \mathcal{X} \setminus \delta_e(q)} \neg z^1 \wedge \varphi_s^t[\bigcirc \pi/\pi^1][\pi/\pi^0],$$

where $\varphi[a/b]$ represents φ with all occurrences of subformula a replaced with b. Intuitively, this formula represents the satisfaction of the robot safety condition in the next step from state q, with the additional restriction that the input variables be bound to the values provided by $\delta_e(q)$ in the next time step. Notice that by the definition of deadlock, $\psi_{dead}(q, \varphi)$ represents an unsatisfiable propositional formula.

In the above case, $\psi_{dead}(q_1, \varphi) =$

$$\pi^0_{person} \wedge \pi^0_{r5} \wedge \pi^0_{camera} \wedge \pi^1_{person} \wedge \pi^1_{camera} \wedge \varphi^0_{topo} \wedge (\pi^1_{person} \Rightarrow \neg \pi^1_{r5}) \wedge (\pi^1_{person} \Rightarrow (\pi^1_{r5} \Leftrightarrow \pi^0_{r5} \wedge ...)),$$

where φ^i_{topo} is a formula over $AP^i \cup AP^{i+1}$ representing the topological constraints on the robot motion at time i (i.e. which rooms it can move to at time $i+1$ given where it is at time i, and mutual exclusion between rooms).

Note that if q is a deadlocked state, then by definition $\psi_{dead}(q, \varphi)$ is unsatisfiable, since there is no truth assignment to the robot propositions in the next time step that satisfies φ_s^t. An off-the-shelf satisfiability (SAT) solver can be used to verify unsatisfiability of $\psi_{dead}(q, \varphi)$; this work uses PicoSAT [2]. The same SAT solver can then be used to find a minimal

unsatisfiable subformula, which is then mapped back to the originating portions of the safety and initial formulas.

In the above example, the SAT solver finds the core of $\psi_{dead}(q,\varphi)$ as the subformula

$$\pi^0_{r5} \wedge \pi^1_{person} \wedge (\pi^1_{person} \Rightarrow \neg\pi^1_{r5}) \wedge (\pi^1_{person} \Rightarrow (\pi^1_{r5} \Leftrightarrow \pi^0_{r5})).$$

This is because the two statements combined require the robot to both stay in r5 and not be in r5 in time step 1. This gives us a core explanation of the deadlock caused in state q. Taking the union over the cores for all the deadlocked states provides a concise explanation of the cause of deadlock.

Specification 2 Core-finding example – deadlock

1) Robot starts in r5 with camera (φ^i_s):
   $\pi_{r5} \wedge \pi_{camera}$
2) If you are sensing person then do not r5 (φ^t_s):
   $\square(\bigcirc \pi_{person} \Rightarrow \neg \bigcirc \pi_{r5})$
3) If you are sensing person then stay there (φ^t_s):
   $\square(\bigcirc \pi_{person} \Rightarrow (\bigcirc \pi_{start} \Leftrightarrow \pi_{start} \wedge \bigcirc \pi_{r2} \Leftrightarrow \pi_{r2}...))$
4) Always activate your camera (φ^t_s):
   $\square \bigcirc \pi_{camera}$

B. Unsatisfiable Cores for Livelock

Consider Specification 1 again. If the environment action is to always set π_{person}, then the safety requirement in 2 enforces that the robot will never activate π_{r5}, because it is explicitly forbidden from doing so when sensing a person. This is livelock because the robot can continue to move between *start* and $r2-4$. Counterstrategy A^e_φ is as follows:

- $Q = \{q_1, q_2, q_3, q_4\},\ Q_0 = \{q_1\}$
- $\mathcal{X} = \{\pi_{person}\}, \mathcal{Y} = \{\pi_{start}, \pi_{r2}, \pi_{r3}, ..., \pi_{r8}, \pi_{goal}, \pi_{camera}\}$
- $\forall q \in Q, \delta_e(q) = \{\pi_{person}\}$
- $$\delta_s(q_i, \{\pi_{person}\}) = \begin{cases} \{q_1, q_2\} & \text{if } i = 1 \\ \{q_{i-1}, q_i, q_{i+1}\} & \text{for } 2 \le i \le 3 \\ \{q_3, q_4\} & \text{if } i = 4 \end{cases}$$
  $\delta_s(q, \emptyset) = \emptyset$ for $q \in Q$
- $\forall q \in Q, \gamma_{\mathcal{X}}(q) = \{\pi_{person}\}$.
- $$\gamma_{\mathcal{Y}}(q_i) = \begin{cases} \{\pi_{camera}, \pi_{start}\} & \text{if } i = 1 \\ \{\pi_{camera}, \pi_{r_i}\} & \text{for } 2 \le i \le 4 \end{cases}$$
- $\forall i, \gamma_{goals}(q_i) = 1$ (since there is only one goal).

In the case of livelock, we know there exists a set of states $\mathcal{C}$ in the counterstrategy that trap the robot, locking it away from the goal. Without loss of generality, $\mathcal{C}$ consists of cycles of states. In the specifications of the form considered in this work, robot goals are of the form $\square\diamondsuit B_i$ for $1 \le i \le n$, where each B_i is a propositional formula over $AP = \mathcal{X} \cup \mathcal{Y}$. Suppose the algorithm in [17] identified goal $\square\diamondsuit B_k$ as the goal responsible for livelock. Let Q_k be the set of all states in A^e_φ that prevent goal B_k, and let $\mathcal{C}_k$ be the set of *maximal k-preventing cycles* in Q_k, i.e. cycles that are not contained in any other cycle in Q_k (modulo state-repetition). Let $C_1 = (q^1_0, q^1_1, ..., q^1_a)$ and $C_2 = (q^2_0, q^2_1, ..., q^2_b)$ be cycles, and define $C_1 \prec C_2$ if $a < b$ and there is some offset index o in C_2 such that all of C_1 is found in C_2 starting at o, i.e. $q^1_i = q^2_{(i+o) \bmod (b+1)}$ for all $0 \le i \le a$. This expresses that C_1 is a strict sub-cycle of C_2. Then formally,

$$\begin{aligned} Q_k &= \{q \in Q | \gamma_{goals}(q) = k\} \\ \mathcal{C}^{all}_k &= \{(q_0, q_1, ..., q_l) | \forall 0 \le i \le l, q_i \in Q_k, \\ &\quad \forall i < l, q_{i+1} \in \delta_s(q_i, \delta_e(q_i)), q_i \ne q_{i+1}, \\ &\quad q_0 \in \delta_s(q_l, \delta_e(q_l)), q_0 \ne q_l\}, \\ \mathcal{C}_k &= \{C \in \mathcal{C}^{all}_k | \forall C' \in \mathcal{C}^{all}_k, C \not\prec C'\}. \end{aligned}$$

In Specification 1, there is only one goal, $\square\diamondsuit \pi_{goal}$. $C_1 = (q_1, q_2, q_3, q_4, q_3, q_2)$ is a maximal 1-preventing cycle.

Given an initial state q, a depth d and an LTL safety formula φ^t_s over $\pi \in AP$, there exists a propositional formula $\psi^d(\varphi^t_s, q)$ over $\bigcup_{0 \le i \le d+1} AP^i$, constructed as:

$$\psi^d(\varphi^t_s, q) = \psi_{state}(q)[\pi/\pi^0] \wedge \bigwedge_{0 \le i \le d} \varphi^t_s[\bigcirc\pi/\pi^{i+1}][\pi/\pi^i].$$

This formula is called the *depth-d unrolling of* φ^t_s *from* q, and represents the tree of length-$d+1$ truth assignment sequences that satisfy φ^t_s, starting from q. Note that there are $d+1$ time steps in a depth-d unrolling because each conjunct in φ^t_s governs two time steps. In the example, $\psi^d(\varphi^t_s, q_1) =$

$$\pi^0_{start} \wedge \pi^0_{camera} \wedge \bigwedge_{0 \le i \le d} (\varphi^i_{topo} \wedge \pi^{i+1}_{camera} \wedge (\pi^{i+1}_{person} \Rightarrow \neg\pi^{i+1}_{r5})).$$

Given a cycle of states $C = (q_0, q_1, ..., q_l) \in A^e_\varphi$, and a depth d, construct a propositional formula $\psi^d_{\mathcal{X}}(C)$ over $\bigcup_{0 \le i \le d} \mathcal{X}^i$, where $x^i \in \mathcal{X}^i$ represents the value of each input $x \in \mathcal{X}$ in state $q_{i \bmod (l+1)}$ for $0 \le i \le d$, as:

$$\psi^d_{\mathcal{X}}(C) = \bigwedge_{0 \le i \le d} \Big(\bigwedge_{p \in \gamma_{\mathcal{X}}(q_{i \bmod (l+1)})} x^i \wedge \bigwedge_{p \in \mathcal{X} \setminus \gamma_{\mathcal{X}}(q_{i \bmod (l+1)})} \neg x^i \Big).$$

This formula is called the *depth-d environment-unrolling of* C, and represents the sequence of inputs seen when following cycle C for d time-steps. In the example, the depth-d environment unrolling of C_1 is $\psi^d_{\mathcal{X}}(C) = \bigwedge_{0 \le i \le d} \pi^i_{person}$.

Now, given an LTL safety specification φ^t_s over $\pi \in AP$, a goal B_k, a maximal k-preventing cycle $C_k = (q_0, q_1, ..., q_l) \in \mathcal{C}_k$, and an unrolling depth d, construct propositional formula $\psi^d_{live}(B_k, C_k, \varphi^t_s)$ over $\bigcup_{0 \le i \le d+1} AP^i$ as:

$$\psi^d_{live}(B_k, C_k, \varphi^t_s) = \psi^{d+1}_{\mathcal{X}}(C_k) \wedge \psi^d(\varphi^t_s, q_0) \wedge B_k[\bigcirc\pi/\pi^{d+1}][\pi/\pi^d].$$

Intuitively, this formula expresses the requirement that the goal B_k be fulfilled after some depth-d unrolling of the safety formula starting from state q_0, given the input sequence provided by $\psi^{d+1}_{\mathcal{X}}(C_k)$ (note that this input sequence extends to the final time step in the safety formula unrolling). Again, this is an unsatisfiable propositional formula, and can be used to determine the core set of clauses that prevent a goal from being fulfilled. Taking the union of cores over all $C_k \in \mathcal{C}_k$ gives a concise explanation of the ways in which the environment can prevent the robot from fulfilling the goal.

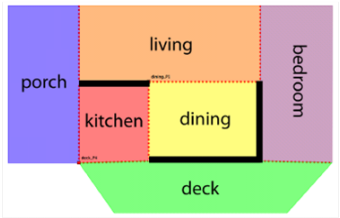


Fig. 2: Map of robot workspace for specifications in Section V

In the above example, $\psi^d_{live}(\pi_{goal}, C_1, \varphi^t_s) =$

$$\pi^0_{start} \wedge \pi^0_{camera} \wedge \bigwedge_{0 \le i \le d+1} \pi^i_{person} \wedge \bigwedge_{0 \le i \le d} (\varphi^i_{topo} \wedge \pi^{i+1}_{camera} \wedge (\pi^{i+1}_{person} \Rightarrow \neg \pi^{i+1}_{r5})) \wedge \pi^d_{goal}.$$

In the case of livelock, the choice of unroll depth d determines the quality of the core returned. Recall that for deadlock, the propositional formula $\psi_{dead}(q, \varphi)$ is built over just one step, since it is already known to cause a conflict with the robot transition relation, and be unsatisfiable. The unsatisfiable core of this formula is a meaningful unrealizable core in this case because it provides the immediate reason for the deadlock. For livelock, on the other hand, determining the shortest depth to which a cycle C_k must be unrolled to produce a meaningful core is not obvious.

In the above example, for unroll depths less than or equal to 8, the unsatisfiable core returned will include just the environment topology, since the robot cannot reach the goal from the start in 8 steps or fewer, even if it is allowed into $r5$; however, this is not a meaningful core. Unrolling to depth 9 or greater returns the expected subformula that includes $\bigwedge_{0 \le i \le d}(\pi^{i+1}_{person} \Rightarrow \neg \pi^{i+1}_{r5})$. Automatically determining the shortest depth that will produce a meaningful core remains a research challenge. This minimum depth is often tied to the number of regions in the robot workspace, and is usually easy to estimate based on the diameter of the graph representing the counterstrategy. However, no efficient, sound method is known for determining this minimal unrolling depth, and for the examples in this paper a depth of 15 time steps was used.

If the SAT-based analysis in this section returns a core that does not capture the real cause of failure, alternative, more computationally expensive techniques can be used to return a minimal core. For example, the approach in [5] is guaranteed to yield a minimal core, but requires repeated calls to a realizability oracle, which may be expensive for specifications with a large number of conjuncts.

Note that, since unsatisfiability is a special case of unrealizability (in which not just *some*, but *any* environment can prevent the robot from fulfilling its specification), the above analysis also applies to unsatisfiable specifications.

V. Examples

This section presents examples of the improved feedback provided for unrealizable specifications. The examples presented previously appeared in [16], and this section demonstrates the improvement of the proposed analysis over the approach presented in that work.

A. Deadlock

Consider the specification in Fig. 3, where the robot is operating in the workspace depicted in 2. The robot starts in the porch. The safety conditions govern what it should do when it senses a "person" (stay with them and radio for help) or a "hazardous item" (pick up the hazardous item and carry it to the porch). The robot should not return to the porch unless it is carrying a hazardous item, and its goal is to patrol all the other rooms.

The environment can cause deadlock by setting the person sensor to true and the hazardous item sensor to false when the robot is in the porch. Sensing a hazardous item results in the robot activating the "pick_up" action, which in turn results in the proposition "carrying_item" being set. Similarly, sensing a person results in the robot turning on the radio. Now the state in which both "radio" and "carrying_item" are true is a deadlocked state, since there is no way to satisfy both the safety conditions "If you are. activating radio or you were activating radio then stay there" and "If you did not activate carrying_item then always not porch" from this state.

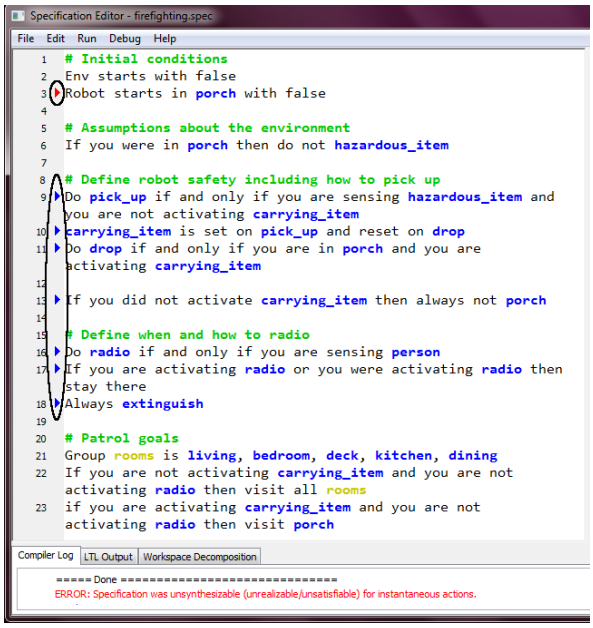
Fig. 3(a) depicts the sentences highlighted by the algorithm in [17]. Sentences in the specification are identified by triangle-shaped markers in the left-hand margin. The sentences highlighted in 3(a) include all initial (red) and safety (blue) conditions, which forms a very large subset of the original specification. On the other hand, Fig. 3(b) depicts the much smaller subset of sentences highlighted by the analysis in this paper (all in red). These sentences correspond to the safety conditions that cause deadlock – removing any one of them makes the specification synthesizable.

B. Livelock

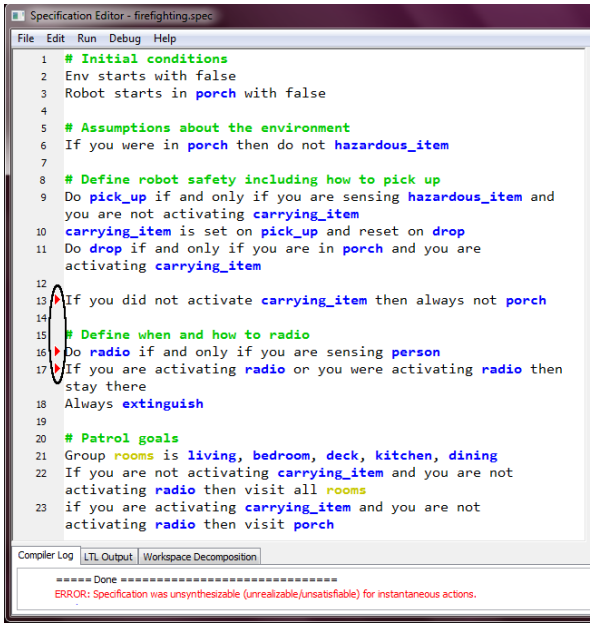
Consider the specification in Fig. 4, also in the same workspace. The robot starts in the deck and its goal is to visit the porch. However, based on whether it senses a person or a fire, it has to keep out of the kitchen and the living room, respectively. Fig. 4(a) depicts the sentences highlighted by the algorithm in [17], which includes all safety conditions (red) in addition to the goal (green). This includes irrelevant sentences, such as the one that requires the robot to always turn on the camera. Fig. 4(b) depicts the core returned by the analysis in this work – only those safeties that directly contribute to keeping the robot out of the porch are returned.

VI. Conclusions

This paper presents techniques for analyzing high-level robot specifications that are unrealizable, i.e. for which no implementing controller exists because the environment can prevent the desired robot behaviour. The approach is based on finding minimal unsatisfiable cores for propositional encodings of specific state sequences in the environment counterstrategy. Examples show that the additional analysis finds the minimal cause of unrealizability and ignores irrelevant subformulas. Future work includes automatically determining the depth for obtaining a meaningful core for livelock, and exploring techniques that do not require explicit state extraction of the counterstrategy automaton.

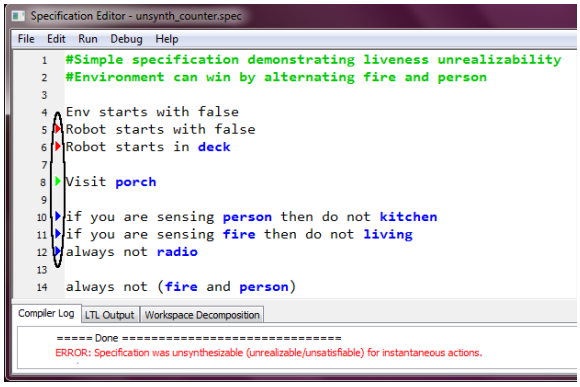


(a) Sentences highlighted using approach in [17]

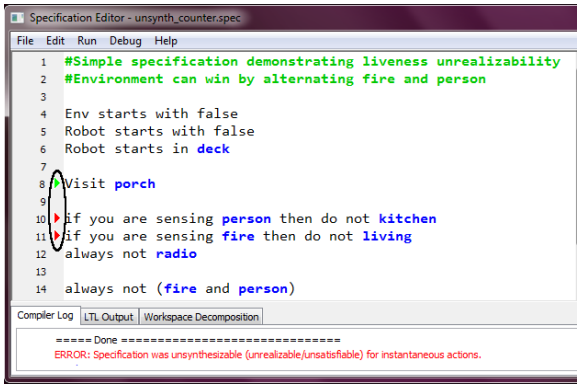


(b) Sentences highlighted using proposed approach

Fig. 3: Core-Finding Example: Deadlock



(a) Sentences highlighted using approach in [17]



(b) Sentences highlighted using proposed approach

Fig. 4: Core-Finding Example: Livelock

REFERENCES

[1] Amit Bhatia, Lydia E. Kavraki, and Moshe. Y. Vardi. Sampling-Based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.

[2] Armin Biere. PicoSAT Essentials. *Journal on Satisfiability (JSAT)*, 4(2-4):75–97, 2008.

[3] Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, Katrina Gossman, and Steven LaValle. Controlling Wild Bodies Using Linear Temporal Logic. In *RSS*, Los Angeles, CA, USA, June 2011.

[4] Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment Assumptions for Synthesis. In *CONCUR*, pages 147–161, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Andrei Tchaltsev. Diagnostic Information for Realizability. In *VMCAI*, pages 52–67, 2008.

[6] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[7] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. LTLMoP: Experimenting with Language, Temporal Logic and Robot Control. In *IROS*, pages 1988 – 1993, 2010.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-Based Motion Planning with Deterministic μ-Calculus Specifications. In *CDC*, 2009.

[9] Kangjin Kim, Georgios E. Fainekos, and Sriram Sankaranarayanan. On the Revision Problem of Specification Automata. In *ICRA*, pages 5171–5176, 2012.

[10] Marius Kloetzer and Calin Belta. A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[11] Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging Formal Specifications Using Simple Counterstrategies. In *FMCAD*, pages 152–159, 2009.

[12] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[13] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. Mining Assumptions for Synthesis. In *MEMOCODE*, pages 43–50, 2011.

[14] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380. Springer, 2006.

[15] Amir Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57, 1977.

[16] Vasumathi Raman and Hadas Kress-Gazit. Automated Feedback For Unachievable High-Level Robot Behaviors. In *ICRA*, pages 5156–5162, 2012.

[17] Vasumathi Raman and Hadas Kress-Gazit. Explaining Impossible High-Level Robot Behaviors. *IEEE Transactions on Robotics*, PP(99):1 –11, 2012.

[18] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding Horizon Control for Temporal Logic Specifications. In *HSCC*, pages 101–110, 2010.