Analyzing Unsynthesizable Specifications for High-Level Robot Behavior Using LTLMoP

Vasumathi Raman and Hadas Kress-Gazit [*]

Cornell University,
Ithaca, NY, USA 14853
vraman@cs.cornell.edu, hadaskg@cornell.edu

Abstract. Recent work in robotics has applied formal verification tools to automatically generate correct-by-construction controllers for autonomous robots. However, when it is not possible to create such a controller, these approaches do not provide the user with feedback on the source of failure, making the experience of debugging a specification somewhat ad hoc and unstructured, and a source of frustration for the user. This paper describes an extension to the LTLMoP toolkit for robot mission planning that encloses the control-generation process in a layer of automated reasoning to identify the cause of failure, and targets the users attention to flawed portions of the specification.

Keywords: synthesis, LTL, GR(1), unrealizability, unsatisfiability, robot control

1 Introduction

High-level robot control is a topic of current research in robotics. The goal is to automatically generate controllers for autonomous robots to achieve desired high-level behavior involving a non-trivial sequence of actions, such as, "collect all my socks from the apartment floor and put them in the laundry bag". Recent work in robotics [11, 15] has applied efficient synthesis techniques [12] to automatically generate provably correct, closed loop, low-level robot controllers that satisfy high-level behaviors specified in temporal logic. A discrete abstraction of the workspace is used, and the robot goals and environment assumptions are described using Linear Temporal Logic, which can express a rich set of infinite behaviors. The generated continuous robot controllers are provably correct in that the closed loop system they form is guaranteed, by construction, to satisfy the desired specification when the robot operates in an environment that satisfies the modeled assumptions.

However, such synthesis-based approaches present the user with no feedback when synthesis is impossible, i.e., when there exists an environment in which the robot fails to achieve the desired behavior – we call such a specification *unsynthesizable*. An unsynthesizable specification is either *unsatisfiable*, in which case the robot cannot achieve the desired behavior in any environment, or *unrealizable*, in which case there exist environments that can thwart the robot. When the specification is unsynthesizable, the

[*] This work was supported by NSF CAREER CNS-0953365 and ARO MURI (SUBTLE) W911NF-07-1-0216

above approaches fail to produce the desired behavior, but do not provide the user with a source of failure, or suggest changes that would allow synthesis of a controller. In addition, even when synthesis is possible, the generated automaton may fail to produce the desired behavior for reasons that involve unsatisfiability or unrealizability of the environment assumptions. This can make the experience of debugging a specification somewhat ad hoc and unstructured, and a source of frustration. This paper describes a procedure for enclosing the control-generation process in a layer of automated reasoning that focuses the cause of failure and targets the users attention to relevant portions of the specification. We present a method of narrowing down the source of unsynthesizability in specifications that can be represented as GR(1) formulas in LTL.

The debugging procedure described in this paper is implemented within Linear Temporal Logic MissiOn Planning (LTLMoP)[8], an open source, modular, Python-based toolkit that allows users to input structured English specifications describing high-level robot behavior, and automatically generates and implements the relevant hybrid controllers using the approach of [11]; the synthesized controllers can be embedded within a simulator or used with physical robots. The most recent version of LTLMoP can be downloaded online[1]. There has been considerable previous work on analyzing unsatisfiable and unrealizable LTL formulas [2, 4–6, 14]. Our work is most closely related to that of [9], who present an interactive tool, RATSY [3] for demonstrating specification unrealizability. The debugging procedure we implement for LTLMoP differs from RATSY in that rather than allowing the user to explore counterexamples, it provides explicit information about specification components in the context of the robot control problem. We highlight flawed portions of the user-defined specification (either desired system behavior or environment assumptions), and identify cases of unexpected behavior that specifically affect this domain, such as trivial solutions.

2 Technical Overview

We first review some preliminaries relating to the application of formal methods to high-level robot control, and outline LTLMoP's controller-synthesis procedure (depicted in Fig. 1). We consider a robot functioning in a continuous environment. The robot reacts to the environment as perceived through its sensor inputs, and chooses from a set of actions including moving between adjacent locations. The tasks themselves include infinite behaviors such as visiting locations or performing actions infinitely often.

Applying formal methods techniques such as model checking and synthesis to continuous settings in robotics requires a discrete abstraction of problems to enable description with a formal language. As mentioned earlier, the underlying formal language used to define high-level specifications in this work is Linear Temporal Logic (LTL) (cf. [7]). LTLMoP includes a parser that automatically translates English sentences from a defined grammar [10] into LTL formulas. This allows users to define desired robot behaviors (including reactive behaviors) and specify assumptions on the environment's behavior using an intuitive descriptive language rather than the underlying formalism.

As shown in Fig. 1, LTLMoP takes as input a user-defined specification, a map of the environment and a description of the robot sensors and actuators. The specification

[1] http://ltlmop.github.com

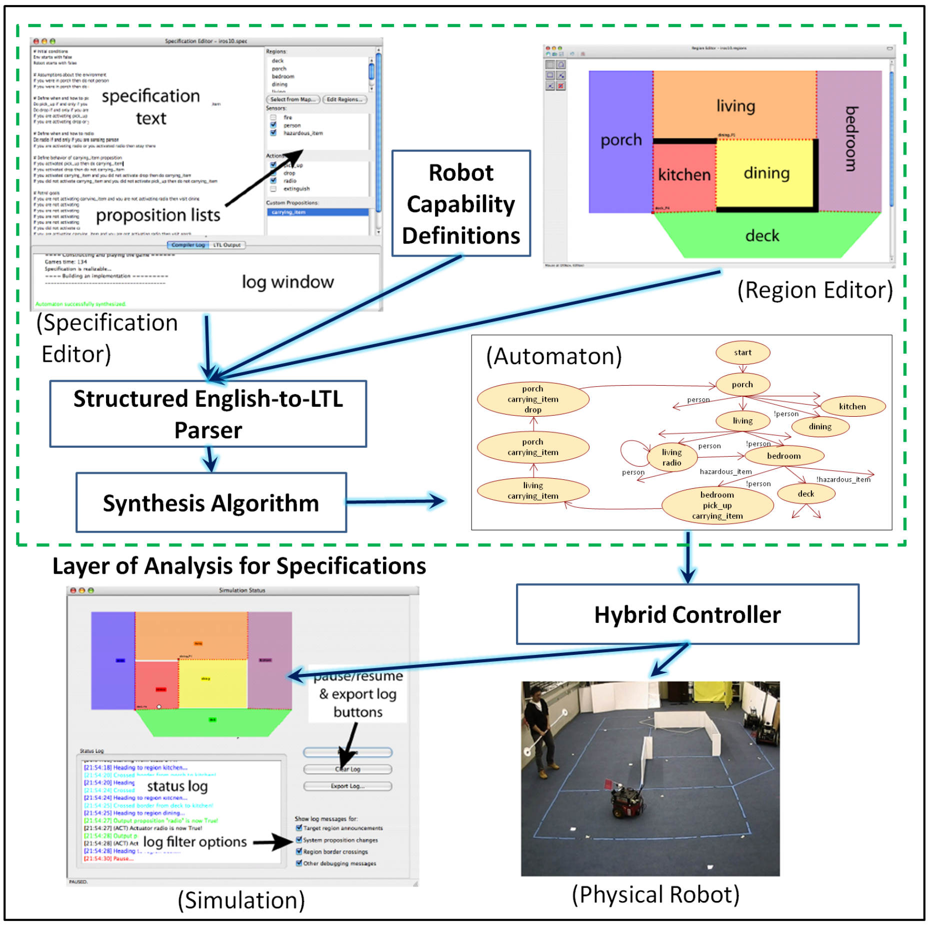


Fig. 1: Overview of LTLMoP architecture

is parsed into a formula of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$, where φ_e includes assumptions about the sensor propositions, and thus about the behavior of the environment, and φ_s represents the desired robot behavior. φ is in the subclass of LTL described in [12], and the efficient algorithm introduced therein is used to synthesize an automaton that implements the input specification with the described robot in the given environment. The synthesis algorithm is implemented in the JTLV framework [13], with formulas for the system and environment initial conditions, transitions and goals represented as Binary Decision Diagrams (BDDs).

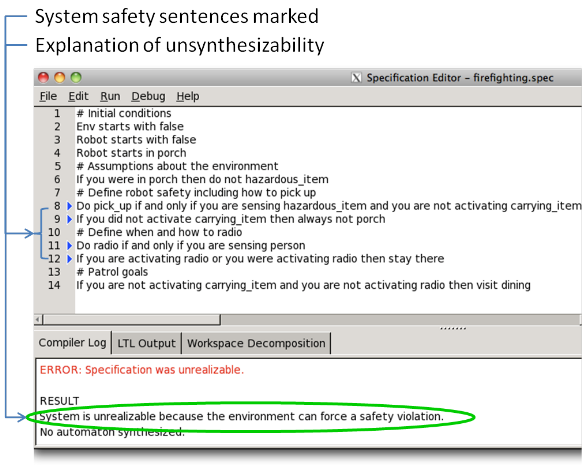
The created automaton is correct-by-construction: if a behavior can be achieved in all environments satisfying the given assumptions, then the LTL formula describing the behavior holds for every possible execution of the automaton. This implementing discrete automaton is then viewed as a hybrid controller, wherein a transition between states corresponds to the activation of one or more atomic continuous controllers that satisfy the bisimulation property [1] (e.g., the motion controllers are guaranteed to drive the robot from one region to another regardless of the initial state within the region).

We refer the reader to [11] for a complete discussion of the hybrid controller, and to [8] for a description of how atomic controllers are incorporated into the hybrid controller in LTLMoP. Finally, the synthesized hybrid controller can be embedded within a simulator or used with physical robots (such as the Pioneer 3-DX depicted in Fig. 1).

3 Unsynthesizable Specifications and Undesirable Behavior

A specification $\varphi_e \implies \varphi_s$ that does not produce a controller is either *unrealizable* or *unsatisfiable*, and there are several possible reasons for either. In addition, if φ_e is unsatisfiable, then all initial states are winning for the system, and so we do get an automaton, but a trivial one consisting of all the initial states but no transitions. We would like to identify this case, since the resulting behavior is probably not as intended.

With regards to unrealizability, we can just as well consider winning system strategies that prevent the environment from satisfying the formula φ_e. Overloading terminology, we say that the environment is unrealizable in this case. Note that if the environment is unrealizable, an otherwise unrealizable robot specification may be synthesizable if the robot can win by preventing the environment from upholding its assumptions. In fact, if the environment is unsatisfiable, every robot specification (even an unsatisfiable one) is synthesizable. In the robotics domain, we would like to flag this case, since we would like the robot to fulfill its goals rather than thwart the given environment.

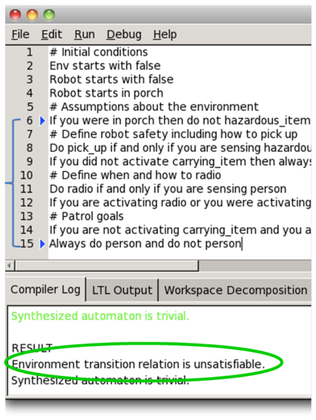

System safety sentences marked
Explanation of unsynthesizability
Specification Editor – firefighting.spec
File Edit Run Debug Help
Compiler Log
LTL Output
Workspace Decomposition

```
# Initial conditions
Env starts with false
Robot starts with false
Robot starts in porch
# Assumptions about the environment
If you were in porch then do not hazardous_item
# Define robot safety including how to pick up
Do pick_up if and only if you are sensing hazardous_item and you are not activating carrying_item
If you did not activate carrying_item then always not porch
# Define when and how to radio
Do radio if and only if you are sensing person
If you are activating radio or you were activating radio then stay there
# Patrol goals
If you are not activating carrying_item and you are not activating radio then visit dining
```

```
ERROR: Specification was unrealizable.

RESULT
System is unrealizable because the environment can force a safety violation.
No automaton synthesized.
```

(a) Analyzing an unsynthesizable specification



```
# Initial conditions
Env starts with false
Robot starts with false
Robot starts in porch
# Assumptions about the environment
If you were in porch then do not hazardous_item
# Define robot safety including how to pick up
Do pick_up if and only if you are sensing hazardou
If you did not activate carrying_item then alway
# Define when and how to radio
Do radio if and only if you are sensing person
If you are activating radio or you were activating
# Patrol goals
If you are not activating carrying_item and you a
Always do person and do not person
```

```
Synthesized automaton is trivial.

RESULT
Environment transition relation is unsatisfiable.
Synthesized automaton is trivial.
```

(b) Trivially synthesizable spec.

Fig. 2

Listing 1 Excerpt of a fire-fighting scenario specification with corresponding LTL

1	Env starts with false	$\neg\pi_{person} \wedge \neg\pi_{hazardous\_item}$
2	Robot starts with false	$\neg\pi_{pick\_up} \wedge \neg\pi_{drop} \wedge \neg\pi_{carrying\_item}$
3	Robot starts in **porch**	φ_{porch}
4	If you were in **porch** then do not **person**	$\square(\varphi_{porch} \Rightarrow \neg \bigcirc \pi_{person})$
5	If you were in **porch** then do not **hazardous_item**	$\square(\varphi_{porch} \rightarrow \neg \bigcirc \pi_{hazardous\_item})$
6	Do **pick_ up** if and only if you are sensing **hazardous_ item** and you are not activating **carrying_ item**	$\square(\bigcirc \pi_{pick\_up} \Leftrightarrow (\bigcirc \pi_{hazardous\_item} \wedge \neg \bigcirc \pi_{carrying\_item}))$
7	If you did not activate **carrying_ item** then always not **porch**	$\square(\neg\pi_{carrying\_item} \rightarrow \neg \bigcirc \varphi_{porch})$
8	Do **radio** if and only if you are sensing **person**	$\square(\bigcirc \pi_{radio} \Leftrightarrow \bigcirc \pi_{person})$
9	If you are activating **radio** or you were activating **radio** then stay there	$\square((\bigcirc \pi_{radio} \vee \pi_{radio}) \rightarrow \bigwedge_{l}(\varphi_l \Leftrightarrow \bigcirc \varphi_l))$
10	If you are not activating **carrying_ item** and you are not activating **radio** then visit **dining**	$\square \diamondsuit((\neg\pi_{carrying\_item} \wedge \neg\pi_{radio}) \rightarrow \varphi_{dining})$
		...

4 Analyzing a Specification in LTLMoP

Given an unsynthesizable specification in LTLMoP, we apply a series of simple checks to determine which components of the corresponding LTL formula are flawed, trace them back to their structured English counterparts, and highlight these in the Specification Editor. We identify unsatisfiability of the system and environment initial conditions, single-step transitions, goals, and the conjunction of transitions and goals using boolean satisfiability tests, without checking the LTL specification as a whole. As a result, some unsatisfiable safety conditions are identified as unrealizable instead.

Consider the specification in Listing 1 from the fire-fighting scenario described in [8]. Removing the environment safety requirement in line 4 makes the specification unrealizable, because the environment can force the robot into a safety violation by setting π_{person} to true in the porch. By line 8, this causes the robot to set π_{radio} to true in the next time step; line 9 then requires it to stay where it is (i.e., $\bigcirc \varphi_{porch}$), but line 7 requires $\neg \bigcirc \varphi_{porch}$. The robot thus has no legal next state. Our analysis determines that the system (robot) is unrealizable because the environment can force a safety violation, and marks all safety sentences as in Fig. 2(a). Consider the same specification, augmented with the (clearly unsatisfiable) environment safety condition, Always person and not person ($\square(\pi_{person} \wedge \neg\pi_{person})$). Synthesis succeeds, but as noted in Fig. 2(b), the environment liveness is unsatisfiable and the generated automaton is trivial.

5 Conclusions and Future Work

We have described a method for systematically analyzing the environment and system components of autonomous robot control specifications. By exploiting the structure of the specification, we identify possible reasons for failure to create an implementing robot controller. Our approach is implemented as part of the open source LTLMoP toolkit. We enclose the synthesis in a layer of reasoning that identifies the cause of failure, enabling the user to target their attention to the relevant portions of the specification. Once we identify a specification (or part thereof) as unsatisfiable or unrealizable, there is still potential for further analysis. Future work will leverage existing techniques [2, 5, 6, 9] to isolate the source of failure and provide the user with comprehensive feedback, including modifications to the input that would result in an implementing automaton.

References

1. R. Alur, T. A. Henzinger, G. Lafferriere, George, and G. J. Pappas. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*, pages 971–984, 2000.
2. I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. J. Trefler. Explaining counterexamples using causality. In *Computer Aided Verification*, pages 94–108, 2009.
3. R. P. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, and R. Seeber. RATSY - a new requirements analysis tool with synthesis. In Springer, editor, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 425 – 429, 2010.
4. K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proceedings of the 19th international conference on Concurrency Theory*, CONCUR '08, pages 147–161, Berlin, Heidelberg, 2008. Springer-Verlag.
5. A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In F. Logozzo, D. Peled, and L. D. Zuck, editors, *VMCAI*, volume 4905 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2008.
6. A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 532–546. Springer, 2007.
7. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
8. C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP: Experimenting with language, temporal logic and robot control. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 1988 – 1993, 2010.
9. R. Könighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *Formal Methods in Computer-Aided Design*, pages 152–159, 2009.
10. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.
11. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.
12. N. Piterman and A. Pnueli. Synthesis of reactive(1) designs. In *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI06*, pages 364–380. Springer, 2006.
13. A. Pnueli, Y. Sa'ar, and L. D. Zuck. JTLV: A framework for developing verification algorithms. In *Computer Aided Verification*, pages 171–174, 2010.
14. V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In *Fundamentals of Software Engineering, Third IPM International Conference*, pages 129–145, 2009.
15. T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *Hybrid Systems*, pages 101–110, 2010.