SynKit: LTL Synthesis as a Service

Alberto Camacho†, Christian Muise*, Jorge A. Baier‡, Sheila A. McIlraith†
†Department of Computer Science, University of Toronto
*CSAIL, Massachusetts Institute of Technology
‡Pontificia Universidad Católica de Chile
†Chilean Center for Semantic Web Research
†{acamacho,sheila}@cs.toronto.edu, *cjmuise@mit.edu, ‡jabayer@ing.puc.cl

Abstract

Automatic synthesis of software from specification is one of the classic problems in computer science. In the last decade, significant advances have been made in the synthesis of programs from specifications expressed in Linear Temporal Logic (LTL). LTL synthesis technology is central to a myriad of applications from the automated generation of controllers for Internet of Things devices, to the synthesis of control software for robotic applications. Unfortunately, the number of existing tools for LTL synthesis is limited, and using them requires specialized expertise. In this paper we present SynKit, a tool that offers LTL synthesis as a service. SynKit integrates a RESTful API and a web service with an editor, a solver, and a strategy visualizer.

1 Introduction

Automatic synthesis of software from specification is a classic problem in computer science that dates back to Church in 1957. Synthesis is a hard problem that has been well-studied, and no efficient solution exists in the general case. In the context of constructing strategies for reactive systems, Pnueli and Rosner [1989] proposed Linear Temporal Logic (LTL) synthesis where the specification is expressed in LTL. In the last decade, there have been significant algorithmic advances in LTL synthesis. In particular, so-called bounded synthesis techniques transform the problem into a game, and limit the search for solutions to spaces of bounded size [Kupferman and Vardi, 2005; Schewe and Finkbeiner, 2007]. Bounded synthesis was a major breakthrough in the development of practical algorithms that we can see in most modern tools.

The synthesis research field is rapidly evolving with the introduction of richer models that account for, e.g., environment assumptions in the model and quality of the solutions (c.f. [Chatterjee and Henzinger, 2007; Almagor et al., 2016]). The annual SYNTCOMP competition emerged in 2015 in an effort to standardize models and stimulate progress in the field. Yet, we find that there still exist important barriers that make it difficult for practitioners to adopt state-of-the-art synthesis tools in their workflow. Our purpose with SynKit is to reduce these barriers by offering LTL synthesis as a service.

1.1 Synthesis of LTL Specifications

LTL is a modal logic with standard logical connectives and temporal operators next (□) and until (U). Intuitively, LTL formula □α denotes that α has to hold in the next time step, and α U β denotes that α has to hold until β holds. Temporal operators eventually (□α := T U α), always (□α := ¬□¬α), and others can be derived from the basic operators. The truth of an LTL formula is evaluated over infinite traces.

Following the notation in [Camacho et al., 2018c], an LTL specification is a tuple \( \langle X, Y, \varphi \rangle_s \), where \( X \) and \( Y \) are disjoint sets of variables and \( \varphi \) is an LTL formula over \( X \cup Y \). LTL specifications are usually interpreted as two-player games, where the environment player controls \( X \) and the agent player controls \( Y \). In each turn, players select a subset of the variables they control. A play is an infinite sequence \( w = (x_1 \cup y_1)(x_2 \cup y_2) \cdots \) of subsets of \( X \cup Y \). The play is winning for the agent if \( w \) satisfies \( \varphi \). LTL realizability consists of determining if the agent player has a winning strategy for the game, and LTL synthesis is the problem of computing one such strategy. The order of turn taking is important, and is indicated by the semantics, \( s \). If the agent plays first, then \( s = \text{“Moore”} \); otherwise, \( s = \text{“Mealy”} \) (e.g. [Ehlers, 2011]).

Example We want a smart home system to turn the lights off if the room is not occupied, and to turn them on when someone enters the room. Sensors indicate when the room is occupied by activating event variable \( x \). Lights are turned...
on when signal $y$ is active. This is captured by LTL specification $\{x\}, \{y\}, \square(x \leftrightarrow y))_{\text{Mealy}}$. The winning strategy outputs $\{y\}$ if the environment selected $\{x\}$ in the same time step, and $\emptyset$ otherwise. This implements the desired behaviour.

2 LTL Synthesis as a Service

The main advantage of LTL synthesis over traditional programming is that the code that is produced is correct by construction, and programming is reduced to the task of declaring and maintaining a specification. LTL synthesis is potentially useful to generate controllers for robots, Internet of Things devices, and other applications. Unfortunately, the number of existing tools for LTL synthesis is limited, and often devoted to academic use. Practitioners that want to explore the benefits of LTL synthesis may experience a number of difficulties along the way. These include but are not limited to:

- Installation: most existing synthesis tools do not work on all platforms and require a number of dependencies.
- Scalability: synthesis of hard specifications requires a significant amount of computational resources.
- Learning curve: Most existing LTL synthesis tools were designed for academic use, and often have tunable parameters whose purpose and effect in the performance of the tool is not trivial for an non-expert user to master.
- Verification and debugging of specification: LTL is a compelling language to express temporal constraints, but design of correct specifications can be challenging.
- Strategy export: not all synthesis tools export synthesized strategies in a widely used standard format.

The purpose of SynKit is to mitigate for the difficulties listed above, and facilitate rapid prototyping and synthesis of controllers. Most similar to our tool is the online demo of Acacia$^+$ [Bohy et al., 2012], accessible through the website of the first author, which produces controllers in Verilog from a given specification in LTL or TLSF [Jacobs et al., 2016].

3 Functionality

SynKit determines realizability of LTL specifications, and returns a winning strategy if one exists. Its functionalities are accessible via web service (c.f. Figure 1) and also via API. For specifications that are not realizable, SynKit computes a certificate of unrealizability. Unrealizability certificates are strategies for the environment that prevent the agent from realizing the specification. They are useful for understanding why the specification is unrealizable – and, in particular, to debug mistakes in the declaration of realizable specifications.

SynKit implements the algorithms for LTL realizability and synthesis in [Camacho et al., 2018c], building on our preliminary work in [Camacho et al., 2018b]. Algorithms for finite LTL synthesis in [Camacho et al., 2018a] are also implemented. LTL realizability and synthesis is reduced to automata games, that are solved using fully observable nondeterministic (FOND) planning machinery. In the following, we discuss the most relevant functionality components.

4 Summary and Future Work

We presented SynKit, an LTL synthesis tool that facilitates rapid prototyping and debugging of LTL specifications. It can be used to generate controllers for robots, Internet of Things devices, and other applications. With LTL synthesis, programs are constructed and maintained by updating the specification; code is generated automatically and is correct by construction. In future work we plan to improve functionality of the web service and API.
References


