SynKit: Finite 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, and Chilean Center for Semantic Web Research
[†]{acamacho,sheila}@cs.toronto.edu, [⋆]cjmuise@mit.edu, [‡]jabaier@ing.puc.cl

Abstract

Automatic synthesis of software from specification is one of the classical problems in computer science. Recent research has explored the use of *finite linear temporal logic* ($\mathsf{LTL_f}$) as a specification language. Engineers, researchers, and practitioners who wish to explore $\mathsf{LTL_f}$ synthesis must overcome several barriers, including the lack of convenient tools to synthesize programs. In this paper we present *SynKit*, a web service that provides an $\mathsf{LTL_f}$ synthesis capability. SynKit aims to simplify the task of synthesizing programs and debugging specifications. Offered as a web service, it is very accessible and does not require installation. SynKit integrates an editor, a solver, and a strategy visualizer.

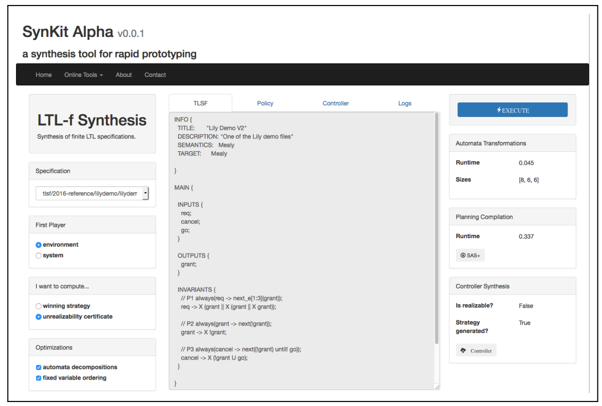

Figure 1: Capture of SynKit web service for $\mathsf{LTL_f}$ synthesis.

Introduction

Automated synthesis of software from specification was proposed by Church in 1957, and is a well-studied problem. In the context of constructing strategies for reactive systems, Pnueli and Rosner (1989) adopted *Linear Temporal Logic* (LTL) as the specification language. LTL is a modal logic with logical connectives ($\vee, \neg$) and temporal operators *next* ($\bigcirc$) and *until* (U). Intuitively, $\bigcirc\alpha$ denotes that LTL formula α must hold in the next time step, and $\alpha \mathsf{U} \beta$ denotes that α must hold until β holds. Other operators such as *always* ($\Box$) and *eventually* ($\Diamond$) can be defined in terms of $\bigcirc$ and U. For many years, the 2-EXP-hardness of the problem limited LTL synthesis to LTL fragments for which synthesis could be performed in polynomial time (e.g. GR(1) (Piterman *et al.* 2006)). In the last decade, significant algorithmic advances have been made, many based on *bounded synthesis* techniques (Kupferman and Vardi 2005; Schewe and Finkbeiner 2007)

Synthesis of Finite LTL specifications

LTL synthesis is conventionally studied with LTL specifications denoting properties that are interpreted with respect to infinite realizations of a program. Nevertheless, fields such as automated planning have a long-standing tradition of employing LTL interpreted over *finite traces* (henceforth $\mathsf{LTL_f}$) to specify temporally extended objectives (e.g., (Bacchus and Kabanza 2000; Baier and McIlraith 2006; Gerevini *et al.* 2009; Bienvenu *et al.* 2011)). $\mathsf{LTL_f}$ has the same syntax as LTL, with the difference that the truth of $\mathsf{LTL_f}$ formulae is evaluated over finite traces.

Recently, De Giacomo and Vardi (2015) studied the complexity of synthesis for $\mathsf{LTL_f}$ specifications, and determined the problem to be 2EXP-complete. Following the notation in (Camacho *et al.* 2018a; 2018c), an $\mathsf{LTL_f}$ specification is a triplet $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_s$, where $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of variables and φ is an $\mathsf{LTL_f}$ formula over $\mathcal{X} \cup \mathcal{Y}$. The synthesis problem can be characterized as a two-player game where the *environment* player controls $\mathcal{X}$ and the *agent* player controls $\mathcal{Y}$. In each turn, players select a subset of the variables they control. A *play* yields a sequence $w = (x_1 \cup y_1)(x_2 \cup y_2)\cdots$ of subsets of $\mathcal{X} \cup \mathcal{Y}$. The play is *winning* for the agent if w has a finite prefix, say $(x_1 \cup y_1)\cdots(x_n \cup y_n)$, that satisfies φ. $\mathsf{LTL_f}$ realizability determines whether the agent player has a winning strategy for the game; $\mathsf{LTL_f}$ synthesis is the problem of computing one such strategy. The order of turn-taking is relevant, and is indicated with the *semantics* of the game, s: if the agent plays first, then $s =$ "Moore"; otherwise, $s =$ "Mealy". With Moore semantics, the finite prefix can have length zero.

Example Consider a variant of the *visit-all* planning domain in which the agent can visit any cell c_i at a time, denoted by the $\mathsf{LTL_f}$ formula $\varphi_{\mathtt{agt}} := \Box \bigwedge_i (y_i \rightarrow \neg\bigcirc\neg y_i) \wedge \bigwedge_i \Box((\neg y_i \wedge \bigcirc y_i) \rightarrow \bigwedge_{j \neq i}(y_j \leftrightarrow \bigcirc y_j))$. Intuitively, variable y_i denotes that cell c_i has been visited. Initially, all cells are unvisited: $\varphi_{\mathcal{I}} := \bigwedge_i \neg y_i$. At each timestep, some cells may be poisoned, indicated by the truth of uncontrollable variables x_i.

The objective of the agent is to avoid poisoned cells, and to visit cells until all unvisited cells are poisoned. This is captured by LTL$_\mathsf{f}$ formula $\varphi_{\mathcal{G}} := \Box(\bigwedge_i(\neg y_i \wedge \bigcirc y_i) \rightarrow \neg\bigcirc x_i) \wedge \Diamond\bigwedge_i(\neg y_i \rightarrow x_i)$. The problem is modeled with specification $\langle \{x\}_i, \{y\}_i, \varphi_{\mathcal{I}} \wedge \varphi_{\mathtt{agt}} \wedge \varphi_{\mathcal{G}}\rangle_{\text{Mealy}}$. Winning strategies visit unpoisoned cells that are unvisited until the LTL specification is realized. $\square$

To the best of our knowledge, only two LTL$_\mathsf{f}$ synthesis tools exist to date: *Syft* (Zhu *et al.* 2017) and *SynKit* (Camacho *et al.* 2018a). Both reduce synthesis to solving a game played on finite-state automata. The first tool, *Syft*, solves the game using BDDs. The second tool, integrated into the SynKit web service presented in this paper, solves the game via reductions to *Fully Observable Non-Deterministic* (FOND) planning. It includes a facility to compute *certificates of unrealizability* as proposed in (Camacho *et al.* 2018a). These are environment strategies that prevent the agent from realizing the specification, and serve to explain why the specification is unrealizable.

Even though LTL$_\mathsf{f}$ synthesis is 2EXP, and therefore as hard as LTL synthesis, algorithms for LTL$_\mathsf{f}$ synthesis appear to be more scalable in practice (cf. (Zhu *et al.* 2017)). Due to its recent introduction, LTL$_\mathsf{f}$ synthesis is not as popular as LTL synthesis. One of our objectives with SynKit is to enable practitioners to explore LTL$_\mathsf{f}$ synthesis. Similar tools exist for planning – e.g. *planning.domains* (Muise 2016) – and LTL synthesis – e.g. the Acacia$^+$ online demo.

Finite LTL Synthesis as a Service

Non-expert users that want to use LTL$_\mathsf{f}$ synthesis tools may experience a number of difficulties, e.g.:

- Installation: existing implementations do not work on all platforms, and require a number of dependencies to transform LTL$_\mathsf{f}$ into automata and solve automata games.
- Computational resources: synthesis of hard specifications requires a significant amount of computational resources.
- Learning curve: the user not familiar with LTL$_\mathsf{f}$ and LTL$_\mathsf{f}$ synthesis may experience difficulties in writing specifications and in correctly setting tool parameters.
- Verification and debugging of specification: While LTL$_\mathsf{f}$ is a natural formalism for expressing temporal constraints, it can be hard to capture desired behaviour. Easy verification of solutions and easy debugging of specifications are desirable features of a synthesis tool.
- Strategy export: synthesized strategies have limited use if not exported in a widely accepted standard format.

The purpose for releasing a tool for LTL$_\mathsf{f}$ synthesis as a service is to relief practitioners from these pains, allow for rapid prototyping and synthesis of controllers, and at the end contribute towards adoption of synthesis technology.

Functionality

SynKit is a web service for LTL and LTL$_\mathsf{f}$ synthesis.[1] Currently, SynKit implements the algorithms presented in (Camacho *et al.* 2018a) for LTL$_\mathsf{f}$ synthesis and certificate generation, and the algorithms presented in (Camacho *et al.* 2018b; 2018c) for LTL synthesis. In the following, we discuss its basic functionality.

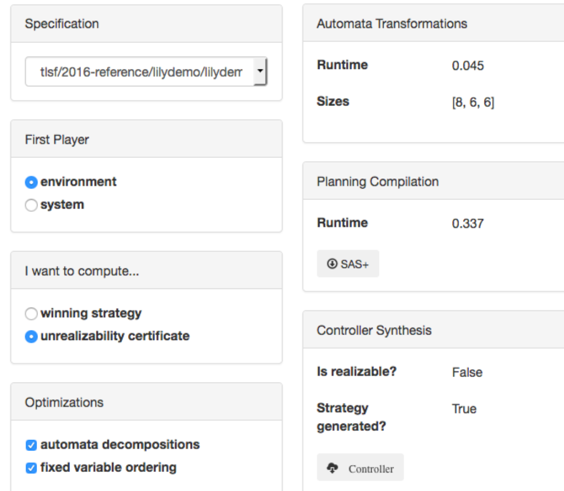

Figure 2: Detail of the options and results panels.

Specifications Editor The user can select one of the preloaded benchmarks, or write a custom specification in TLSF format (Jacobs *et al.* 2016).

Strategy Generation A graphical menu lets the user generate winning strategies, or unrealizability certificates (Figure 2). The former option computes a winning strategy if the specification is realizable, or returns that none exists. The latter option computes a certificate of unrealizability if the specification is not realizable, or returns that none exists. Either option determines realizability of the specification.

An optional optimization mode augments the scalability of automata transformations by decomposing the LTL$_\mathsf{f}$ formula and transforming it into multiple automata. However, the resulting game played on multiple automata is not necessarily easier to solve in practice. Another optimization mode forces a fixed action order in the compiled FOND problems.

Strategy Visualization and File Export Compiled FOND problems, and synthesized policies that are solution to those problems can be downloaded.

RESTful API Besides the graphical interface, basic functionalities are accessible via API.

Summary and Future Work

Synthesizing software from specification reduces programming to the task of designing and maintaining specifications. We presented SynKit, a web service that offers LTL and LTL$_\mathsf{f}$ synthesis as a service. SynKit facilitates rapid prototyping and debugging of LTL$_\mathsf{f}$ (and LTL) specifications, all in an easy-to-use toolkit. Going forward, we plan to improve the RESTful API, and to enable the export of synthesized controllers in Verilog format.

[1] SynKit is available through the first author's webpage, and: `http://www.cs.toronto.edu/~acamacho/synkit`

References

Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

Jorge A. Baier and Sheila A. McIlraith. Planning with temporally extended goals using heuristic search. In *Proceedings of the 16th International Conference on Automated Planning and Sched. (ICAPS)*, pages 342–345, 2006.

Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175(7–8):1308–1345, 2011.

Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Finite LTL synthesis as planning. In *Proceedings of the 28th International Conference on Automated Planning and Sched. (ICAPS)*, 2018. To appear.

Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Synthesizing controllers: On the correspondence between LTL synthesis and non-deterministic planning. In *Advances in Artificial Intelligence – Proceedings of the 31st Canadian Conference on Artificial Intelligence*, pages 45–59, 2018.

Alberto Camacho, Christian Muise, Jorge A. Baier, and Sheila A. McIlraith. LTL realizability via safety and reachability games. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018. To appear.

Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic, Cornell University 1957*, 1:3–50, 1957.

Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1558–1564, 2015.

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Swen Jacobs, Felix Klein, and Sebastian Schirmer. A high-level LTL synthesis format: TLSF v1.1. In *Proceedings of the 5th Workshop on Synthesis (SYNT)*, pages 112–132, 2016.

Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 531–542, 2005.

Christian Muise. Planning.Domains. In *The 26th International Conference on Automated Planning and Scheduling - Demonstrations*, 2016.

Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation,*, pages 364–380, 2006.

Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 179–190, 1989.

Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Proceedings of the 5th International Symposium on on Automated Technology for Verification and Analysis (ATVA)*, pages 474–488, 2007.

Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTLf synthesis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1362–1369, 2017.