SynKit: Finite LTL synthesis as a service

Alberto Camacho†, Christian Muisse*, 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, ‡jabaiер@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 (LTLf) as a specification language. Engineers, researchers, and practitioners who wish to explore LTLf 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 LTLf 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.

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 (\( \lor \), \( \land \)) and temporal operators next (\( \diamond \)) and until (\( \mathcal{U} \)). Intuitively, \( \diamond \alpha \) denotes that LTL formula \( \alpha \) must hold in the next time step, and \( \alpha \mathcal{U} \beta \) denotes that \( \alpha \) must hold until \( \beta \) holds. Other operators such as always (\( \square \)) and eventually (\( \lozenge \)) can be defined in terms of \( \diamond \) and \( \mathcal{U} \). 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 LTLf) to specify temporally extended objectives (e.g., (Bacchus and Kabanza 2000; Baier and McIlraith 2006; Gerevini et al. 2009; Bienvenu et al. 2011)). LTLf has the same syntax as LTL, with the difference that the truth of LTLf formulae is evaluated over finite traces.

Recently, De Giacomo and Vardi (2015) studied the complexity of synthesis for LTLf specifications, and determined the problem to be 2EXP-complete. Following the notation in (Camacho et al. 2018a; 2018c), an LTLf specification is a triplet \( \langle X, Y, \varphi \rangle \), where \( X \) and \( Y \) are disjoint sets of variables and \( \varphi \) is an LTLf formula over \( X \cup Y \). The synthesis problem can be characterized as a two-player game 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 yields a sequence \( w = (x_1 \cup y_1) \cup (x_2 \cup y_2) \cdots \) of subsets of \( X \cup Y \). The play is winning for the agent if \( w \) has a finite prefix, say \( (x_1 \cup y_1) \cup (x_2 \cup y_2) \cdots \), that satisfies \( \varphi \). LTLf realizability determines whether the agent player has a winning strategy for the game; LTLf 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, \( \mathcal{S} \): 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 LTLf formula \( \varphi_{\mathcal{S}_{ag}} := \Box \bigwedge_i (y_i \rightarrow \neg c_i \land y_I (\bigwedge_i (\neg y_i \land c_i \land y_i)) \rightarrow \bigwedge_{j \neq i} (y_j \leftrightarrow c_i \land y_J_i)) \). Intuitively, variable \( y_i \) denotes that cell \( c_i \) has been visited. Initially, all cells are unvisited: \( \varphi_X := \bigwedge_i y_i \). At each timestep, some cells may be poisoned, indicated by the truth of uncontrollable variables \( x_i \).
SynKit is a web service for LTL contribute towards adoption of synthesis technology. Rapid prototyping and synthesis of controllers, and at the end service is to relief practitioners from these pains, allow for macho rently, SynKit implements the algorithms presented in (Cama-
cho 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 LTLf synthesis is 2EXP, and therefore as hard as LTL synthesis, algorithms for LTLf synthesis appear to be more scalable in practice (cf. (Zhu et al. 2017)). Due to its recent introduction, LTLf synthesis is not as popular as LTL synthesis. One of our objectives with SynKit is to enable practitioners to explore LTLf 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 LTLf 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 LTLf 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 LTLf and LTL synthesis may experience difficulties in writing specifications and in correctly setting tool parameters.
- Verification and debugging of specification: While LTLf 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 LTLf 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 LTLf synthesis.1 Currently, SynKit implements the algorithms presented in (Camacho et al. 2018a) for LTLf synthesis and certificate gen-

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

Figure 2: Detail of the options and results panels.

eration, and the algorithms presented in (Camacho et al. 2018b; 2018c) for LTL synthesis. In the following, we discuss its basic functionality.

Specifications Editor The user can select one of the pre-loaded benchmarks, or write a custom specification in TLF 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 LTLf 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 LTLf synthesis as a service. SynKit facilitates rapid prototyping and debugging of LTLf (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.
References


