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, [‡]jabaier@ing.puc.cl

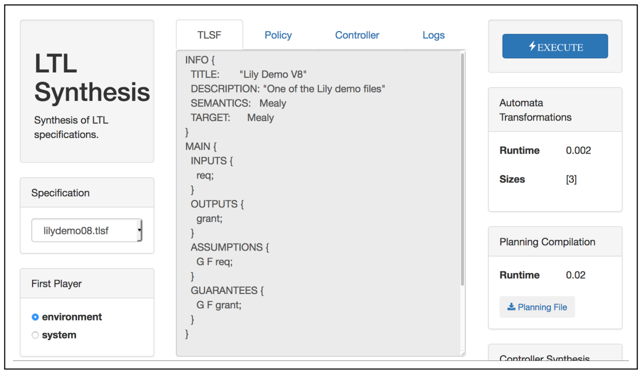
Abstract

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.



Specification: lilydemo08.tlsf

First Player: [✓ environment] system

```
INFO {
 TITLE:     "Lily Demo V8"
 DESCRIPTION: "One of the Lily demo files"
 SEMANTICS:  Mealy
 TARGET:     Mealy
}
MAIN {
 INPUTS {
  req;
 }
 OUTPUTS {
  grant;
 }
 ASSUMPTIONS {
  G F req;
 }
 GUARANTEES {
  G F grant;
 }
}
```

Automata Transformations

Runtime	0.002
Sizes	[3]

Planning Compilation

Runtime	0.02

Figure 1: Capture of the SynKit tool for LTL synthesis.

1.1 Synthesis of LTL Specifications

LTL is a modal logic with standard logical connectives and temporal operators *next* ($\bigcirc$) and *until* (U). Intuitively, LTL formula $\bigcirc\alpha$ denotes that α has to hold in the next time step, and $\alpha \mathsf{U} \beta$ denotes that α has to hold until β holds. Temporal operators *eventually* ($\Diamond\alpha := \top \mathsf{U} \alpha$), *always* ($\Box\alpha := \neg\Diamond\neg\alpha$), 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 $\langle \mathcal{X}, \mathcal{Y}, \varphi \rangle_s$, where $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of variables and φ is an LTL formula over $\mathcal{X} \cup \mathcal{Y}$. LTL specifications are usually interpreted as two-player games, 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* is an infinite 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 satisfies φ. 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 $s =$ "Moore"; otherwise, $s =$ "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 $\langle\{x\},\{y\},\square(x \leftrightarrow y)\rangle_{\text{Mealy}}$. 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.

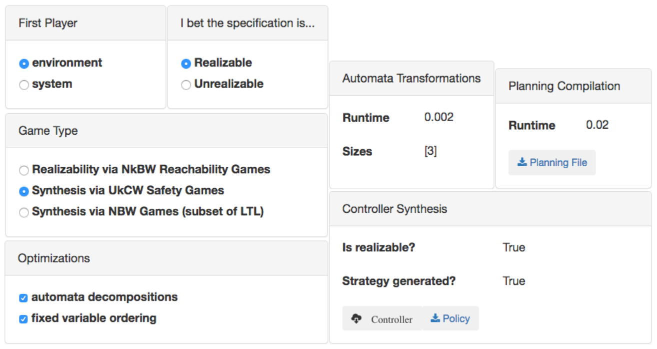
The purpose of SynKit is to mitigate for the difficulties listed above, and facilitate rapid prototyping and synthesis of controllers. Most similar to our tool is the online demo of Acacia$^+$ [Bohy *et al.*, 2012], accessible through the website of the first author, which produces controllers in Verilog 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.[1] 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 non-deterministic (FOND) planning machinery. In the following, we discuss the most relevant functionality components.

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



First Player: [✓ environment] system

I bet the specification is...: [✓ Realizable] Unrealizable

Game Type: Realizability via NkBW Reachability Games [✓ Synthesis via UkCW Safety Games] Synthesis via NBW Games (subset of LTL)

Optimizations: [✓ automata decompositions] [✓ fixed variable ordering]

Automata Transformations
Runtime 0.002
Sizes [3]

Planning Compilation
Runtime 0.02

Controller Synthesis
Is realizable? True
Strategy generated? True

Figure 2: Detail of the options (left) and results (right) panels.

Specifications Editor The user can write a custom LTL specification in the TLSF format, adopted by SYNTCOMP, or select one of the preloaded SYNTCOMP benchmarks.

Realizability Guess: The user has to guess whether the specification is realizable or unrealizable. If the guess is wrong, then SynKit's search for strategies will not terminate. This *incompleteness* is common to many tools based on bounded synthesis (e.g. Acacia$^+$), and can be avoided by interleaving two searches with different guesses.

Game Reductions: The user can choose reductions to safety games (i.e., games played on UkCW automata), or reachability games (i.e., games played on NkBW automata). Reductions to safety games produce winning strategies or certificates of unrealizability. Reductions to reachability games determine realizability, and do not return a strategy.

Optimizations: Reductions to automata games involve transfoming the LTL specification formula into automata. The user can choose to perform decompositions into multiple automata for better scalability in the automata transformations of the LTL formula. However, the resulting game played on multiple automata is not necessarily easier to solve in practice. In addition, the user can force the use of a fixed ordering of the actions in the compilations of games to FOND planning.

Strategy Visualization and File Export: Synthesized strategies can be exported in Verilog format. To facilitate rapid design of correct specifications, the tool allows for inspection of the policy graphs obtained from the solution to reduced FOND problems. FOND problems can also be downloaded.

RESTful API: The current implementation includes a RESTful API with basic functionalities that will be extended.

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

[Almagor *et al.*, 2016] Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *Journal of the ACM*, 63(3):24:1–24:56, 2016.

[Bohy *et al.*, 2012] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*, pages 652–657, 2012.

[Camacho *et al.*, 2018a] 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 Scheduling (ICAPS)*, 2018. To appear.

[Camacho *et al.*, 2018b] 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.

[Camacho *et al.*, 2018c] Alberto Camacho, Christian J. 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.

[Chatterjee and Henzinger, 2007] Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 261–275. Springer, 2007.

[Church, 1957] 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.

[Ehlers, 2011] Rüdiger Ehlers. Experimental aspects of synthesis. In *Proceedings of the International Workshop on Interactions, Games and Protocol (IWIGP)*, pages 1–16, 2011.

[Jacobs *et al.*, 2016] 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.

[Kupferman and Vardi, 2005] 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.

[Pnueli and Rosner, 1989] 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 (POPL)*, pages 179–190, 1989.

[Schewe and Finkbeiner, 2007] 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.