Reactive Synthesis of Linear Temporal Logic on Finite Traces

Shufang Zhu
shufang.zhu@cs.ox.ac.uk
University of Oxford
UK

ABSTRACT

Reactive synthesis holds the promise of automatically generating a verifiably correct program from a high-level specification. In this work, we focus on reactive synthesis problems of Linear Temporal Logic on finite traces (LTL$_f$) and present an evolving journey. We first present the advances in LTL$_f$ synthesis. Then we show that when concerning environment specifications expressed in LTL, we can avoid the detour to the challenging problem of LTL synthesis and keep the simplicity of LTL$_f$ synthesis in interesting cases. Furthermore, we can still partially avoid this detour even when the environment specifications are expressed in general LTL.

KEYWORDS

LTL$_f$, reactive synthesis, synthesis with environment specifications

1 INTRODUCTION

Reactive synthesis promises to automatically generate a verifiably correct program from a high-level specification [24]. A popular such specification language is Linear Temporal Logic (LTL) [23]. Unfortunately, synthesizing programs from *general* LTL formulas, which rely on first constructing a game arena and then solving the game, remains challenging [17, 22]. Nevertheless, the synthesis problem of a finite trace variant of LTL, which is LTL$_f$ [20], has shown to be much simpler than LTL synthesis [15]. The key idea is that synthesizing LTL$_f$ formulas only involves games on finite traces instead of infinite traces as for LTL, though both problems share the same worst-case complexity of 2EXPTIME-complete.

In this paper, we will review an evolving journey motivated by this idea. We start from an attempt to devise a symbolic LTL$_f$ synthesis framework [29], which consists of a backward reachability game on the constructed Deterministic Finite Automaton (DFA) of the corresponding LTL$_f$ formula and has demonstrated its significant efficiency in various application scenarios. Then, the journey evolves into a forward LTL$_f$ synthesis technique that synthesizes a strategy while constructing the DFA, thus being possible to avoid the 2EXPTIME worst-case complexity [14**?**]. Next, we study LTL$_f$ synthesis under environment specifications, which are constraints on the environment that rule out certain environment behaviours [1, 7]. A key observation is that even if we consider an agent with LTL$_f$ tasks on finite traces, environment specifications need to be expressed over infinite traces since accomplishing the agent tasks may require an unbounded number of environment actions [1, 7]. While a naive solution to LTL$_f$ synthesis under environment specifications expressed in LTL would be reducing the problem to LTL synthesis, which remains challenging [1, 7], we show in this paper that we can avoid the detour to LTL synthesis and keep the simplicity of LTL$_f$ synthesis in interesting cases. More specifically, we consider the following certain environment specifications: safety [10], simple fairness and stability [27], and Generalized-Reactivity(1) [11]. Furthermore, we show that even when the environment specifications are expressed in general LTL, we can still partially avoid the full detour to LTL synthesis [12].

2 PRELIMINARIES

2.1 LTL and LTL$_f$

LTL is one of the most popular specification languages for expressing temporal properties [23].Given a set $\mathcal{P}$ of propositions, the syntax of LTL formulas is defined as:

$$\varphi ::= p \mid (\varphi_1 \wedge \varphi_2) \mid (\neg\varphi) \mid (\circ\varphi) \mid (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

where $p \in \mathcal{P}$ is an *atom*. $\circ$ (Next), $\mathcal{U}$ (Until) are temporal connectives. We use the abbreviations $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$, for temporal connectives $\Diamond$ (Eventually) and $\Box$ (Always).

A *trace* $\pi = \pi_0\pi_1\ldots$ is a sequence of propositional interpretations (sets), where $\pi_m \in 2^{\mathcal{P}}$ ($m \geq 0$) is the m-th interpretation of π, and $|\pi|$ represents the length of π. Trace π is an *infinite* trace if $|\pi| = \infty$, which is formally denoted as $\pi \in (2^{\mathcal{P}})^{\omega}$. Otherwise π is a *finite* trace, denoted as $\pi \in (2^{\mathcal{P}})^{*}$. LTL formulas are interpreted over infinite traces. Given an infinite trace π and an LTL formula φ, we inductively define when φ is *true* in π at instant i ($i \geq 0$), written $\pi, i \models \varphi$, as follows:

- $\pi, i \models p$ iff $p \in \pi_i$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$, iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \neg\varphi$, iff $\pi, i \not\models \varphi$;
- $\pi, i \models \circ\varphi$, iff $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$, iff $\exists k.k \geq i$ such that $\pi, k \models \varphi_2$, and $\forall j.i \leq j < k, \pi, j \models \varphi_1$.

We say π *satisfies* φ, written as $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

LTL$_f$ is a variant of LTL interpreted over *finite traces* instead of infinite traces [20]. The syntax of LTL$_f$ is exactly the same as LTL, but they have different semantics. We define $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators:

- $\pi, i \models \circ\varphi$ iff $i < \mathit{last}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff $\exists j.i \leq j \leq \mathit{last}(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k.i \leq k < j$ we have $\pi, k \models \varphi_1$.

where we denote the last position in the finite trace π by $\mathit{last}(\pi)$. In addition, we define the *weak next* operator $\bullet$ as the abbreviation of $\bullet\varphi \equiv \neg\circ\neg\varphi$. Note that, on finite traces, $\neg\circ\varphi \equiv \circ\neg\varphi$ **does not** hold, instead it holds that $\neg\circ\varphi \equiv \bullet\neg\varphi$. We say that a trace *satisfies* an LTL$_f$ formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

2.2 Reactive Synthesis

Reactive synthesis can be viewed as a game between the *environment* and the *agent*, contrasting each other by controlling two disjoint sets of variables $\mathcal{X}$ and $\mathcal{Y}$, respectively. Reactive synthesis aims to synthesize an agent strategy such that no matter how the environment behaves, the combined trace from two players

satisfies desired properties [24]. An environment strategy is a function $\gamma : (2^{\mathcal{Y}})^+ \rightarrow 2^{\mathcal{X}}$, and an agent strategy is a function $\sigma : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$. A trace $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1) \cdots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$, is *compatible* with an environment strategy γ if $\gamma(Y_0 Y_1 \dots Y_i) = X_i$ for every $i \geq 0$. A trace π being compatible with an agent strategy σ is defined analogously. Sometimes, we write $\sigma(\pi^k)$ instead of $\sigma(X_0 X_1 \cdots X_k)$ for simplicity. We denote the unique infinite sequence that is compatible with γ and σ as $\pi(\sigma, \gamma)$.

3 LTL$_f$ SYNTHESIS

The *synthesis problem* for an agent task specified as an LTL$_f$ formula φ is to find an agent strategy $\sigma : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ such that for every environment strategy $\gamma : (2^{\mathcal{Y}})^+ \rightarrow 2^{\mathcal{X}}$, there exists $k \geq 0$, chosen by the agent, such that the finite trace $\pi(\sigma, \gamma)^k \models \varphi$ i.e., φ is agent realizable. One can solve LTL$_f$ synthesis by translating φ to an LTL formula ψ and then solving LTL synthesis on ψ [29].

3.1 Backward LTL$_f$ Synthesis

The first solution to LTL$_f$ synthesis problem was based on a reduction to reachability game [15], which proceeds as follows: build the corresponding DFA of the agent task φ, solve the reachability game over it, and hence return the winning strategy for the agent. However, the size of the constructed DFA can be, in the worst case, doubly-exponential in the size of the formula. In order to combat this difficulty, we proposed a symbolic LTL$_f$ synthesis framework, which essentially represents the DFA as Boolean formulas [29]. Hence, the reachability game can be solved on the symbolic DFA. Moreover, the symbolic DFA is practically represented in Binary Decision Diagrams (BDDs) [6]. The reachability game is performed on BDDs through a fixpoint computation. The winning strategy is abstracted utilizing Boolean synthesis [18]. This symbolic LTL$_f$ synthesis framework has shown outperformance compared to the explicit approach described in [15] and the solution of reducing to LTL synthesis. Furthermore, it also has been integrated into state-of-the-art LTL$_f$ synthesizers, e.g., Lisa [4] and Lydia [13]. The main difficulty of this synthesis framework is that it requires computing the entire DFA of the LTL$_f$ specification, hence cannot avoid the worst-case 2EXPTIME blowup. Hence, even though the backward fixpoint computation can be performed symbolically, enabling scalable performance, the DFA construction step can become a significant bottleneck [29].

3.2 Forward LTL$_f$ Synthesis

In order to combat the worst-case 2EXPTIME blowup, we investigated LTL$_f$ forward synthesis adopting an AND-OR graph search that is able to create on-the-fly the DFA corresponding to the LTL$_f$ specification [14, 26]. This technique avoids a detour to automata theory and instead builds directly deterministic transitions from a current state. In particular, this technique exploits LTL formula progression [2, 16] to separate what happens now (label) and what should happen next accordingly (successor state). Crucially, in [14], we exploit the structure that formula progression provides to branch on propositional formulas (representing several evaluations) instead of individual evaluations as in [26]. This drastically reduces the branching factor of the AND-OR graph to be searched (recall that in LTL$_f$ synthesis, both the agent choices and the environment choices can be exponentially many). More specifically, we label transitions/edges with propositional formulas on propositions controlled by the agent (for OR-nodes) and by the environment (for AND-nodes). Every such propositional formula captures a set of evaluations leading to the same successor node. We leverage Knowledge Compilation (KC) techniques, and in particular Sentential Decision Diagrams (SDDs) [9], to effectively generate such propositional formulas for OR-nodes and AND-nodes, and thus reduce the branching factor of the search space.

4 LTL$_f$ SYNTHESIS UNDER ENVIRONMENT SPECIFICATIONS

In standard synthesis, the agent assumes the environment to be free to choose an arbitrary move at each step, but in reality, often the agent has some knowledge of how the environment works, which it can exploit in order to enforce the goal, specified as an LTL$_f$ formula φ. Here, we specify the environment behaviour by an LTL formula *env* and call it *environment specification* [1].

Given an LTL formula *env*, we say that an agent strategy (resp. environment strategy) *enforces* φ, written $\sigma \triangleright env$ (resp., $\gamma \triangleright env$), if for every environment strategy γ (resp. agent strategy σ), we have $\pi(\sigma, \gamma) \models \varphi$. This LTL formula *env*, in particular, specifies the set of environment strategies that enforces *env*. As usual, we require that *env* must be *environment realizable*, i.e., the set of environment strategies that enforce *env* is nonempty.

The problem of *synthesis under environment specifications* is to find an agent strategy σ such that

$$\forall \gamma \triangleright env, trace(\sigma, \gamma)^k \models \varphi \text{ for some } k \in \mathcal{N}.$$

As shown in [1], this can be reduced to solving the synthesis problem for the implication $env \rightarrow ltl(\varphi)$, with $ltl(\varphi)$ being a suitable LTL$_f$-to-LTL transformation [20]. However, the algorithms available for LTL synthesis are much harder in practice than those for LTL$_f$ synthesis. In particular, the lack of efficient algorithms for the crucial step of automata determinization is a major obstacle for scalable implementations [17]. In spite of several advancements in synthesis, such as reducing to parity games [22], bounded synthesis based on solving iterated safety games [19], or recent techniques based on iterated FOND planning [8], LTL synthesis remains challenging.

In the following, we present the solutions of LTL$_f$ synthesis under LTL environment specifications for certain types of environment specifications, which allow us to avoid the detour to LTL synthesis and keep the simplicity of LTL$_f$ synthesis. Furthermore, we can still partially avoid this detour even when the environment specifications are expressed in general LTL.

4.1 Safety Environment Specifications

We consider here environment specifications that are safety properties. Intuitively, a *safety* property excludes traces whose "badness" follows from a finite prefix. We formally refer to these prefixes as bad prefixes. Consider a language $\mathcal{L} \subseteq (2^{\mathcal{P}})^\omega$ of infinite traces $\mathcal{P}$. A finite trace $h \in (2^{\mathcal{P}})^*$ is a *bad* prefix for $\mathcal{L}$ iff for all infinite traces $\pi \in (2^{\mathcal{P}})^\omega$, we have that $h \cdot \pi \notin \mathcal{L}$. A formula φ expresses a *safety property* iff every trace that violates φ has a bad prefix. One can write a safety environment specification either with Safety

LTL, a syntactic fragment of LTL or LTL_f in all prefix semantics, for more details of which, we refer to [3, 28] and [11], respectively. To solve the problem, we can proceed as follows. First, translate the safety environment specification *env* into a Deterministic Safety Automaton (DSA) S [28] and solve a safety game for the environment on S. By restricting S to the environment winning region, we can obtain all the environment strategies that can enforce *env*. Finally, we need to solve the reachability game over the product of the corresponding DFA of LTL_f formula φ and the restricted part of the DSA S [10], thus obtaining an agent winning strategy if there exists one.

4.2 Stability and Fairness Environment Specifications

We consider here two different basic forms of environment specifications: a basic form of fairness $\Box\Diamond\alpha$ (always eventually α) and a basic form of stability $\Diamond\Box\alpha$. Note that in both cases, the truth value of α is under the control of the environment. Hence, the environment specification *env* is indeed environment realizable, i.e., the set of environment strategies that enforce *env* is not empty. The key idea of solving such problems is integrating the environment specification as the winning condition of the reduced game between the environment and the agent [27]. We can solve the problem as follows. First, translate the LTL_f formula φ into a DFA D. Then, in case of fairness environment specifications, solve the fair DFA game on D, in which the environment (resp. the agent) winning condition is to remain in a region (resp., to avoid the region), where α holds infinitely often. Meanwhile, the accepting states are forever avoidable by applying a nested fixed-point computation on D. Similarly, in the case of stability environment specifications, solve the stable DFA game on D, in which the environment (resp. the agent) winning condition is to reach a region (resp., to avoid the region) where α holds forever. Meanwhile, the accepting states are forever avoidable by applying a nested fixed-point computation.

4.3 Generalized-Reactivity(1) Specifications

There have been great successes with LTL synthesis on the Generalized-Reactivity (1) [5], or GR(1), approach: focusing on a significant syntactic fragment of LTL that uses safety conditions to determine the possible transitions in a game between the environment and the agent, plus one powerful notion of fairness. We brought it together with the successes on LTL_f synthesis, devising an approach to solve LTL_f synthesis of agent task φ under GR(1) environment specification env_{gr1} [11]. In more detail, we first observe that the agent's goal is to satisfy $\neg env_{gr1} \vee \varphi$, while the environment's goal is to satisfy $env_{gr1} \wedge \neg\varphi$. Moreover, we know that φ can be represented by a DFA [20]. Then, focusing on the environment point of view, we show that the problem of LTL_f synthesis under GR(1) environment specification can be reduced into a GR(1) game, in which the game arena is the complement of the DFA for φ, i.e., a DSA expressing safety conditions, and env_{gr1} is the GR(1) winning condition. Since we want a winning strategy for the agent, we need to deal with the complement of the GR(1) game to obtain a winning strategy for the antagonist. We can also enrich the synthesis problem by adding safety conditions for both the environment and the agent, obtaining a highly expressive yet still scalable form of synthesis. Observe, though, that there is a difference between the safety conditions on the environment and those on the agent: the first must hold forever, while the second must hold until the agent task is terminated, i.e., the LTL_f formula φ is fulfilled. The enriched problem can also be reduced to a GR(1) game, where the game arena consists of the DFAs for the environment safety condition, the agent safety condition, and the agent task.

4.4 General LTL Environment Specifications

Regarding LTL_f synthesis with general LTL environment specifications, dealing with LTL synthesis seems to be unavoidable. Nevertheless, we developed a two-stage technique that maximizes the simplicity of LTL_f synthesis and mitigates the difficulty of LTL synthesis [12]. Intuitively, the two-stage technique first solely deals with the agent task, LTL_f formula φ, and thus confines the difficulty of handling the LTL environment specification *env* to the bare minimum in the second stage. In detail, the two-stage techniques proceed as follows:

- **Stage 1**: Build the DFA D of φ and solve the reachability game for the agent over D. If the agent has a winning strategy σ in D then the algorithm returns σ. Otherwise, continue to Stage 2.
- **Stage 2**: Perform the following steps:
  (1) Remove from D the agent winning region, obtaining D';
  (2) Build the Deterministic Parity Automaton (DPA) P of *env* and get the product of D' and P, obtaining a new DPA $A = D' \times P$, and solve the parity game for the environment over A;
  (3) If the agent has a winning strategy in A, then the synthesis problem is realizable and hence returns the agent winning strategy as a combination of the agent winning strategies in the two stages.

5 CONCLUSION

Reactive synthesis on LTL_f has been an exciting research problem. Our work on this problem spans from standard LTL_f synthesis to synthesis concerning environment specifications with efficient solution techniques. In the future, we would like to consider environment specifications expressed in different languages, e.g., PDDL [21], a popular specification language in planning. Furthermore, planning is highly related to synthesis, since both concern games between the environment and the agent. One advantage of utilizing PDDL is that the corresponding state space obtained from PDDL only leads to a single-exponential blowup [25] instead of double-exponential as LTL. An interesting question is how to integrate the synergies of PDDL better-expressed planning and LTL/LTL_f synthesis.

ACKNOWLEDGMENTS

We thank the contributions of all the co-authors (in the order of publications): Jianwen Li, Geguang Pu, Lucas M. Tabajara, Moshe Y. Vardi, Giuseppe De Giacomo, Antonio Di Stasio, Giuseppe Perelli, Shengping Xiao, Yingying Shi, Marco Favorito, Suguman Bansal and Yong Li. This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228).

REFERENCES

[1] Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. 2019. Planning under LTL Environment Specifications. In *ICAPS*. 31–39.

[2] F. Bacchus and F. Kabanza. 1998. Planning for Temporally Extended Goals. *Ann. Math. Artif. Intell.* 22, 1-2 (1998).

[3] Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi, and Shufang Zhu. [n. d.]. Compositional Safety LTL Synthesis. In *VSTTE*. 1–19.

[4] Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. 2020. Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications. In *AAAI*. 9766–9774.

[5] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3, 911–938.

[6] Randal E. Bryant. 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.* 24, 3 (1992), 293–318.

[7] Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. 2018. Finite LTL Synthesis with Environment Assumptions and Quality Measures. In *KR*. 454–463.

[8] Alberto Camacho, Christian J. Muise, and Jorge A. Baier and Sheila A. McIlraith. 2018. LTL Realizability via Safety and Reachability Games. In *IJCAI*. 4683–4691.

[9] A. Darwiche. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI*.

[10] Giuseppe De Giacomo, Antonio Di Stasio, Giuseppe Perelli, and Shufang Zhu. 2021. Synthesis with Mandatory Stop Actions. In *KR*. 237–246.

[11] Giuseppe De Giacomo, Antonio Di Stasio, Luca M. Tabajara, Moshe Y. Vardi, and Shufang Zhu. 2021. Finite-Trace and Generalized-Reactivity Specifications in Temporal Synthesis. In *IJCAI*.

[12] Giuseppe De Giacomo, Antonio Di Stasio, Moshe Y. Vardi, and Shufang Zhu. 2020. Two-Stage Technique for LTL_f Synthesis Under LTL Assumptions. In *KR*.

[13] Giuseppe De Giacomo and Marco Favorito. 2021. Compositional Approach to Translate LTLf/LDLf into Deterministic Finite Automata. In *ICAPS (to appear)*, Vol. 14.

[14] Giuseppe De Giacomo, Marco Favorito, Jianwen Li, Moshe Y. Vardi, Shengping Xiao, and Shufang Zhu. 2022. LTL_f Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In *IJCAI*. 2591–2598.

[15] Giuseppe De Giacomo and Moshe Y. Vardi. 2015. Synthesis for LTL and LDL on Finite Traces. In *IJCAI*.

[16] E. Allen Emerson. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*.

[17] Bernd Finkbeiner. 2016. Synthesis of Reactive Systems. In *Dependable Software Systems Engineering*. 72–98.

[18] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. 2016. BDD-Based Boolean Functional Synthesis. In *CAV*. 402–421.

[19] Carsten Gerstacker, Felix Klein, and Bernd Finkbeiner. 2018. Bounded Synthesis of Reactive Programs. In *ATVA*. 441–457.

[20] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*.

[21] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. *PDDL – The Planning Domain Definition Language – Version 1.2.* Technical Report. TR-98-003, Yale Center for Computational Vision and Contro.

[22] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. 2018. Strix: Explicit Reactive Synthesis Strikes Back!. In *CAV*, Hana Chockler and Georg Weissenbacher (Eds.). 578–586.

[23] Amir Pnueli. 1977. The Temporal Logic of Programs. In *FOCS*. 46–57.

[24] Amir Pnueli and Roni Rosner. 1989. On the Synthesis of a Reactive Module. In *POPL*.

[25] Jussi Rintanen. 2004. Complexity of Planning with Partial Observability. In *ICAPS*. 345–354.

[26] Shengping Xiao, Jianwen Li, Shufang Zhu, Yingying Shi, Geguang Pu, and Moshe Y. Vardi. 2021. On-the-fly Synthesis for LTL over Finite Traces. In *AAAI*. 6530–6537.

[27] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu, and Moshe Y. Vardi. 2020. LTL_f Synthesis with Fairness and Stability Assumptions. In *AAAI*. 3088–3095.

[28] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. A Symbolic Approach to Safety LTL Synthesis. In *HVC*. 147–162.

[29] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. Symbolic LTL_f Synthesis. In *IJCAI*. 1362–1369.