LTL_f Synthesis Under Environment Specifications (Short Paper)

Antonio Di Stasio

Sapienza University of Rome, Italy

Abstract

In this communication we present recent advances in the field of synthesis for agent goals specified in Linear Temporal Logic on finite traces under environment specifications. In synthesis, environment specifications are constraints on the environments that rule out certain environment behavior. To solve synthesis of LTL_f goals under environment specifications, we could reduce the problem to LTL synthesis. Unfortunately, while synthesis in LTL_f and in LTL have the same worst-case complexity (both are 2EXPTIME-complete), the algorithms available for LTL synthesis are much harder in practice than those for LTL_f synthesis. We report recent results showing that when the environment specifications are in form of fairness, stability, or GR(1) formulas, we can avoid such a detour to LTL and keep the simplicity of LTL_f synthesis. Furthermore, even when the environment specifications are specified in full LTL we can partially avoid this detour.

Keywords

Linear Temporal Logic on Finite Traces, Synthesis, Automata-Theoretic Approach

1. Introduction

Program synthesis is one of the most ambitious and challenging problem of CS and AI. Reactive synthesis is a class of program synthesis problems which aims at synthesizing a program for interactive/reactive ongoing computations [1, 2]. We have two sets of Boolean variables $\mathcal{X}$ and $\mathcal{Y}$, controlled by the environment and the agent, respectively, and a specification φ over $\mathcal{X} \cup \mathcal{Y}$ in terms of Linear Temporal Logic (LTL) [3]. The synthesis has to generate a program, aka a strategy, for the agent such that for every environment strategy the simultaneous execution of the two strategies generate a trace that satisfies φ [2, 4, 5].

Recently, the problem of reactive synthesis has been studied in the case the specification is expressed in LTL over finite traces (LTL_f) and its variants [6]. This logic is particularly fitting for expressing temporally-extended goals in Planning since it retains the fact that ultimately the goal must be achieved and the plan terminated [7].

In the classical formulation of the problem of reactive synthesis the environment is free to choose an arbitrary move at each step, but in AI typically we have a model of world, i.e., of the environment behavior, e.g., encoded in a planning domain [8, 9, 10], or more generally directly in temporal logic [11, 12, 13, 14]. In other words, we are interested in synthesis under environment specifications [15, 16, 17, 18, 19, 20, 21], which can be reduced to standard synthesis of the

Proceedings of the 23rd Italian Conference on Theoretical Computer Science, Rome, Italy, September 7-9, 2022
distasio@diag.uniroma1.it (A. Di Stasio)
0000-0001-5475-2978 (A. Di Stasio)

implication: $Env \rightarrow Goal$ where Env is the specification of the environment (the environment specification) and $Goal$ is the specification of the task of the agent [13, 16]. The agent has to realize its task $Goal$ only on those traces that satisfy the environment specification Env. Specifically, we focus on synthesis under environment specifications for LTL$_f$ goals. However, while it is natural to consider the task specification $Goal$ as an LTL$_f$ formula, requiring that also Env is an LTL$_f$ formula is often too strong, since accomplishing the agent task may require an unbounded number of environment moves and such number is decided by the agent that determines when its task is finished. As a result Env typically needs to be expressed over LTL not LTL$_f$ [17, 21]. So, even when focusing on LTL$_f$, what we need to study is the case where we have the task $Goal$ expressed in LTL$_f$ and the environment specification Env expressed in LTL.

One way to handle this case is to translate $Goal$ into LTL [22] and then do LTL synthesis for $Env \rightarrow Goal$, see e.g. [17]. But, while synthesis in LTL$_f$ and in LTL have the same worst-case complexity, being both 2EXPTIME-complete [2, 6], the algorithms available for LTL synthesis are much harder in practice than those for LTL$_f$ synthesis as experimentally shown [23, 24, 25].

For these reasons, several specific forms of LTL environment specifications have been considered. In this communication we reports recent results showing how to avoid the detour to LTL synthesis in cases where the environment specifications are in the form of fairness or stability [21], or GR(1) formulas [20], or specified in both LTL$_f$ (e.g., for representing nodeterministic planning domains or other safety properties) and LTL (e.g., for liveness/fairness), but separating the contributions of the two by limiting the second one as much as possible [18].

2. LTL and LTL$_f$

LTL is one of the most popular logics for temporal properties [3]. Given a set of propositions $\mathcal{P}$, the formulas of LTL are generated by the following grammar:

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

where $p \in \mathcal{P}$. We use common abbreviations for *eventually* $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ and *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

LTL formulas are interpreted over infinite traces $\pi \in (2^{\mathcal{P}})^\omega$. A *trace* $\pi = \pi_0\pi_1\ldots$ is a sequence of propositional interpretations (sets), where for every $i \geq 0$, $\pi_i \in 2^{\mathcal{P}}$ is the i-th interpretation of π. Intuitively, π_i is interpreted as the set of propositions that are $true$ at instant i. Given π, we define when an LTL formula φ *holds* at position i, written as $\pi, i \models \varphi$, inductively on the structure of φ, as follows:

- $\pi, i \models p$ iff $p \in \pi_i$ (for $p \in \mathcal{P}$);
- $\pi, i \models \varphi_1 \wedge \varphi_2$ 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 there exists $i \leq j$ such that $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

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 [22]. The syntax of LTL$_f$ is the same as the syntax of LTL. We define $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators we have:

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

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

3. LTL$_f$ Synthesis Under Environment Specifications

Let $\mathcal{X}$ and $\mathcal{Y}$ Boolean variables, controlled by the environment and the agent, respectively. An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, and an environment strategy is a function $\sigma_{env} : (2^{\mathcal{Y}})^+ \to 2^{\mathcal{X}}$. A *trace* is a sequence $(X_0 \cup Y_0)(X_1 \cup Y_1) \cdots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$. A trace $(X_i \cup Y_i)_i$ is consistent with an agent strategy σ_{ag} if $\sigma_{ag}(\epsilon) = Y_0$ and $\sigma_{ag}(X_0 X_1 \cdots X_j) = Y_{j+1}$ for every $j \geq 0$. A trace $(X_i \cup Y_i)_i$ is consistent with and environment strategy if $\sigma_{env}(Y_0 Y_1 \cdots Y_j) = X_j$ for every $j \geq 0$. For an agent strategy σ_{ag} and an environment strategy σ_{env} let $\mathsf{play}(\sigma_{ag}, \sigma_{env})$ denote the unique trace induced by both σ_{ag} and σ_{env}, and $\mathsf{play}^k(\sigma_{ag}, \sigma_{env}) = (X_0 \cup Y_0), \ldots, (X_k \cup Y_k)$ be the finite trace up to k.

Let $Goal$ be an LTL$_f$ formula over $\mathcal{X} \cup \mathcal{Y}$. An agent strategy σ_{ag} *realizes* $Goal$ if for every environment strategy σ_{env} there exists $k \geq 0$, chosen by the agent, such that the finite trace $\mathsf{play}^k(\sigma_{ag}, \sigma_{env}) \models Goal$, i.e., $Goal$ is *agent realizable.*

In standard synthesis the environment is free to choose an arbitrary move at each step, but in AI typically 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 $Goal$. Here, we specify the environment behaviour by an LTL formula Env and call it *environment specification.* In particular, Env specifies the set of environment strategies that enforce Env [16]. Moreover, we require that Env must be *environment realizable*, i.e., the set of environment strategies that enforce Env is not empty. Formally, given an LTL formula φ, we say that an environment strategy *enforces* φ, written $\sigma_{env} \rhd \varphi$, if for every agent strategy σ_{ag} we have $\mathsf{play}(\sigma_{ag}, \sigma_{env}) \models \varphi$.

The problem of LTL$_f$ *synthesis under environment specifications* is to find an agent strategy σ_{ag} such that

$$\forall \sigma_{env} \rhd Env : \exists k.\mathsf{play}^k(\sigma_{ag}, \sigma_{env}) \models Goal.$$

As shown in [16], this can be reduced to solving the synthesis problem for the implication $Env \to LTL(Goal)$ where $LTL(Goal)$ being a suitable LTL$_f$-to-LTL transformation [22], which is 2EXPTIME-complete [2].

3.1. GR(1) Environment Specifications

There have been two success stories with LTL synthesis, both having to do with the form of the specification. The first is the GR(1) approach: use safety conditions to determine the possible transitions in a game between the environment and the agent, plus one powerful notion of fairness, Generalized Reactivity(1), or GR(1). The second, inspired by AI planning, is focusing on finite-trace temporal synthesis, with LTL_f as the specification language. In [20] we take these two lines of work and bring them together, devising an approach to solve LTL_f synthesis under GR(1) environment specification. In more details, to solve the problem we observe that the agent's goal is to satisfy $\neg Env_{GR(1)} \vee Goal$, while the environment's goal is to satisfy $Env_{GR(1)} \wedge \neg Goal$. Moreover, $Goal$ can be represented by a deterministic finite automata (DFA) [6]. Then, focusing on the environment point of view, we show that the problem can be reduced into a GR(1) game in which the game arena is the complement of the DFA for $Goal$ and $Env_{GR(1)}$ 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.

This framework can be enriched by adding *safety conditions* for both the environment and the agent, obtaining a highly expressive yet still scalable form of LTL synthesis. These two kinds of safety conditions differ, since the environment needs to maintain its safety indefinitely (as usual for safety), while the agent has to maintain its safety conditions only until s/he fulfills its LTL_f goal, i.e., within a finite horizon. Again, the problem can be reduced to a GR(1) game in which the game arena is the product of the DFA for the environment safety condition and the complement of the DFA obtained by the product of deterministic automaton of the agent safety condition and the one for the agent task.

Tool. The two approaches were implemented in a so-called tool GFSynth which is based on two tools: Syft [25] for the the construction of the corresponding DFAs and Slugs [26] to solve and compute a strategy in a GR(1) game.

3.2. Fairness and Stability Environment Specifications

We now consider two different basic forms of environment specifications: a basic form of fairness $\Box\Diamond\alpha$ (always eventually α), and a basic form of stability $\Diamond\Box\alpha$ where in both cases the truth value of α is under the control of the environment, and hence the environment specifications are trivially realizable by the environment. Note that due to the existence of LTL_f goals, synthesis under both kinds of environment specifications does not fall under known easy forms of synthesis, such as GR(1) formulas [27]. For these kinds of environment specifications, [18] develops a specific algorithm based on using the DFA for the LTL_f goal as the arena and then computing 2-nested fixpoint properties over such arena.

The algorithm proceeds as follows. First, translate the LTL_f $Goal$ into a DFA $\mathcal{G}$. Then, in case of fairness environment specifications, solve the fair DFA game $\mathcal{G}$, i.e., a game over the DFA $\mathcal{G}$, 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 $\mathcal{G}$.

Likewise, for stability environment specifications, solve the stable DFA game $\mathcal{G}$, 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 on $\mathcal{G}$.
Tool. The fixpoint-based techniques for solving LTL$_f$ synthesis under fainerss and stability environment specifications are implemented in two tools called FSyft and StSyft, both based on Syft, a tool for solving symbolic LTL$_f$ synthesis.

3.3. General LTL Environment Specifications

We now consider the general case where the environment specifications are expressed in both LTL$_f$ and LTL. For this case, in [18] we develop two-stage technique to effectively handle general LTL environment specifications. This technique takes advantage of the simpler way to handle LTL$_f$ goals in the first stage and confines the difficulty of handling LTL environment specification to the bare minimum in stage 2. In particular, given an LTL environment specification and an LTL$_f$ formula $Goal$ that specifies the agent goal, the problem is to find a strategy for the agent σ_{ag} that realizes $Goal$ under LTL environment specification Env_{LTL}. The algorithm proceeds by taking the following stages.

- **Stage 1:** Build the corresponding deterministic finite automaton $\mathcal{A}_{Goal}$ of $Goal$ and solve the reachability game for the agent over $\mathcal{A}_{Goal}$. If the agent has a winning strategy in $\mathcal{A}_{Goal}$, then return it.
- **Stage 2:** If not, computes the following steps:
  1. remove from $\mathcal{A}_{Goal}$ the agent winning region, obtaining $\mathcal{A}'_{Goal}$;
  2. do the product of $\mathcal{A}'_{Goal}$ with the corresponding deterministic parity automaton (DPA) of Env_{LTL} $\mathcal{D}$, obtaining $\mathcal{B} = \mathcal{A}' \times \mathcal{D}$, and solve the parity game for the environment over it [28, 29, 30];
  3. if the agent has a winning strategy in $\mathcal{B}$ then the synthesis problem is realizable and hence return the agent winning strategy as a combination of the agent winning strategies in the two stages.

An interesting observation in [18] is that when the part of the environment specifications are expressed in LTL$_f$, i.e., the environment specifications have the form $Env = Env_\infty \wedge Env_f$, where Env_∞ can be expressed as LTL formula and Env_f as an LTL$_f$ formula. In this case the synthesis problem $Env \to Goal$ becomes $(Env_\infty \wedge Env_f) \to Goal$ which is equivalent to $Env_\infty \to (Env_f \to Goal)$ where $(Env_f \to Goal)$ is expressible in LTL$_f$. In this way Env_f does not contribute the resulting DPA and can be handled during stage 1 instead of 2 of our technique. Specifically, it builds a DFA as the union of the DFA $\overline{\mathcal{A}_{Env_f}}$, i.e., the complement of the DFA for Env_f, and the DFA $\mathcal{A}_{Goal}$ for the goal.
Tool. The two-stage technique was implemented in the tool 2SLS which is based on two tools: Syft [25] for building the corresponding DFAs and OWL [31] a tool for translating LTL into different types of automata, and thus DPAs.

Acknowledgments

We thank our co-authors on the publications mentioned in this communication: Giuseppe De Giacomo, Lucas Tabajara, Moshe Y. Vardi and Shufang Zhu. This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), by the PRIN project RIPER (No. 20203FFYLK), and by the JPMorgan AI Faculty Research Award "Resilience-based Generalized Planning and Strategic Reasoning".

References

[1] A. Church, Logic, arithmetics, and automata, in: Proc. Int. Congress of Mathematicians, 1963.
[2] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: POPL, 1989.
[3] A. Pnueli, The temporal logic of programs, in: FOCS, 1977, pp. 46–57.
[4] R. Ehlers, S. Lafortune, S. Tripakis, M. Y. Vardi, Supervisory control and reactive synthesis: a comparative introduction, Discrete Event Dynamic Systems 27 (2017) 209–260.
[5] B. Finkbeiner, Synthesis of Reactive Systems., Dependable Software Systems Eng. 45 (2016) 72–98.
[6] G. De Giacomo, M. Y. Vardi, Synthesis for LTL and LDL on finite traces, in: IJCAI, 2015.
[7] J. A. Baier, S. A. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: AAAI, 2006, pp. 788–795.
[8] G. De Giacomo, S. Rubin, Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals, in: IJCAI, 2018, pp. 4729–4735.
[9] H. Geffner, B. Bonet, A Coincise Introduction to Models and Methods for Automated Planning, Morgan&Claypool, 2013.
[10] C. Green, Theorem proving by resolution as basis for question-answering systems, in: Machine Intelligence, volume 4, American Elsevier, 1969, pp. 183–205.
[11] R. Bloem, R. Ehlers, S. Jacobs, R. Könighofer, How to handle assumptions in synthesis, in: SYNT, 2014.
[12] B. Bonet, G. De Giacomo, H. Geffner, S. Rubin, Generalized planning: Non-deterministic abstractions and trajectory constraints, in: IJCAI, 2017, pp. 873–879.
[13] K. Chatterjee, T. A. Henzinger, B. Jobstmann, Environment assumptions for synthesis, in: CONCUR, 2008, pp. 147–161.
[14] N. D'Ippolito, N. Rodríguez, S. Sardiña, Fully observable non-deterministic planning as assumption-based reactive synthesis, J. Artif. Intell. Res. 61 (2018) 593–621.
[15] B. Aminof, G. De Giacomo, A. Murano, S. Rubin, Synthesis under assumptions, in: KR, 2018, pp. 615–616.
[16] B. Aminof, G. De Giacomo, A. Murano, S. Rubin, Planning under LTL environment specifications, in: ICAPS, 2019, pp. 31–39.
[17] A. Camacho, M. Bienvenu, S. A. McIlraith, Finite LTL synthesis with environment assumptions and quality measures, in: KR, 2018, pp. 454–463.
[18] G. De Giacomo, A. Di Stasio, M. Y. Vardi, S. Zhu, Two-stage technique for LTL_f synthesis under LTL assumptions, in: KR 2020, 2020, pp. 304–314.

[19] G. De Giacomo, A. Di Stasio, G. Perelli, S. Zhu, Synthesis with mandatory stop actions, in: KR 2021, 2021, pp. 237–246.
[20] G. D. Giacomo, A. D. Stasio, L. M. Tabajara, M. Y. Vardi, S. Zhu, Finite-trace and generalized-reactivity specifications in temporal synthesis, in: IJCAI 2021, 2021, pp. 1852–1858.
[21] S. Zhu, G. De Giacomo, G. Pu, M. Vardi, LTL_f synthesis with fairness and stability assumptions, in: AAAI, 2020.
[22] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013, pp. 854–860.
[23] S. Bansal, Y. Li, L. M. Tabajara, M. Y. Vardi, Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications, in: AAAI, 2020.
[24] A. Camacho, J. A. Baier, C. J. Muise, S. A. McIlraith, Finite LTL synthesis as planning, in: ICAPS, 2018, pp. 29–38.
[25] S. Zhu, L. M. Tabajara, J. Li, G. Pu, M. Y. Vardi, Symbolic LTL_f synthesis, in: IJCAI, 2017, pp. 1362–1369.
[26] R. Ehlers, V. Raman, Slugs: Extensible GR(1) synthesis, in: CAV, 2016, pp. 333–339.
[27] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, Synthesis of reactive(1) designs, J. Comput. Syst. Sci. 78 (2012) 911–938.
[28] A. Di Stasio, A. Murano, G. Perelli, M. Vardi, Solving parity games using an automata-based algorithm, in: CIAA 2016, 2016, pp. 64–76.
[29] M. Jurdzinski, Small progress measures for solving parity games, in: STACS 2000, 2000, pp. 290–301.
[30] A. D. Stasio, A. Murano, M. Y. Vardi, Solving parity games: Explicit vs symbolic, in: CIAA 2018, 2018, pp. 159–172.
[31] J. Kretínský, T. Meggendorfer, S. Sickert, Owl: A library for ω-words, automata, and LTL, in: ATVA, 2018, pp. 543–550.