LTLf+ and PPLTL+: Extending LTLf and PPLTL to Infinite Traces

Benjamin Aminof[1] , **Giuseppe De Giacomo**[1,2] , **Sasha Rubin**[3] and **Moshe Y. Vardi**[4]
[1]University of Rome "La Sapienza", Italy
[2]University of Oxford, United Kingdom
[3]University of Sydney, Australia
[4]Rice University, USA
benjamin.aminof@tuwien.ac.at, giuseppe.degiacomo@cs.ox.ac.uk, sasha.rubin@sydney.edu.au, vardi@cs.rice.edu

Abstract

We study two logics, LTLf+ and PPLTL+, to express properties of infinite traces, that are based on the linear-time temporal logics LTLf and PPLTL on finite traces. LTLf+/PPLTL+ use levels of Manna and Pnueli's LTL safety-progress hierarchy, and thus have the same expressive power as LTL. However, they also retain a crucial characteristic of reactive synthesis for the base logics: the game arena for strategy extraction can be derived from deterministic finite automata (DFA). Consequently, these logics circumvent the notorious difficulties associated with determinizing infinite trace automata, typical of LTL synthesis. We present optimal DFA-based technique for solving reactive synthesis for LTLf+ and PPLTL+. Additionally, we adapt these algorithms to optimally solve satisfiability and model-checking for these two logics.

1 Introduction

Reactive synthesis is concerned with synthesizing programs (aka, strategies) for reactive computations (e.g., processes, protocols, controllers, robots) in active environments [Pnueli and Rosner, 1989; Finkbeiner, 2016; Ehlers *et al.*, 2017], typically from temporal logic specifications. The most common specification language is possibly Linear Temporal Logic (LTL) [Pnueli, 1977]. The basic techniques for reactive synthesis share several similarities with Model Checking, and are based on the connections between Logics, Automata, and Games [Fijalkow and others, 2023]. Indeed, Reactive Synthesis for LTL involves the following Steps: (i) having a specification φ of the desired system behavior in LTL, in which one distinguishes controllable and uncontrollable variables; (ii) extracting from the specification an equivalent automaton on infinite words, corresponding to the infinite traces satisfying φ; (iii) (differently from Model Checking) determinizing the automaton to obtain an arena for a game between the system and the environment; (iv) solving the game, by fixpoint computation, for an objective determined by the automaton's accepting condition (e.g., a parity objective for LTL), yielding a strategy for the system that fulfills the original specification.

Model Checking is mature, and its techniques may be exploited in Reactive Synthesis as well, including symbolic techniques based on Boolean encodings to compactly represent the game arena and compute fixpoints over it. However, despite this, LTL synthesis has lagged behind since the required determinization in Step (iii) remains a major performance obstacle: determinizing nondeterministic Büchi automata (NBA) is notoriously difficult [Vardi, 2007].

At first glance, the difficulty appears to be a simple consequence of the worst-case computational complexity gap: the NBA built from an LTL formula at Step (ii) is worst-case exponentially larger than the formula, and then determinizing it at Step (iii) requires a further exponential blowup, resulting in a 2EXPTIME-algorithm for reactive synthesis with LTL specifications (with a matching lower-bound). In comparison, model checking LTL specifications does not require the determinization step and is only PSPACE-complete. However there is more.

In AI, Reactive Synthesis has been studied with a focus on logics on finite traces (instead of infinite traces), e.g., LTLf [Gabbay *et al.*, 1980; Baier and McIlraith, 2006; De Giacomo and Vardi, 2013; De Giacomo and Vardi, 2015]. In fact, LTLf synthesis [De Giacomo and Vardi, 2015] is one of the two main success stories of reactive synthesis so far (the other being the GR(1) fragment of LTL [Piterman *et al.*, 2006]), and has brought about scalable results that are unprecedented [Zhu *et al.*, 2017; Bansal *et al.*, 2020; De Giacomo and Favorito, 2021; De Giacomo *et al.*, 2022].

The reason behind such good results can not be explained by worst-case complexity arguments since the complexity of LTLf synthesis is exactly the same that of LTL, i.e., 2EXPTIME-complete. Furthermore, the basic steps of the LTLf synthesis algorithm are similar to Steps (i) to (iv) earlier outlined for LTL, with each step potentially introducing the same asymptotic blowup as the corresponding step for LTL. Indeed, for LTLf, Step (ii) produces a nondeterministic finite state automaton (NFA) which in the worst case is exponentially larger than the input formula, and Step (iii) determinizes that NFA to produce a DFA which in the worst case is exponentially larger than the NFA. It turns out that the difference between the practical performance of LTL and LTLf synthesis algorithms lies in the practical performance differences of Steps (ii) and (iii) of these algorithms: for LTLf, neither of these usually manifests its worst-case potential in practice! For LTL synthesis, Step (ii) is also shared by the model-checking algorithm, which performs well in practice, thus in-

dicating that Step (iii) is the main culprit. Indeed, the exponential blow-up of the determinization of NBA is serious also in practice [Althoff *et al.*, 2006]. In comparison, for LTLf, Step (iii) not only does not usually introduce a blowup, but in fact it has been observed multiple times that determinizing an NFA results in a DFA that is *smaller* than the NFA [Tabakov and Vardi, 2005; Armoni *et al.*, 2006; Rozier and Vardi, 2012; Tabakov *et al.*, 2012].

Besides LTLf, another finite-trace logic that is gaining popularity in AI is *Pure Past* LTL (PPLTL) [Lichtenstein *et al.*, 1985; De Giacomo *et al.*, 2020; Cimatti *et al.*, 2020; Bonassi *et al.*, 2023b; Bonassi *et al.*, 2023a; Bonassi *et al.*, 2024]. This is a variant of LTLf that sees the trace backwards and has the notable property that one can obtain a symbolic (i.e., factorized) DFA directly from the formula in linear time. Moreover, while the size of the (non-symbolic) DFA corresponding to an LTLf formula can be double-exponential in the size of the formula itself, the size of the (non-symbolic) DFA corresponding to a PPLTL formula is at most a single-exponential in the size of the formula.

In this paper we show how to lift the DFA techniques at the base of the success story of LTLf and PPLTL to handle the expressive power of full LTL. To do so, we leverage the classic hierarchy of LTL properties — the *safety-progress* hierarchy [Manna and Pnueli, 1990]. [1] It consists of six classes of semantic properties, organized by inclusion. The bottom first level has the *safety* properties that intuitively express that nothing bad ever happens, and the *guarantee* (aka *co-safety*) properties that intuitively express that something good eventually happens. The second level has the *obligation* properties obtained by taking positive Boolean combinations of safety and guarantee properties. The third level has *recurrence* properties that intuitively express that something good occurs infinitely often, and *persistence* properties that intuitively say that nothing bad occurs infinitely often. The fourth level contains the *reactivity properties* which are positive Boolean combinations of recurrence and persistence properties. Each such property, which is interpreted over infinite traces, is defined in terms of sets of finite traces, which refer to finite prefixes of the infinite trace. For example, a set F of finite traces induces a basic safety (resp. progress) property that is satisfied by an infinite trace iff every prefix (resp. all but finitely many prefixes) of that trace are in F. The reactivity properties contain all properties expressible in LTL.[2]

We revisit Manna and Pnueli's hierarchy, and extract from it extensions of LTLf and PPLTL, which we call LTLf+ and PPLTL+, that can express arbitrary LTL properties on infinite traces. These logics retain a crucial characteristic for reactive synthesis of their base logics: one can exploit the techniques for translating LTLf and PPLTL formulas into DFAs [De Giacomo and Vardi, 2015; De Giacomo *et al.*, 2020]. By taking a simple product of these DFAs one can form the game arena for strategy extraction of LTLf+/PPLTL+ specifications. Naturally, the game objectives for LTLf+/PPLTL+ go beyond the simple adversarial reachability for LTLf/PPLTL. In particular, we exploit the Emerson-Lei condition [Emerson and Lei, 1987] for handling Boolean combinations, and the possibility of translating these conditions into parity conditions (typically used for LTL) or to fixpoint computations [Hausmann *et al.*, 2024]. Beside studying Reactive Synthesis in LTLf+/PPLTL+, we also study satisfiability and model checking problems of these logics, and provide both upper bounds and corresponding lower bounds.

We believe that the importance of this work is not in the presentation of new logics, new algorithmic techniques, or new hard to achieve lower-bounds. In fact, the logics LTLf+/PPLTL+ can be easily extracted from Manna and Pnueli's work, the algorithms we present, though new, employ variations of known building blocks, and the same goes for the lower bounds. The main contribution of this paper is in observing that if one combines the exact building blocks we use, in the exact way that we do, one can obtain for the first time, for natural LTLf-based and PPLTL-based logics that are as expressive as LTL, synthesis algorithms that are DFA based — and thus avoid the notoriously difficult stumbling block of determinizing automata over infinite words which has been the bane of putting LTL synthesis into practical use. In many ways, this paper has been "hiding in plain sight" and could have been written many decades ago, as all the necessary pieces were available for everyone to see for a very long time. The fact that it was never written, and that nobody noticed how to obtain the algorithms we present and tried to put them into practical use, is a strong indication to us that this information should be finally brought to the attention of the AI and reactive synthesis communities.

2 Preliminaries

Trace Properties. Let Σ be a finite alphabet. If $S \subseteq \Sigma$, then S^ω (resp. S^*) is the set of infinite (resp. finite) sequences over S. The empty sequence is denoted ε. Indexing sequences starts at 0, and we write $\tau = \tau_0\tau_1 \ldots$. For $0 \leq i < |\tau|$, write $\tau_{\leq i}$ for the prefix $\tau_0 \ldots \tau_i$, and $\tau_{\geq i}$ for the suffix $\tau_i\tau_{i+1} \ldots$ of τ. Let AP be a finite non-empty set of *atoms*. A *trace* is a non-empty finite or infinite sequence τ over $\Sigma = 2^{AP}$ (valuations of atoms); in particular, the empty sequence ε is not a trace. The length of τ is denoted $|\tau| \in \mathbb{N} \cup \{\infty\}$. A *finite-trace property* is a set of finite traces. An *infinite-trace property* is a set of infinite traces. A trace *satisfies* a property Z if it is in Z. Let $\mathfrak{L}$ be a logic for representing infinite-trace (resp. finite-trace) properties (we will later consider a few instantiations of $\mathfrak{L}$). The set of infinite (resp. finite) traces satisfying a formula φ of $\mathfrak{L}$ is denoted $[\varphi]$. An infinite-trace (resp. finite-trace) property Z is *definable* (aka, *expressible*) in $\mathfrak{L}$ iff there is some formula φ of $\mathfrak{L}$ such that $Z = [\varphi]$.

Transition Systems. A *nondeterministic transition system* $T = (\Sigma, Q, I, \delta)$ consists of a finite *input alphabet* Σ (typ-

[1] The hierarchy was introduced by Lichtenstein *et al.* in 1985, later described in detail by Manna and Pnueli in 1990; also, see the survey [Piterman and Pnueli, 2018].

[2] The hierarchy is not limited to LTL, i.e., to properties that are expressible in first-order logic (FO) over infinite sequences [Kamp, 1968], but extends to omega-regular properties, i.e., to monadic-second order logic (MSO) over infinite sequences. Indeed, all the results we present here can be extended to omega-regular properties by substituting LTLf (resp. PPLTL) by its MSO-complete variant LDLf (resp. PPLDL) [De Giacomo and Vardi, 2013].

ically, $\Sigma = 2^{AP}$), a finite set Q of *states*, a set $I \subseteq Q$ of *initial states*, and a *transition relation* $\delta \subseteq Q \times \Sigma \times Q$. The *size of* T is $|Q|$, the number of its states. We say that T is *total* if for every q, a there exists q' such that $(q, a, q') \in \delta$. Unless stated otherwise, nondeterministic transition systems are assumed to be total. In case $I = \{\iota\}$ is a singleton and δ is functional, i.e., for every q, a there is a unique q' such that $(q, a, q') \in \delta$, the transition system is called *deterministic* instead of *nondeterministic*; in this case we write ι instead of I, and $\delta(s, a) = s'$ instead of $(s, a, s') \in \delta$. A *run* (aka *path*) induced by the trace τ is a sequence $\rho = \rho_0\rho_1 \dots$ of states, where $\rho_0 \in I$ and $\rho_{i+1} \in \delta(\rho_i, \tau_i)$ for $0 \leq i < |\tau|$. We simply say that τ *has the run* ρ and that ρ *is a run of* τ. In a nondeterministic transition system T, a trace may have any number of runs (including none). If a trace τ has a run in T, we say that T *generates* τ. For $i \in \{1, 2\}$, let $T_i = (\Sigma, Q_i, \iota_i, \delta_i)$ be deterministic transition systems over the same alphabet. The *product* $T_1 \times T_2$ is the deterministic transition system $(\Sigma, Q', \iota', \delta')$ where $Q' = Q_1 \times Q_2$, $\iota' = (\iota_1, \iota_2)$, and $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. This naturally extends to a product of n-many systems. For an infinite run ρ, define *inf*$(\rho) \subseteq Q$ to be the set of states $q \in Q$ such that $q = \rho_i$ for infinitely many i.

Automata on finite traces. A *finite automaton* is a pair $\mathcal{A} = (T, F)$, where T is a transition system and $F \subseteq Q$ is the set of *final* states. The *size of* $\mathcal{A}$ is the size T. If T is nondeterministic, then $\mathcal{A}$ is called a *nondeterministic finite automaton (NFA)*; if T is deterministic, then $\mathcal{A}$ is called a *deterministic finite automaton (DFA)*. A finite trace is *accepted* by $\mathcal{A}$ if it has a run that ends in a state of F. The set of finite traces accepted by $\mathcal{A}$ is called the *language of* $\mathcal{A}$, and is denoted $L(\mathcal{A})$. Two automata $\mathcal{A}, \mathcal{B}$ are *equivalent* if $L(\mathcal{A}) = L(\mathcal{B})$.

Games and synthesis. An *arena* is a deterministic transition system $D = (\Sigma, Q, \iota, \delta)$ where $\Sigma = 2^{AP}$ and AP is partitioned into $X \cup Y$ for some sets X, Y of atoms (we use T to denote general transition systems, and D to denote arenas). Elements of 2^Y (resp. 2^X) are called *environment moves* (resp. *agent* moves). An *objective* O over D is a subset of Q^ω. Elements of O are said to *satisfy* O. A *game* is a pair $G = (D, O)$. A *strategy* is a function $\sigma : (2^Y)^* \to (2^X)$ that maps sequences of environment moves to agent moves (including the empty sequence, because the agent moves first). An *outcome* of σ is an infinite trace τ over 2^{AP} such that (i) $\tau_0 \cap X = \sigma(\varepsilon)$, and (ii) for $i > 0$ we have $\tau_i \cap X = \sigma(\tau_0 \cap Y \cdot \tau_1 \cap Y \cdots \tau_{i-1} \cap Y)$. A strategy σ is *winning* if for every outcome τ, the run $\rho \in Q^\omega$ induced by the trace τ in D satisfies O. The following computational problem is called *game solving*: given $G = (D, O)$, decide if there is a winning strategy, and if so, to return a finite representation of one. Intuitively, in a game, the agent moves first, the environment responds, and this repeats producing an infinite trace τ. The agent is trying to ensure that the run induced by τ satisfies the objective, while the environment is trying to ensure it does not.

Emerson-Lei (EL) automata and games. An *EL condition* [Emerson and Lei, 1987] over a set Q of states is a triplet (Γ, λ, B), where Γ is a finite set of *labels*, $\lambda \colon Q \to 2^\Gamma$ is a *labeling function* that assigns to a state a (possibly empty) subset of labels, and $B \colon 2^\Gamma \to \{\mathsf{true}, \mathsf{false}\}$ is a Boolean function over the set Γ (treated as variables).[3] For a state $q \in Q$ and a label $l \in \lambda(q)$, we say that q *visits* l. We will sometimes (but not always) write B as a Boolean formula over the set Γ of variables, with the usual syntax and semantics, e.g., the formula $l_1 \wedge l_2 \wedge \neg l_3$ denotes the Boolean function that assigns true to a set $Z \subseteq \Gamma$ iff Z contains l_1 and l_2 but not l_3. For a sequence $\rho \in Q^\omega$, we denote by $\mathit{inf}_\lambda(\rho) = \bigcup\{\lambda(q) : q \in \mathit{inf}(\rho)\}$, i.e., the set of labels that are visited infinitely many times by states on ρ. The EL condition (Γ, λ, B) *induces* the objective $O = \{\rho \in Q^\omega : B(\mathit{inf}_\lambda(\rho)) = \mathsf{true}\}$, i.e., $\rho \in O$ if the set of labels that are visited infinitely often satisfies B; in this case we say that ρ *satisfies the EL condition*. An *EL automaton* is a pair $\mathcal{A} = (T, (\Gamma, \lambda, B))$, where T is a transition system, and (Γ, λ, B) is an EL condition over the states of T. If T is deterministic then $\mathcal{A}$ is called a *deterministic EL automaton (DELA)*. An *EL game* is a DELA $(D, (\Gamma, \lambda, B))$ where D is an arena. DELA are closed under Boolean operations, with very simple constructions, and trivial proofs:

Proposition 1. *Given a DELA $\mathcal{A} = (T, (\Gamma, \lambda, B))$, the DELA $\mathcal{B} = (T, (\Gamma, \lambda, \neg B))$ accepts the complement of $L(\mathcal{A})$. Given DELA $\mathcal{A} = (T_1, (\Gamma_1, \lambda_1, B_1))$ and $\mathcal{B} = (T_2, (\Gamma_2, \lambda_2, B_2))$, with Γ_1 and Γ_2 being disjoint, let T be the product of T_1 and T_2. We have that the DELA $\mathcal{C} = (T, (\Gamma_1 \cup \Gamma_2, \lambda, B))$, where $\lambda(p, q) = \lambda_1(p) \cup \lambda_2(q)$, accepts the language $L(\mathcal{A}) \cup L(\mathcal{B})$ if we take $B = B_1 \vee B_2$, and the language $L(\mathcal{A}) \cap L(\mathcal{B})$ if we take $B = B_1 \wedge B_2$.*

Theorem 1. *[Hausmann* et al.*, 2024] EL games can be solved in time polynomial in the size of the arena and exponential in the number of labels.*

3 Linear Temporal Logics on Finite Traces

Let AP be a finite non-empty set of *atoms*.

LTLf. LTLf formulas are given by the syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

Here $p \in AP$, X ("next") and U ("until") are the *future operators*. Common abbreviations include $\mathsf{WX}\varphi = \neg\mathsf{X}\neg\varphi$ ("weak next"). We interpret LTLf formulas over (non-empty) finite traces. Intuitively, we evaluate starting in position 0 of the trace, $\mathsf{X}\varphi$ says that φ holds in the next step, and $\varphi_1 \,\mathsf{U}\, \varphi_2$ says that φ_2 holds eventually, and φ_1 holds at every point in time until then. In LTLf, one can predicate about both ends of the finite trace: the first instant of the trace by avoiding temporal operators (as in LTL), and the last instant by using the abbreviation *last* $:= \neg\mathsf{X}\,\mathsf{true}$. There is an algorithm that converts a given LTLf formula φ into an equivalent NFA of size exponential in the size of φ, and also into an equivalent DFA of size double-exponential in φ [De Giacomo and Vardi, 2013; De Giacomo and Vardi, 2015].

[3] This definition is implicit in [Emerson and Lei, 1987]. However, later references to the EL-condition in the literature, e.g., [Hunter and Dawar, 2005], often give a special case where the Boolean formula is over states (not labels). This special case would not allow us to achieve optimal complexities.

PPLTL. PPLTL formulas are given by the syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{Y}\,\varphi \mid \varphi\,\mathsf{S}\,\varphi$$

Here $p \in AP$, Y ("yesterday") and S ("since") are the *past operators*, analogues of "next" and "until", respectively, but in the past. Common abbreviations include $\mathsf{O}\,\varphi = \mathsf{true}\,\mathsf{S}\,\varphi$ ("(at least) once in the past"), $\mathsf{H}\,\varphi = \neg\,\mathsf{O}\,\neg\varphi$ ("historically" or "hitherto"), *first* $= \neg\,\mathsf{Y}\,\mathsf{true}$. We interpret PPLTL formulas over (non-empty) finite traces starting at the last position of the trace. One can also predicate about both ends of the finite trace: the last instant of the trace by avoiding temporal operators, and the first instant by using the abbreviation *first* $:= \neg\,\mathsf{Y}\,\mathsf{true}$. We say that an automaton $\mathcal{A}$ is *equivalent* to an LTLf/PPLTL formula φ if $L(\mathcal{A}) = [\varphi]$. There is an algorithm that converts a given PPLTL formula φ into an equivalent DFA of size exponential in φ [De Giacomo *et al.*, 2020].

Remark 1. *LTLf and PPLTL define the same set of finite trace properties, namely, the properties definable by FOL over finite traces, or equivalently by the star-free regular expressions [Gabbay* et al.*, 1980; Lichtenstein* et al.*, 1985]. However, translating LTLf into PPLTL, and vice-versa, is at least exponential in the worst case, see [Artale* et al.*, 2023] and the discussions in [De Giacomo* et al.*, 2020; Bonassi* et al.*, 2023b; Geatti* et al.*, 2024].*

LTL, and LTL with past Linear Temporal Logic (LTL) [Pnueli, 1977] is perhaps the most used logic for specifying temporal properties. LTL formulas have the same syntax as LTLf, but are interpreted over infinite traces, which allows LTL to capture liveness properties which cannot be fully done using finite trace semantics. LTL with past, i.e., enhanced with the past operators "yesterday" and "since", is expressively equivalent to LTL [Zuck, 1987].

4 LTLf+ and PPLTL+

Based on LTLf (resp. PPLTL), we define the logic LTLf+ (resp. PPLTL+) for specifying infinite-trace properties. We do so by quantifying over prefixes of infinite traces relativized to LTLf/PPLTL finite-trace properties. Formally, for a finite-trace property Z, we define four fundamental infinite-trace properties as follows: $\forall Z$ (resp. $\exists Z$) denotes the set of infinite traces τ such that every (resp. some) non-empty finite prefix of τ satisfies Z; $\forall\exists Z$ (resp. $\exists\forall Z$) denotes the set of infinite traces τ such that infinitely many (resp. all but finitely many, aka almost all) finite prefixes of τ satisfies Z. Notice that these properties cover the entire safety-progress hierarchy. Based on these we now define our logics.

Syntax. Fix a set AP of atoms. The syntax of LTLf+ (resp. PPLTL+) is given by the following grammar:

$$\Psi ::= \forall\Phi \mid \exists\Phi \mid \forall\exists\Phi \mid \exists\forall\Phi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi \mid \neg\Psi$$

where Φ are formulas in LTLf (resp. PPLTL) over AP. We use common abbreviations, e.g., $\Psi \supset \Psi'$ for $\neg\Psi \vee \Psi'$. The formulas $\forall\Phi, \exists\Phi, \forall\exists\Phi, \exists\forall\Phi$ are called *infinite-trace formulas*, and the formulas Φ are called *finite-trace formulas*.

Semantics. For an LTLf/PPLTL formula Φ, recall that we write $[\Phi] \subseteq (2^{AP})^+$ for the set of (non-empty) finite traces that satisfy Φ. For an LTLf+/PPLTL+ formula Ψ let $[\Psi] \subseteq (2^{AP})^\omega$ denote the set of infinite-trace properties, defined recursively as follows:

- $[\Psi \vee \Psi'] = [\Psi] \cup [\Psi']$, $[\Psi \wedge \Psi'] = [\Psi] \cap [\Psi']$, and $[\neg\Psi] = (2^{AP})^\omega \setminus [\Psi]$;
- $[\forall\Phi] = \forall[\Phi]$, $[\exists\Phi] = \exists[\Phi]$, $[\forall\exists\Phi] = \forall\exists[\Phi]$, and $[\exists\forall\Phi] = \exists\forall[\Phi]$.

In words, the Boolean operations are handled as usual, and $[\forall\Phi]$ (resp. $[\exists\Phi]$) denotes the set of infinite traces τ such that every (resp. some) non-empty finite prefix of τ satisfies Φ, and $[\forall\exists\Phi]$ (resp. $[\exists\forall\Phi]$) denotes the set of infinite traces τ such that infinitely many (resp. all but finitely many) finite prefixes of τ are in Φ. Note that $\forall\Phi$ and $\exists\Phi$ are dual since $\forall\Phi \equiv \neg\exists\neg\Phi$, as are $\forall\exists\Phi$ and $\exists\forall\Phi$ since $\forall\exists\Phi \equiv \neg\exists\forall\neg\Phi$. We write $\tau \models \Psi$ to mean that $\tau \in [\Psi]$.

In the terminology of the safety-progress hierarchy [Manna and Pnueli, 1990]: $[\forall\Phi]$ is a *safety* property; $[\exists\Phi]$ is a *guarantee* property; $[\exists\forall\Phi]$ is a *persistence* property; $[\forall\exists\Phi]$ is a *recurrence* property; and $[\Psi]$ is a *reactivity* property.

Examples. We illustrate LTLf+ and PPLTL+ with some examples. To specify properties of the initial instant we use propositional formulas ϕ as abbreviations for $\forall\phi$ in LTLf+, and for $\forall(\mathsf{H}(\textit{first} \supset \phi))$ in PPLTL+.

Preconditions and effect specifications in planning domains [Geffner and Bonet, 2013], non-Markovian domains [Gabaldon, 2011] are essentially safety properties, which can all be expressed with formulas of the form $\forall\Phi$, see also [Aminof *et al.*, 2023b]. For example — below ϕ_x are propositional formulas — conditional effect axioms can be expressed in LTLf+ and PPLTL+ by $\forall\,\mathsf{G}(\phi_c \supset (\mathsf{X}\,a \supset \mathsf{X}\,\phi_e))$ and $\forall\,\mathsf{H}(\mathsf{Y}\,\phi_c \supset (a \supset \phi_e))$. Frame axioms can be expressed as $\forall\,\mathsf{G}(\phi_c \wedge \mathsf{X}\,a \supset (p \equiv \mathsf{X}\,p))$ and $\forall\,\mathsf{H}(\mathsf{Y}\,\phi_c \wedge a \supset (\mathsf{Y}\,p \equiv p))$. Initial conditions ϕ_{init} can be expressed as mentioned above. Fairness of effects, e.g., assuming that action a under condition ϕ_c can effect two effects ϕ_{e_1} or ϕ_{e_2}, can be expressed as $\forall\exists\,\mathsf{F}(\phi_c \wedge \mathsf{X}(a \wedge \textit{last})) \supset \bigwedge_i \forall\exists\,\mathsf{F}(\phi_c \wedge \mathsf{X}(a \wedge \textit{last} \wedge \phi_{e_i}))$ and $\forall\exists((\mathsf{Y}\,\phi_c) \wedge a) \supset \bigwedge_i \forall\exists((\mathsf{Y}\,\phi_c) \wedge a \wedge \phi_{e_i})$, see e.g., [D'Ippolito *et al.*, 2018]. Arbitrary LTLf temporally extended goals Φ [De Giacomo and Vardi, 2013] can be expressed using formulas of the form $\exists\Phi$, e.g., eventually reaching a state where ϕ_g holds while guaranteeing to maintain a safety condition ϕ_s can be expressed as $\exists(\mathsf{F}\,\phi_g \wedge \mathsf{G}\,\phi_s)$ and $\exists(\phi_g \wedge \mathsf{H}\,\phi_s)$. Some LTL formulas, though not all [Esparza *et al.*, 2024], are also easy to express, for example $\mathsf{G}(\phi_1 \supset \mathsf{F}\,\phi_2)$ can be expressed as $\forall\exists(\mathsf{G}(\phi_1 \supset \mathsf{F}\,\phi_2))$ in LTLf+ and $\forall\exists(\neg\phi_1\,\mathsf{S}\,\phi_2 \vee \mathsf{H}\,\neg\phi_1)$ in PPLTL+. Note that although in these examples the sizes of the LTLf formulas and corresponding PPLTL are polynomially (linearly, in fact) related, this is not always the case, see Remark 1.

Expressive Power. The logics PPLTL+, LTLf+, and LTL have the same expressive power.

Theorem 2. *The logics LTLf+, PPLTL+, and LTL define the same infinite-trace properties.*

Proof. For $\mathbb{Q} \in \{\forall, \exists, \forall\exists, \exists\forall\}$, write $\mathbb{Q}$PPLTL for the fragment of PPLTL+ consisting of formulas of the form $\mathbb{Q}\Phi$

for Φ in PPLTL. Recall from [Manna and Pnueli, 1990] that: a *reactivity formula* is defined as a Boolean combination of formulas from $\forall\exists$PPLTL, and that every formula in $\forall$PPLTL, $\exists$PPLTL, $\exists\forall$PPLTL is equivalent to a reactivity formula.[4] Thus, PPLTL+ defines the same properties as the reactivity formulas. Also, recall from [Manna and Pnueli, 1990] that the reactivity formulas define the same infinite-trace properties as LTL. Finally, use the fact that PPLTL and LTLf define the same finite-trace properties [De Giacomo *et al.*, 2020]. □

PNF. The LTLf+ (resp. PPLTL+) formulas in *positive normal-form (PNF)* are given by the following grammar: $\Psi ::= \forall\Phi \mid \exists\Phi \mid \forall\exists\Phi \mid \exists\forall\Phi \mid \Psi \vee \Psi \mid \Psi \wedge \Psi$ where Φ are formulas in LTLf (resp. PPLTL) over AP. We can convert, in linear time, a given LTLf+ (resp. PPLTL+) formula into one that is in positive normal-form, in the usual way, by pushing negations into the finite-trace formulas Φ.

Remark 2. *We observe that PPLTL+ can be viewed as a syntactical variant of the normal form of LTL, enhanced with past operators, as originally introduced by Manna and Pnueli in defining the safety-progress hierarchy in [Manna and Pnueli, 1990]. Specifically, it suffices to use* G *instead of* $\forall$ *and* F *instead of* $\exists$*, without altering the meaning of the PPLTL+ formulas. Consequently, our results for PPLTL+ apply directly to Manna and Pnueli's normal form as well.*

5 Reactive Synthesis

In this section we study reactive synthesis [Pnueli and Rosner, 1989] for PPLTL+ and LTLf+ specifications.

Definition 1. *The* LTLf+ synthesis problem (resp. PPLTL+ synthesis problem) *asks, given Ψ in LTLf+ (resp. PPLTL+), to decide if there is a strategy σ such that every outcome of σ satisfies Ψ (such a strategy is said to* enforce Ψ*), and, if so, to return a finite representation of such a strategy.*

We solve the LTLf+/PPLTL+ synthesis problem with an automata-theoretic approach based on DFAs, in a 4-step **Synthesis Algorithm**, as follows. For convenience, we begin with a formula Ψ in positive normal-form. The formula Ψ can be thought of as a Boolean formula $\widehat{\Psi}$ without negations over a set of the form $[k] = \{1, 2, \ldots, k\}$ (for some k); thus Ψ can be formed from $\widehat{\Psi}$ by replacing, for every $i \in [k]$, every occurrence of i in $\widehat{\Psi}$ by a certain infinite-trace formula of the form $\mathbb{Q}_i\Phi_i$, where $\mathbb{Q}_i \in \{\exists, \forall, \exists\forall, \forall\exists\}$. Elements of $[k]$ are called *component numbers*.

Step (1). For each $i \in [k]$, convert the finite-trace formula Φ_i into an equivalent DFA $\mathcal{A}_i = (D_i, F_i)$. Say $D_i = (\Sigma, Q_i, \iota_i, \delta_i)$. We assume, without loss of generality, that the initial state has no incoming transitions. Note that one only has to do this conversion once for each finite-trace formula, even if it appears in multiple infinite-trace formulas.

Step (2). For each $i \in [k]$, build a transition system $D'_i = (\Sigma, Q_i, \iota_i, \delta'_i)$, a set $F'_i \subseteq Q_i$, and an objective O_i over Q_i:

1. Say $\mathbb{Q}_i = \forall\exists$. Let $\delta'_i = \delta_i$, $F'_i = F_i$, and $O_i = \{\rho : \mathit{inf}(\rho) \cap F'_i \neq \emptyset\}$.
2. Say $\mathbb{Q}_i = \exists\forall$. Let $\delta'_i = \delta_i$, $F'_i = F_i$, and $O_i = \{\rho : \mathit{inf}(\rho) \cap (Q \setminus F'_i) = \emptyset\}$.
3. Say $\mathbb{Q}_i = \forall$. If ι_i is not a final state, make it into a final state, i.e., define $F'_i = F_i \cup \{\iota_i\}$. Let δ'_i be like δ_i except that every non-final state is a *sink*, i.e., define $\delta'_i(q, a)$ to be q if $q \notin F'_i$, and otherwise to be $\delta_i(q, a)$. Let $O_i = \{\rho : \mathit{inf}(\rho) \cap F'_i \neq \emptyset\}$.
4. Say $\mathbb{Q}_i = \exists$. If ι_i is a final state, make it into a non-final state, i.e., define $F'_i = F_i \setminus \{\iota_i\}$. Let δ'_i be like δ_i except that every final state q is a *sink*, i.e., define $\delta'_i(q, a)$ to be q if $q \in F'_i$, and otherwise to be $\delta_i(q, a)$. Let $O_i = \{\rho : \mathit{inf}(\rho) \cap F'_i \neq \emptyset\}$. [5]

The next Theorem connects the formula $\mathbb{Q}_i\Phi_i$ to the runs in the transition system D'_i that satisfy the objective O_i, and follows by the construction in Step (1) and Step (2).

Theorem 3. *An infinite trace satisfies $\mathbb{Q}_i\Phi_i$ iff the run it induces in D'_i satisfies the objective O_i.*

Proof. Let τ be an infinite trace and let ρ (resp. ρ') be the run in D_i (resp. D'_i) induced by τ. Recall that the empty string is not a trace, and so a non-empty prefix of τ corresponds to a prefix of its run of length > 1.

We look at each case separately.

1. τ satisfies $\forall\exists\Phi_i$ iff (by Definition of $\forall\exists$) infinitely many prefixes of τ satisfy Φ_i iff infinitely many prefixes of ρ end in F_i iff $\mathit{inf}(\rho) \cap F_i \neq \emptyset$ iff (since $\rho' = \rho$ and $F'_i = F_i$) ρ' satisfies O_i.
2. τ satisfies $\exists\forall\Phi_i$ iff (by Definition of $\exists\forall$) almost all prefixes of τ satisfy Φ_i iff almost all prefixes of ρ end in F_i iff it is not the case that infinitely many prefixes of ρ end in $Q_i \setminus F_i$ iff $\mathit{inf}(\rho) \cap (Q \setminus F_i) = \emptyset$ iff (since $\rho' = \rho$ and $F'_i = F_i$) ρ' satisfies O_i.
3. τ satisfies $\forall\Phi_i$ iff (by Definition of $\forall$) all non-empty prefixes of τ satisfy Φ_i iff $\rho_n \in F_i$ for all $n > 0$ iff (since the transitions from the initial state and final states in D_i are unchanged in D'_i) $\rho'_n \in F'_i$ for all $n > 0$ iff $\rho'_n \in F'_i$ for infinitely many n (since all non-final states are sinks) iff $\mathit{inf}(\rho') \cap F'_i \neq \emptyset$ iff ρ' satisfies O_i.
4. τ satisfies $\exists\Phi_i$ iff (by Definition of $\exists$) some non-empty prefix of τ satisfies Φ_i iff $\rho_n \in F_i$ for some $n > 0$ iff (since the transitions from the initial state and non-final states in D_i are unchanged in D'_i, and taking the least such n) $\rho'_n \in F'_i$ for some $n > 0$ iff $\rho'_n \in F'_i$ for infinitely many n (since all final states are sinks) iff $\mathit{inf}(\rho') \cap F'_i \neq \emptyset$ iff ρ' satisfies O_i.

This completes the proof. □

Call a state $q_i \in Q_i$ *marked* if either (a) $\mathbb{Q}_i \neq \exists\forall$ and $q_i \in F'_i$, or (b) $\mathbb{Q}_i = \exists\forall$ and $q_i \in Q_i \setminus F'_i$. Thus, by the

[4] This equivalence may involve a blowup, but this is not relevant to understand expressive power.

[5] We make two remarks about the cases $\mathbb{Q} \in \{\forall, \exists\}$: by our assumption that the initial state has no incoming transitions, redefining whether the initial state is a final state or not does not change the set of non-empty strings accepted by the DFA $\mathcal{A}_i$; we could have defined either or both these cases to have an objective like the $\exists\forall$ case (instead of the $\forall\exists$ case) since seeing a sink is the same as seeing it infinitely often which is the same as seeing it from some point on.

construction in Step (2), in case $\mathbb{Q}_i \neq \exists\forall$, the objective O_i says that some marked state is seen infinitely often, and in case $\mathbb{Q}_i = \exists\forall$ the objective O_i says that no marked state is seen infinitely often.

Step (3). Build the product transition system $D = \prod_{i\in[k]} D'_i$. Let Q be the state set of D. Define an EL condition (Γ, λ, B) over Q as follows: $\Gamma = [k]$ (i.e., the labels are the component numbers), $\lambda(q) = \{i \in [k] : q_i \text{ is marked}\}$ (i.e., a state is labeled by the numbers of those components that are in marked states), and the Boolean formula B is formed from the Boolean formula $\widehat{\Psi}$ by simultaneously replacing, for every i with $\mathbb{Q}_i = \exists\forall$, every occurrence of i by $\neg i$. Define the deterministic EL-automaton $\mathcal{A}_\Psi = (D, (\Gamma, \lambda, B))$.

Example 1. *Say $\Psi = \forall\exists\Phi_1 \vee \exists\forall\Phi_2$. Then the label set is $\Gamma = \{1, 2\}$. Let τ be an infinite trace, let ρ (resp. ρ^i) be the induced run in D (resp. D'_i). By Theorem 3, $\tau \models \Psi$ iff ρ^1 satisfies O_1 or ρ^2 satisfies O_2. By definition of O_i, this is iff ρ^1 sees a marked state of D'_1 infinitely often or ρ^2 does not see a marked state of D'_2 infinitely often. And this is iff the set of labels $i \in \{1, 2\}$ for which ρ^i sees a marked state infinitely often, satisfies the Boolean formula $B := 1 \vee \neg 2$.*

Theorem 4. *Ψ is equivalent to the EL-automaton $\mathcal{A}_\Psi$.*

Proof. In Step (2), for $i \in [k]$, the objective O_i is induced by the following EL-condition $(\Gamma_i, \lambda_i, B_i)$ over Q_i. Let $\Gamma = \{i\}$, i.e., the only label is i. For $q_i \in Q_i$, let $\lambda_i(q_i) = \{i\}$ if q_i is marked, and $\lambda_i(q_i) = \{\}$ otherwise. Then, for a run ρ in D'_i, we have that $\mathit{inf}_{\lambda_i}(\rho) = \{i\}$ iff some state seen infinitely often in ρ is marked (and otherwise $\mathit{inf}_{\lambda_i}(\rho) = \{\}$). We consider the two cases:

1. Say $\mathbb{Q}_i \neq \exists\forall$. Recall that q_i is marked iff $q_i \in F'_i$. Thus, $\rho \in O_i$ iff $\mathit{inf}(\rho) \cap F'_i \neq \emptyset$ iff $\mathit{inf}_{\lambda_i}(\rho) = \{i\}$ iff ρ satisfies the EL-condition $(\Gamma_i, \lambda_i, B_i)$ where B_i is the Boolean formula i.
2. Say $\mathbb{Q}_i = \exists\forall$. Recall that q_i is marked iff $q_i \in Q_i \setminus F'_i$. Thus, $\rho \in O_i$ iff $\mathit{inf}(\rho) \cap (Q_i \setminus F'_i) = \emptyset$ iff $\mathit{inf}_{\lambda_i}(\rho) = \{\}$ iff ρ satisfies the EL-condition $(\Gamma_i, \lambda_i, B_i)$ where B_i is the Boolean formula $\neg i$.

By Theorem 3, Ψ is equivalent to the Boolean combination formed from $\widehat{\Psi}$ by replacing each occurrence of index i by the EL-automaton $\mathcal{A}_i := (D'_i, (\Gamma_i, \lambda_i, B_i))$. Applying Proposition 1 to this Boolean combination results in an EL-automaton whose transition system is the product D, and whose EL-condition is (Γ, λ, B) where $\Gamma = [k]$, $\lambda((q_1, \cdots, q_n)) = \bigcup_i \lambda_i(q_i)$, and B is formed from $\widehat{\Psi}$ by simultaneously replacing, for every i with $\mathbb{Q}_i = \exists\forall$, every occurrence of i by $\neg i$. Now notice that this is exactly the definition of $\mathcal{A}_\Psi$ in Step (3) of the Synthesis Algorithm. □

Step (4). Solve the EL game $\mathcal{A}_\Psi$. By Theorem 1, this can be done in time polynomial in the size of the arena and exponential in the number of labels. This completes the description of the algorithm.

Since, for an LTLf+ (resp. PPLTL+) formula Ψ, the size of the arena D is double-exponential (resp. single-exponential) in the size of Ψ, and the number of labels k is linear in the size of Ψ, we get:

Theorem 5. *The LTLf+ (resp. PPLTL+) synthesis problem is in 2EXPTIME (resp. EXPTIME).*

We observe that our algorithms are (worst-case) optimal:

Theorem 6. *The LTLf+ (resp.PPLTL+) synthesis problem is 2EXPTIME-complete (resp. EXPTIME-complete).*

Proof. For the lower bounds, synthesis is 2EXPTIME-hard already for the fragment of LTLf+ consisting of formulas of the form $\exists\Phi$ [De Giacomo and Vardi, 2015], and synthesis is EXPTIME-hard already for the fragment of PPLTL+ for formulas of the form $\exists\Phi$ [De Giacomo *et al.*, 2020]. □

Remark 3. *PPLTL+ synthesis is in EXPTIME (Theorem 5), which is exponentially cheaper than LTL synthesis, which is 2EXPTIME-hard [Pnueli and Rosner, 1990]. This means that LTL is, in the worst case, exponentially more succinct than PPLTL+. One may be tempted to think that the fact that PPLTL+ synthesis is cheaper than LTL synthesis is because LTL is always more succinct than PPLTL+. However, it turns out this is not true. Indeed, PPLTL+ is, in the worst case, exponentially more succinct than LTL [Markey, 2003, Section 3]. This leads to the interesting phenomenon of two logics with the same expressive power, incomparable succinctness, yet solving a logic problem for one of them is exponentially cheaper. This phenomenon holds for the finite-trace logics LTLf/PPLTL [Artale* et al.*, 2023], and so the fact that PPLTL+ synthesis is in EXPTIME shows that this phenomenon can be lifted to all LTL-definable infinite-trace properties.*

Finally, we remark that Step (4) requires solving an EL game built from the product of the DFAs with a simple Boolean formula (that mimics the given formula Ψ). This can be implemented based on fixpoint computations [Hausmann *et al.*, 2024], which has the potential for efficient symbolic implementations. Alternatively, one may convert the EL game to a parity game, and solve the parity game using existing algorithms and solvers [van Dijk, 2018]. The literature provides hints on how to perform this transformation [Hunter and Dawar, 2005; Casares *et al.*, 2022]. However, one has to be careful not to blow up the state space in a way that would degrade the overall complexity stated in Theorem 5.

6 Satisfiability and Model Checking

We now study other standard reasoning tasks for our logics, i.e., satisfiability and model checking. Let $\mathfrak{L}$ be one of LTLf/PPLTL. The $\mathfrak{L}+$ *satisfiability problem* asks, given Ψ in $\mathfrak{L}+$, to decide whether $[\Psi] \neq \emptyset$. For a nondeterministic transition system T, that is not assumed to be total, with input alphabet $\Sigma = 2^{AP}$, we say that T *satisfies* Ψ if every infinite trace generated by T satisfies Ψ.[6] The $\mathfrak{L}+$ *model checking*

[6] There are many ways to define transitions systems for verification. Commonly, these are state-labeled directed graphs. For technical convenience, instead of labeling the state with l, we will instead label all outgoing edges with l, and thus we can view these as (not necessarily total) nondeterministic transition systems.

problem asks, given Ψ in $\mathfrak{L}+$, and a nondeterministic labeled transition system T, whether T satisfies Ψ.

Theorem 7. *The LTLf+ (resp. PPLTL+) satisfiability and model checking problems are EXPSPACE-complete (resp. PSPACE-complete).*

Satisfiability Upper Bound. Given Ψ, build the equivalent EL-automaton $\mathcal{A}_\Psi = (D, (\Gamma, \lambda, B))$ of Step (3) above. Recall that $|\Gamma| = k$, which is linear in $|\Psi|$. Satisfiability can be solved by checking if there is an infinite run in D that satisfies the EL-condition. This can be done in time polynomial in the size of D and k, as follows. Check if there is a state s of D such that (i) there is a path from the initial state to s, and (ii) there is a path (that uses at least one transition) from s to s such that the set of labels visited on this path is a satisfying assignment of B. However, since the size of D is doubly-exponential (resp. exponential) in the size $|\Psi|$ of the given formula, this algorithm would run in 2EXPTIME for LTLf+ (resp. EXPTIME for PPLTL+). These complexities can be lowered to EXPSPACE (resp. PSPACE) by exploiting the fact that reachability in directed-graphs is in NLOGSPACE [Papadimitriou, 2007], and, as usual, doing things "on-the-fly".

Model Checking Upper Bound. Given a nondeterministic transition system T and an LTLf+/PPLTL+ formula Ψ, we have that T satisfies Ψ iff it is not the case that there is an infinite trace generated by T that satisfies $\neg\Psi$. Thus, since deterministic complexity classes are closed under complement, it is enough to provide an optimal algorithm for the following problem: given a nondeterministic transition system T and a LTLf+/PPLTL+ formula Ψ, decide if there is an infinite trace generated by T that satisfies Ψ. To do this we proceed as in the "on-the-fly" algorithm for satisfiability. However, we add to the current state of D a component consisting of the current state of T.

Lower Bounds. The lower bounds already hold for formulas of the form $\forall\Phi$ and $\exists\Phi$. This can proven by relying on results and techniques from [De Giacomo and Vardi, 2013; Vardi and Stockmeyer, 1985; Kupferman and Vardi, 2000; Bansal *et al.*, 2023].

Remark 4. *The EXPSPACE-completeness result for satisfiability and model checking in LTLf+ may appear surprising since satisfiability and model checking for LTL are both PSPACE-complete [Clarke* et al.*, 2018], but in fact confirms and extends a recent EXPSPACE-hardness result for model checking the fragment of LTLf+ limited to the guarantee class [Bansal* et al.*, 2023]. We remark that despite this theoretical worst-case complexity, we expect model checking and satisfiability of LTLf+ to be much more tractable in practice, the same way and for the same reason that the 2EXPTIME-hardness for LTLf synthesis does not manifest in practice (as discussed in the Introduction). Finally, observe that our results show that the complexity of satisfiability and model checking of PPLTL+ is the same as for LTL.*

7 Related Work

The guarantee and safety fragment of LTLf+/PPLTL+, i.e., formulas of the form $\exists\Phi$ and $\forall\Phi$ for Φ in LTLf/PPLTL, has been studied in the literature (under other names), i.e., as LTLf synthesis [De Giacomo and Vardi, 2015], LTLf synthesis under environment specifications [Aminof *et al.*, 2025], and LTLf model-checking (non-terminating semantics) [Bansal *et al.*, 2023]. Synthesis for the Safety fragment has been studied in [Cimatti *et al.*, 2024]

Reactive synthesis is deeply related to Planning [De Giacomo and Rubin, 2018; Camacho *et al.*, 2019; De Giacomo *et al.*, 2023], and in particular to (strong) planning for temporally extended goals in fully observable nondeterministic domains [Cimatti *et al.*, 2003; Bacchus and Kabanza, 1998; Bacchus and Kabanza, 2000; Calvanese *et al.*, 2002; Baier and McIlraith, 2006; Baier *et al.*, 2007; Gerevini *et al.*, 2009]. Following, e.g., [De Giacomo and Rubin, 2018] it is easy to design optimal algorithms for handling LTLf+/PPLTL+ temporally extended goals in nondeterministic planning domains, and would be interesting to extend this to best-effort planning [De Giacomo *et al.*, 2023; Aminof *et al.*, 2021; Aminof *et al.*, 2023a].

The safety-progress hierarchy has mainly been used to understand the expressivity of fragments of LTL. However, recent results that convert arbitrary LTL formulas into a normal form consisting of a combination of classes in the hierarchy in single exponential time [Esparza *et al.*, 2020; Esparza *et al.*, 2024] are providing new interest in the hierarchy. Our work shares this interest, though sidesteps the LTL normalization, and focuses instead on specifying directly using finite traces, and then lifting to infinite traces through quantification on prefixes.

8 Conclusion

We studied two logics LTLf+/PPLTL+ that have the same expressive power as LTL, and showed how to apply DFA techniques to build arenas for synthesis, typical for handling the base finite-trace logics, thus side-stepping the difficulty of determinizing automata on infinite traces, typical of LTL synthesis. Our approach only relies on the base logic being closed under negation and having conversions into DFAs. Thus, it would also work for any other base logic with these properties, e.g., linear dynamic logic on finite traces [De Giacomo and Vardi, 2015] or pure past linear dynamic logic [De Giacomo *et al.*, 2020]. We thus advocate revisiting Manna and Pnueli's safety-progress hierarchy, focusing on (i) using it directly for representation, by putting emphasis on the finite-trace components of infinite properties, and (ii) solving computational problems by exploiting DFA-based arena constructions.

A key characteristic of Planning is that the system continuously receives a goal, "thinks" about how to achieve it, synthesizes a plan, executes the plan, and repeats [Geffner and Bonet, 2013]. However, if we consider goals on infinite traces, e.g., expressed in LTLf+/PPLTL+, this loop needs to be revised, by considering that the agent will receive its new goal while executing for the previous ones, as in live synthesis [Finkbeiner *et al.*, 2022; Zhu and De Giacomo, 2022]. This observation puts *goal reasoning* [Aha, 2018] at the center of the stage when dealing with infinite trace goals, and is an avenue of future research.

Acknowledgments

This work is supported in part by the ERC Advanced Grant WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project FAIR (No.PE0000013), the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI, NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CNS-2016656, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

[Aha, 2018] D. W. Aha. Goal reasoning: Foundations, emerging applications, and prospects. *AI Mag.*, 39(2):3–24, 2018.

[Althoff *et al.*, 2006] C. S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006.

[Aminof *et al.*, 2021] B. Aminof, G. D. Giacomo, and S. Rubin. Best-effort synthesis: Doing your best is not harder than giving up. In *IJCAI*, 2021.

[Aminof *et al.*, 2023a] B. Aminof, G. D. Giacomo, and S. Rubin. Reactive synthesis of dominant strategies. In *AAAI*, 2023.

[Aminof *et al.*, 2023b] B. Aminof, Giuseppe De Giacomo, Antonio Di Stasio and Hugo Francon, S. Rubin, and S. Zhu. LTLf synthesis under environment specifications for reachability and safety properties. In *EUMAS*, 2023.

[Aminof *et al.*, 2025] B. Aminof, G. De Giacomo, A. Di Stasio, H. Francon, S. Rubin, and S. Zhu. ltl synthesis under environment specifications for reachability and safety properties. *Inf. Comput.*, 303:105255, 2025.

[Armoni *et al.*, 2006] R. Armoni, D. Korchemny, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. Deterministic dynamic monitors for linear-time assertions. In *FATES/RV*, 2006.

[Artale *et al.*, 2023] A. Artale, L. Geatti, N. Gigante, A. Mazzullo, and A. Montanari. LTL over finite words can be exponentially more succinct than pure-past LTL, and vice versa. In *TIME*, 2023.

[Bacchus and Kabanza, 1998] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. Math. Artif. Intell.*, 22(1-2):5–27, 1998.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000.

[Baier and McIlraith, 2006] J. A. Baier and S. A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 2006.

[Baier *et al.*, 2007] J. A. Baier, C. Fritz, and S. A. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS*, 2007.

[Bansal *et al.*, 2020] S. Bansal, Y. Li, L. M. Tabajara, and M. Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, 2020.

[Bansal *et al.*, 2023] S. Bansal, Y. Li, L. M. Tabajara, M. Y. Vardi, and A. M. Wells. Model checking strategies from synthesis over finite traces. In *ATVA*, 2023.

[Bonassi *et al.*, 2023a] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A. Gerevini, and E. Scala. FOND planning for pure-past linear temporal logic goals. In *ECAI*, 2023.

[Bonassi *et al.*, 2023b] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A. Gerevini, and E. Scala. Planning for temporally extended goals in pure-past linear temporal logic. In *ICAPS*, 2023.

[Bonassi *et al.*, 2024] L. Bonassi, G. De Giacomo, A. Gerevini, and E. Scala. Shielded FOND: Planning with safety constraints in pure-past linear temporal logic. In *ECAI*, 2024.

[Calvanese *et al.*, 2002] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Reasoning about actions and planning in LTL action theories. In *KR*, 2002.

[Camacho *et al.*, 2019] A. Camacho, M. Bienvenu, and S. A. McIlraith. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, 2019.

[Casares *et al.*, 2022] A. Casares, T. Colcombet, and K. Lehtinen. On the size of good-for-games rabin automata and its link with the memory in muller games. In *ICALP*, 2022.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 1–2(147), 2003.

[Cimatti *et al.*, 2020] A. Cimatti, L. Geatti, N. Gigante, A. Montanari, and S. Tonetta. Reactive synthesis from extended bounded response LTL specifications. In *FMCAD*, 2020.

[Cimatti *et al.*, 2024] A. Cimatti, L. Geatti, N. Gigante, A. Montanari, and S. Tonetta. Extended bounded response LTL: a new safety fragment for efficient reactive synthesis. *Formal Methods Syst. Des.*, 64(1):1–49, 2024.

[Clarke *et al.*, 2018] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[De Giacomo and Favorito, 2021] G. De Giacomo and M. Favorito. Compositional approach to translate LTL_f/LDL_f into deterministic finite automata. In *ICAPS*, 2021.

[De Giacomo and Rubin, 2018] G. De Giacomo and S. Rubin. Automata-theoretic foundations of fond planning for LTL_f/LDL_f goals. In *IJCAI*, 2018.

[De Giacomo and Vardi, 2013] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 2013.

[De Giacomo and Vardi, 2015] G. De Giacomo and M. Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 2015.

[De Giacomo *et al.*, 2020] G. De Giacomo, A. Di Stasio, F. Fuggitti, and S. Rubin. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI*, 2020.

[De Giacomo *et al.*, 2022] G. De Giacomo, M. Favorito, J. Li, M. Y. Vardi, S. Xiao, and S. Zhu. Ltl_f synthesis as AND-OR graph search: Knowledge compilation at work. In *IJCAI*, 2022.

[De Giacomo *et al.*, 2023] G. De Giacomo, G. Parretti, and S. Zhu. LTL_f best-effort synthesis in nondeterministic planning domains. In *ECAI*, 2023.

[D'Ippolito *et al.*, 2018] N. D'Ippolito, N. Rodríguez, and S. Sardiña. Fully observable non-deterministic planning as assumption-based reactive synthesis. *J. Artif. Intell. Res.*, 61:593–621, 2018.

[Ehlers *et al.*, 2017] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi. Supervisory control and reactive synthesis: a comparative introduction. *Discret. Event Dyn. Syst.*, 27(2), 2017.

[Emerson and Lei, 1987] E. A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987.

[Esparza *et al.*, 2020] J. Esparza, J. Kretínský, and S. Sickert. A unified translation of linear temporal logic to ω-automata. *J. ACM*, 67(6):33:1–33:61, 2020.

[Esparza *et al.*, 2024] J. Esparza, R. Rubio, and S. Sickert. Efficient normalization of linear temporal logic. *J. ACM*, 71(2):16:1–16:42, 2024.

[Fijalkow and others, 2023] N. Fijalkow et al. Games on graphs. *arXiv*, 2305.10546, 2023.

[Finkbeiner *et al.*, 2022] B. Finkbeiner, F. Klein, and N. Metzger. Live synthesis. *Innov. Syst. Softw. Eng.*, 18(3):443–454, 2022.

[Finkbeiner, 2016] B. Finkbeiner. Synthesis of reactive systems. *Dependable Softw. Syst. Eng.*, 45:72–98, 2016.

[Gabaldon, 2011] A. Gabaldon. Non-Markovian control in the situation calculus. *Artif. Intell.*, 175(1), 2011.

[Gabbay *et al.*, 1980] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *POPL*, 1980.

[Geatti *et al.*, 2024] L. Geatti, M. Montali, and A. Rivkin. Foundations of reactive synthesis for declarative process specifications. In *AAAI*, 2024.

[Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Springer, 2013.

[Gerevini *et al.*, 2009] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6):619–668, 2009.

[Hausmann *et al.*, 2024] D. Hausmann, M. Lehaut, and N. Piterman. Symbolic solution of Emerson-Lei games for reactive synthesis. In *FoSSaCS*, 2024.

[Hunter and Dawar, 2005] P. Hunter and A. Dawar. Complexity bounds for regular games. In *MFCS*, 2005.

[Kamp, 1968] H. Kamp. *Tense logic and the theory of linear order*. PhD thesis, UCLA, 1968.

[Kupferman and Vardi, 2000] O. Kupferman and M. Y. Vardi. An automata-theoretic approach to modular model checking. *TOPLAS*, 22(1), 2000.

[Lichtenstein *et al.*, 1985] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Logic of Programs*, pages 196–218, 1985.

[Manna and Pnueli, 1990] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *PODC*, 1990.

[Markey, 2003] N. Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bull. EATCS*, 79:122–128, 2003.

[Papadimitriou, 2007] C. H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

[Piterman and Pnueli, 2018] N. Piterman and A. Pnueli. Temporal logic and fair discrete systems. In *Handbook of Model Checking*, pages 27–73. Springer, 2018.

[Piterman *et al.*, 2006] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, 2006.

[Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

[Pnueli and Rosner, 1990] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, 1990.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[Rozier and Vardi, 2012] K. Y. Rozier and M. Y. Vardi. Deterministic compilation of temporal safety properties in explicit state model checking. In *HVC*, 2012.

[Tabakov and Vardi, 2005] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, 2005.

[Tabakov *et al.*, 2012] D. Tabakov, K. Y. Rozier, and M. Y. Vardi. Optimized temporal monitors for systemc. *Formal Methods Syst. Des.*, 41(3):236–268, 2012.

[van Dijk, 2018] T. van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *TACAS*, 2018.

[Vardi and Stockmeyer, 1985] M. Y. Vardi and L. J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In *STOC*, 1985.

[Vardi, 2007] M. Y. Vardi. The Büchi complementation saga. In *STACS*, 2007.

[Zhu and De Giacomo, 2022] S. Zhu and G. De Giacomo. Act for your duties but maintain your rights. In *KR*, 2022.

[Zhu *et al.*, 2017] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi. Symbolic LTL_f Synthesis. In *IJCAI*, 2017.

[Zuck, 1987] L. Zuck. *Past Temporal Logic*. PhD thesis, Weizmann Institute of Science, 1987.