Emerson-Lei and Manna-Pnueli Games for LTL$_f$+ and PPLTL+ Synthesis

Daniel Hausmann[1] , **Shufang Zhu**[1] , **Gianmarco Parretti**[2] , **Christoph Weinhuber**[3]
Giuseppe De Giacomo[2,3] , **Nir Piterman**[4]
[1]University of Liverpool, UK [2]Sapienza University of Rome, Italy [3]University of Oxford, UK
[4]University of Gothenburg and Chalmers University of Technology, Sweden
{shufang.zhu, hausmann}@liverpool.ac.uk, parretti@diag.uniroma1.it,
{christoph.weinhuber, giuseppe.degiacomo}@cs.ox.ac.uk, piterman@chalmers.se

Abstract

Recently, the Manna-Pnueli Hierarchy has been used to define the temporal logics LTL$_f$+ and PPLTL+, which allow to use finite-trace LTL$_f$/PPLTL techniques in infinite-trace settings while achieving the expressiveness of full LTL. In this paper, we present the first actual solvers for reactive synthesis in these logics. These are based on games on graphs that leverage DFA-based techniques from LTL$_f$/PPLTL to construct the game arena. We start with a symbolic solver based on Emerson-Lei games, which reduces lower-class properties (guarantee, safety) to higher ones (recurrence, persistence) before solving the game. We then introduce Manna-Pnueli games, which natively embed Manna-Pnueli objectives into the arena. These games are solved by composing solutions to a DAG of simpler Emerson-Lei games, resulting in a provably more efficient approach. We implemented the solvers and practically evaluated their performance on a range of representative formulas. The results show that Manna-Pnueli games often offer significant advantages, though not universally, indicating that combining both approaches could further enhance practical performance.

1 Introduction

This paper is about devising actual solvers for reactive synthesis in LTL$_f$+ and PPLTL+, which can be seen as a Manna-Pnueli normal form for Linear Temporal Logic (LTL) based respectively on its finite trace variant LTL$_f$ and on Pure Past LTL (PPLTL).

Reactive synthesis deals with synthesizing programs (aka strategies) from temporal specifications, for systems (e.g. agents, processes, protocols, controllers, robots) that interact with their environments during their execution (Pnueli and Rosner 1989; Finkbeiner 2016; Ehlers et al. 2017). The most common specification language is possibly Linear Temporal Logic (LTL) (Pnueli 1977). Reactive synthesis shares foundational techniques with model checking, grounded in the interplay between logic, automata, and games (Fijalkow et al. 2023). For LTL, synthesis normally proceeds by: (1) specifying the desired behavior with controllable and uncontrollable variables; (2) translating the specification into an equivalent automaton over infinite words; (3) determinizing the automaton—unlike in model checking—to define a game between system and environment; and (4) solving the game, typically with a parity objective, to derive a strategy satisfying the original specification.

The symbolic techniques for model checking based on Boolean encodings to compactly represent game arenas and compute fixpoints can also be leveraged in reactive synthesis. Nevertheless, differently from model checking, LTL synthesis has struggled to achieve comparable efficiency, primarily due to the inherent complexity of Step (3): the determinization of nondeterministic Büchi automata (NBA), a process known to be computationally challenging (Vardi 2007; Althoff, Thomas, and Wallmeier 2006).

In AI, reactive synthesis is closely related 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, De Giacomo, and Vardi 2002; Baier, Fritz, and McIlraith 2007; Gerevini et al. 2009; De Giacomo and Rubin 2018; Camacho, Bienvenu, and McIlraith 2019). Plans are typically assumed to terminate, and this has led to a focus on logics over finite traces, rather than infinite ones, with LTL$_f$, the finite-trace variant of LTL, being a common choice (Gabbay et al. 1980; Baier and McIlraith 2006; De Giacomo and Vardi 2013; De Giacomo and Vardi 2015). In fact, LTL$_f$ synthesis (De Giacomo and Vardi 2015) is, along with the GR(1) fragment of LTL (Piterman, Pnueli, and Sa'ar 2006), one of the two major success stories in reactive synthesis to date.

The steps of the LTL$_f$ synthesis algorithm closely mirror Steps (1)–(4) outlined for LTL, potentially incurring similar asymptotic blowups. Specifically, Step (2) yields a nondeterministic finite automaton (NFA) that can be exponentially larger than the input formula, and Step (3) determinizes it into a DFA, which can be exponentially larger than the NFA. However, in practice, these worst-case blowups rarely occur for LTL$_f$: Step (2) is shared with model checking, where it performs well; Step (3) benefits from the fact that NFA determinization is rarely problematic in practice – in fact, it is often observed that the resulting DFA is actually smaller than the original NFA (Tabakov and Vardi 2005; Armoni et al. 2006; Rozier and Vardi 2012; Tabakov, Rozier, and Vardi 2012; Zhu et al. 2021).

Moreover, tools like MONA (Klarlund, Møller, and Schwartzbach 2002), used by LTL$_f$ synthesizers to extract automata, return DFAs in semi-symbolic form, i.e., transitions are represented symbolically. This is a very important aspect, because one other significant barrier that we may underestimate in LTL and LTL$_f$ synthesis is the alphabet explosion problem. That is to say, once there are just 20 propositions, which is not a very large number, the alphabet already has size 1 million, making explicit treatment of the alphabet infeasible. Symbolic methods circumvent this problem (to some extent).

These nice characteristics of LTL$_f$ have enabled the community to build a first fully symbolic solver for LTL$_f$ synthesis (Zhu et al. 2017b). Since then, a series of papers have improved symbolic technology for LTL$_f$ synthesis significantly (Bansal et al. 2020; De Giacomo and Favorito 2021; Kankariya and Bansal 2024; Zhu and Favorito 2025). Furthermore, the LTL$_f$ technology has been extended to several other cases, from safety and reachability (Zhu et al. 2017a; Bansal et al. 2022; Aminof et al. 2025a) all the way to GR(1) (De Giacomo et al. 2022).

These desirable properties are also exhibited by Pure-Past LTL (PPLTL) (Lichtenstein, Pnueli, and Zuck 1985; De Giacomo et al. 2020), which has recently emerged as an even simpler alternative to LTL$_f$, while maintaining the same expressive power (Bonassi et al. 2023b; Bonassi et al. 2023a; Bonassi et al. 2024).

The question is, can we use this technology to do LTL synthesis? Recently (Aminof et al. 2025b) gave affirmative answer to this question by exploiting Manna and Pnueli's normal form (Manna and Pnueli 1990). This normal form is based on specifying finite trace properties, e.g. expressed in LTL$_f$ (or PPLTL) and then requiring that the property holds for *some* prefixes of infinite traces (*guarantee* properties), for *all* prefixes (*safety* properties), for *infinitely many* prefixes (*recurrence* properties), or for *all but finitely many* prefixes (*persistence* properties). Any LTL formula can be expressed as a Boolean combination of these four classes.

How can we take advantage of Manna and Pnueli's normal form? An answer is, first of all, in building the arena. For each finite trace property, we can build the DFA, using LTL$_f$ technology (Aminof et al. 2025b). Then, depending on Manna and Pnueli's class, we choose the accepting condition to (guarantee) visit a final state once, (safety) never leave the set of final states, (recurrence) visit some final state infinitely often, (persistence) stay out of the set of final states finitely many times. All these component automata run in parallel, so we can take their product. In fact, we can represent the product symbolically in a straightforward way. To solve these temporal conditions, we have several options. With a simple (polynomial) manipulation, we can transform guarantee and safety properties, and corresponding automata, into recurrence properties, see (Aminof et al. 2025b). What we get is an Emerson-Lei game where the only temporal conditions are boolean combinations of recurrence and persistence (Emerson and Lei 1987). Such a game can be solved symbolically through a fixpoint algorithm based on compact semantic representations of Emerson-Lei objectives, called Zielonka Trees (Hausmann, Lehaut, and Piterman 2024). Note that Emerson-Lei games can be reduced (with an exponential multiplicative factor) to parity games, for which there are well-developed solvers. However, while in (Aminof et al. 2025b) it was shown that no further determinization would be needed to generate the parity automaton, we do need to handle possible permutations (latest appearance records) which destroy the symbolic representation. Instead, the Emerson-Lei approach preserves the symbolic representation, which is used for the fixpoint computation directly. We stress that both these solutions are worst-case optimal and solve LTL$_f$+ synthesis in 2EXPTIME and PPLTL+ synthesis in EXPTIME, respectively (these problems are 2EXPTIME-complete and EXPTIME-complete) (Aminof et al. 2025b). However, there is a sharp difference in practical terms among them, since only the Emerson-Lei based solution preserves the symbolic structure coming from the finite-trace properties.

Can we improve on the Emerson-Lei approach? In particular, can we avoid reducing guarantee and safety, which do not require nested fixpoints, to recurrence, which indeed requires nesting and complicates the Zielonka Tree? In this paper we answer affirmatively. We introduce a new kind of games called *Manna-Pnueli games* that handle the combination of conditions guarantee, safety, recurrence and persistence directly. In particular, for such games, we give a symbolic fixpoint solution analogous to that of Emerson-Lei games in (Hausmann, Lehaut, and Piterman 2024), but exploiting the simplicity of guarantee and safety conditions to simplify the fixpoint computation.

The contributions of this paper are as follows:

- First, we present a symbolic synthesizer based on Emerson-Lei games, which reduces lower-class properties (guarantee, safety) to higher-class ones (recurrence, persistence) before solving the game.
- We then introduce Manna-Pnueli games, which natively deal with Manna-Pnueli objectives on the game arena.
- Next, we present a symbolic synthesizer based on these new Manna-Pnueli games.
- We show that Manna-Pnueli games can be solved by composing solutions to a DAG of simpler Emerson-Lei games.
- We prove that this compositional approach to solving Manna-Pnueli games is asymptotically more efficient than the naïve reduction to Emerson-Lei games.
- Finally, we implement both solvers using state-of-the-art symbolic technology and evaluate their performance on a range of representative formulas.

The results show that Manna-Pnueli games often offer significant advantages, though not universally, suggesting that combining both approaches could further enhance practical performance.

2 Preliminaries

LTL$_f$+ and PPLTL+. We briefly recall the logics LTL$_f$+ and PPLTL+ following (Aminof et al. 2025b). These logics allow to express Boolean combinations of guarantee, safety, recurrence, and persistence properties over *finite traces*; the underlying finite trace properties are specified either in LTL$_f$ (De Giacomo and Vardi 2013), that is, in LTL evaluated over finite traces, or in PPLTL, that is, in pure past LTL (De Giacomo et al. 2020). Note that in this paper we use X for weak next and X[!] for strong next.

Formulas of LTL$_f$+ (resp. PPLTL+) over a countable set

AP of propositions are constructed by the grammar

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

where the Φ are finite trace LTL$_f$ (resp. PPLTL) formulas. We use $[\Phi] \subseteq (2^{AP})^*$ to denote the set of finite traces over 2^{AP} that satisfy a finite trace formula Φ. Given a set $T \subseteq (2^{AP})^*$ of finite traces, we let $\exists T$ ($\forall T$) denote the set of infinite traces $\tau \in (2^{AP})^\omega$ such that at least one finite prefix of τ is (all finite prefixes of τ are) contained in T; similarly, we let $\forall\exists T$ ($\exists\forall T$) denote the set of infinite traces for which infinitely many (all but finitely many) prefixes are contained in T. Then we evaluate LTL$_f$+/PPLTL+ formulas Ψ over infinite traces using the extension $[\Psi] \subseteq (2^{AP})^\omega$ defined inductively by $[\Psi \vee \Psi'] = [\Psi] \cup [\Psi']$, $[\Psi \wedge \Psi'] = [\Psi] \cap [\Psi']$, $[\neg\Psi] = (2^{AP})^\omega \setminus [\Psi]$, $[\mathbb{Q}\Phi] = \mathbb{Q}[\Phi]$ where $\mathbb{Q} \in \{\exists, \forall, \forall\exists, \exists\forall\}$. The logics LTL$_f$, PPLTL+, and LTL define the same infinite-trace properties.

Automata on finite traces. A transition system $T = (\Sigma, Q, I, \delta)$ consists of a finite alphabet Σ, a finite set Q of states, a set $I \subseteq Q$ of initial states, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. For $q \in Q$ and $a \in \Sigma$, we define $\delta(q, a) = \{q' \in Q \mid (q, a, q') \in \delta\}$. A transition system is *deterministic* if $|I| = 1$ and $|\delta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$, and *nondeterministic* otherwise; for deterministic transition systems, we write $\delta(q, a) = q'$ where $q' \in Q$ is *the* state such that $q' \in \delta(q, a)$ and denote the initial state by ι. A *finite automaton* $\mathcal{A} = (T, F)$ is a transition system together with a set $F \subseteq Q$ of accepting states. If T is deterministic, then $\mathcal{A}$ is a deterministic finite automaton (DFA), otherwise it is a nondeterministic finite automaton (NFA). A *run* of an automaton on a word $w \in \Sigma^*$ is a path through T starting at an initial state such that the sequence of transition labels of the path is w; a finite run is accepting if it ends in an accepting state. An automaton *accepts* the language $L(\mathcal{A})$ consisting of all finite words for which there is an accepting run of $\mathcal{A}$.

Finite trace LTL$_f$ and PPLTL formulas over AP can be turned into equivalent finite automata (with alphabet 2^{AP}), accepting exactly the traces that satisfy the formulas. For each LTL$_f$ formula φ, there is an equivalent NFA of size $2^{\mathcal{O}(|\varphi|)}$ and an equivalent DFA of size $2^{2^{\mathcal{O}(|\varphi|)}}$ (De Giacomo and Vardi 2015). For each PPLTL formula φ, there is an equivalent DFA of size $2^{\mathcal{O}(|\varphi|)}$ (De Giacomo et al. 2020).

Infinite-duration games on finite graphs. A *game arena* is a finite directed graph $A = (V, E \subseteq V \times V)$ such that V is partitioned into the sets V_s and V_e of game nodes controlled by the *system player* and the *environment player*, respectively. Define $E(v) = \{v' \in V \mid (v, v') \in E\}$ for $v \in V$ and assume that $E(v) \neq \emptyset$ for all $v \in V$. A *play* is a path in A; let $\mathsf{plays}(A)$ denote the set of infinite plays over A. A *strategy* for the system player is a function $\sigma : V^* \cdot V_s \to V$ that assigns a single game node $\sigma(v_0 v_1 \ldots v_n) \in E(v_n)$ to any finite play $v_0 v_1 \ldots v_n \in V^* \cdot V_s$ that ends in a game node owned by the system player ($v_n \in V_s$). A play $v_0 v_1 \ldots$ is *compatible* with a strategy if for every i such that $v_i \in V_s$, $v_{i+1} = \sigma(v_0 v_1 \ldots v_i)$. An *objective* on an arena A is a set $O \subseteq \mathsf{plays}(A)$ of plays; then a play is *winning* for the system player if it is contained in O. Notions of strategies and winning plays for the environment player are defined dually. A strategy σ wins a node $v \in V$ for a player if all plays that are compatible with σ and start at v are winning for that player. *Solving* a game amounts to computing the set of game nodes won by the two players, together with their witnessing strategies.

Reactive synthesis and games. In the context of reactive synthesis, we assume that the set AP of atomic propositions is partitioned into sets X and Y, denoting *system* and *environment actions*, respectively. A *(synthesis) strategy* is a function $\sigma : (2^Y)^* \to 2^X$, and an *outcome* of such a strategy is an infinite word $(x_0 \cup y_0)(x_1 \cup y_1) \ldots \in (2^{AP})^\omega$ such that $x_{i+1} = \sigma(y_0 y_1 \ldots y_i)$ for all $i \geq 0$. Thus, synthesis strategies encode the behavior of *transducers*.

Definition 1. *(Aminof et al. 2025b) The* synthesis problem *for LTL$_f$+ (resp. PPLTL+) asks for a given formula Ψ whether there is a strategy σ such that every outcome of σ satisfies Ψ, and if so, to return such a strategy.*

Then a deterministic transition system $T = (2^{\mathsf{AP}}, Q, \iota, \delta)$ induces a game arena $A_T = (Q \cup Q \times 2^X \cup Q \times 2^X \times 2^Y, E)$ with the system player owning nodes $q \in Q$ and the environment player owning all other nodes; the moves are defined by putting $E(q) = \{q\} \times 2^X$, $E(q, x) = \{(q, x)\} \times 2^Y$, and $E(q, x, y) = \{\delta(q, x \cup y)\}$. Plays $q_0(q_0, x_0)(q_0, x_0, y_0)q_1(q_1, x_1)(q_1, x_1, y_1) \ldots$ over A_T induce runs $q_0 q_1 \ldots$ of T on words $(x_0 \cup y_0)(x_1 \cup y_1) \ldots$.

Hence the synthesis problem can be solved by transforming the input formula into a deterministic transition system and then solving the induced game with a suitable objective.

3 Synthesis via EL Games

It has been shown (Aminof et al. 2025b) that the synthesis problem of LTL$_f$+ (resp. PPLTL+) reduces to the solution of games with so-called Emerson-Lei objectives. These are Boolean combinations of recurrence and persistence objectives, formally defined as follows.

Given a finite set Γ of events, an *Emerson-Lei (EL) formula* (over Γ) is a positive Boolean formula over atoms of the shape $\mathsf{GF}\, a$ or $\mathsf{FG}\, a$, where $a \in \Gamma$. We evaluate EL formulas over infinite sequences of sets of events, that is, over elements of $(2^\Gamma)^\omega$. Given $L_1 L_2 \ldots \in (2^\Gamma)^\omega$, we put

$$L_1 L_2 \ldots \models \mathsf{GF}\, a \quad \Leftrightarrow \quad \forall i. \exists j > i. a \in L_j$$
$$L_1 L_2 \ldots \models \mathsf{FG}\, a \quad \Leftrightarrow \quad \exists i. \forall j > i. a \in L_j$$

The evaluation of Boolean combinations ($\wedge, \vee$) of EL formulas and of Boolean constants ($\top$,$\bot$) is as expected. Given a finite set Q, an infinite sequence $\pi = q_0 q_1 \ldots \in Q^\omega$, a labeling function $\gamma : Q \to 2^\Gamma$, and an EL formula φ, we denote $\gamma(q_0)\gamma(q_1) \ldots \models \varphi$ by $\pi \models \varphi$.

An *Emerson-Lei objective* $O = (\Gamma, \gamma, \varphi)$ on a finite set Q is given in the form of a set Γ of events, a labeling function $\gamma : Q \to 2^\Gamma$ and an EL formula φ over Γ; we generally assume that $|\Gamma| \leq |Q|$. A sequence $\pi \in Q^\omega$ is contained in O (by slight abuse of notation) if and only if $\pi \models \varphi$.

An *Emerson-Lei automaton* $\mathcal{A} = (T, O)$ is a transition system T together with an EL objective on T. Then $\mathcal{A}$ recognizes the (ω-regular) language $L(\mathcal{A})$ of all infinite words for which there is run π of T such that $\pi \in O$.

An *Emerson-Lei game* $G = (A, O)$ consists of a game arena $A = (V, E)$ together with an EL objective O on A, specifying the winning plays. EL games are *determined*, that is, every game node is won by exactly one of the players.

Theorem 1. *(McNaughton 1993; Zielonka 1998) Emerson-Lei games with n nodes and k Emerson-Lei events can be solved in time $\mathcal{O}(k! \cdot n^{k+2}) \in 2^{\mathcal{O}(k \log n)}$; winning strategies require at most $k!$ memory values.*

Recently, a symbolic algorithm for the solution of EL games has been proposed (Hausmann, Lehaut, and Piterman 2024). This algorithm leverages *Zielonka trees*, that is, succinct semantic representations of EL objectives, to transform EL objectives into equivalent fixpoint equation systems. EL game solution then can be performed by symbolic solution of the corresponding equation system. The resulting algorithm realizes the time bound stated in Theorem 1.

Remark 1. *Strategy extraction for EL games works as described in (Hausmann, Lehaut, and Piterman 2024). Leaves in Zielonka trees are labelled with sets of events, thought of as a memory of recently visited events. Winning strategies then use leaves in Zielonka trees as memory values and combine the events of current game nodes with the events in the current memory values to update the set of recently visited events and obtain new memory values. Strategies are based on* existential subtrees *of Zielonka trees, reducing strategy sizes (Dziembowski, Jurdzinski, and Walukiewicz 1997).*

Next, we show how this symbolic solution algorithm for EL games can be used for LTL$_f$+ (resp. PPLTL+) synthesis.

Consider an input LTL$_f$+ or PPLTL+ formula Ψ given in positive normal form, that is, given as a positive Boolean formula over k atoms $\mathbb{Q}_i\Phi_i$ where $\mathbb{Q}_i \in \{\exists, \forall, \exists\forall, \forall\exists\}$, and where all Φ_i are LTL$_f$ (resp. PPLTL) formulas. The synthesis algorithm transforms Ψ into an equivalent EL automaton and then solves the EL game induced by this automaton.

Step 1. For each $i \in [k]$, convert the finite trace formula Φ_i into an equivalent DFA (D_i, F_i) where $D_i = (2^{AP}, Q_i, \iota_i, \delta_i)$. Assume without loss of generality that ι_i does not have incoming transitions. If $\mathbb{Q}_i = \forall$, then add ι_i to F_i and turn every non-accepting state into a non-accepting sink state (for $q \notin F_i$ and all $a \in \Sigma$, put $\delta_i(q, a) = q$); if $\mathbb{Q}_i = \exists$, then remove ι_i from F_i and turn every accepting state into an accepting sink state (for $q \in F_i$ and all $a \in \Sigma$, put $\delta_i(q, a) = q$). Construct the product transition system $D_\Psi = \prod_{i\in[k]} D_i$ and let Q_Ψ denote the state space of D_Ψ.

We point out that turning (non)accepting states into sinks in Step 1. does not increase the size of automata. For i such that $\mathbb{Q}_i = \forall$ or $\mathbb{Q}_i = \exists$, the resulting automata intuitively incorporate memory on whether a (non)accepting state has been visited so far, or not. The following is immediate.

Lemma 1. *Let $\mathbb{Q}_i = \exists$ (resp. $\mathbb{Q}_i = \forall$). Then an infinite path in D_i visits F_i (resp. $Q_i \setminus F_i$) at least once if and only if the run eventually visits only states from F_i (resp. $Q_i \setminus F_i$).*

For $q = (q_1, \ldots, q_k) \in Q$, let $L_q = \{i \in [k] \mid q_i \in F_i, \mathbb{Q}_i \in \{\forall, \exists\}\}$ denote the local events for which the corresponding automaton is in an accepting state in q.

Step 2. Define the EL objective $O = (\Gamma, \gamma, \varphi)$ by putting $\Gamma = [k]$ and $\gamma(q_1, \ldots, q_k) = \{i \in [k] \mid q_i \in F_i\}$ for $(q_1, \ldots, q_k) \in Q_\Psi$. Depending on the shape of $\mathbb{Q}_i$, define φ_i to be $\mathsf{GF}\, i$ (if $\mathbb{Q}_i \in \{\exists, \forall, \forall\exists\}$), or $\mathsf{FG}\, i$ (if $\mathbb{Q}_i = \exists\forall$). The EL formula φ is obtained from the input formula Ψ by replacing each atom $\mathbb{Q}_i\Phi_i$ with φ_i. Define the deterministic EL automaton $\mathcal{A}_\Psi = (D_\Psi, O)$.

Proposition 1. *(Aminof et al. 2025b) The formula Ψ is equivalent to the EL automaton $\mathcal{A}_\Psi$.*

Step 3. Solve the EL game induced by $\mathcal{A}_\Psi$. If the system player wins the initial state $(\iota_1, \ldots, \iota_k)$ of D_Ψ, then extract a witnessing strategy.

Theorem 2. *The LTL$_f$+ (resp. PPLTL+) synthesis problem can be decided symbolically via EL games in* 2EXPTIME *(resp.* EXPTIME*).*

In more detail, consider an input formula Ψ of size n and consisting of k recurrence and persistence formulas and d guarantee and safety formulas. The constructed EL game over $\mathcal{A}_\Psi$ is of size at most $n' = 2^{2^n}$ (resp. $n' = 2^n$) and the objective O has $k + d$ events. By Theorem 1, it can be solved in time $2^{\mathcal{O}((k+d) \log n')}$.

Remark 2. *All steps in the described synthesis algorithm are open to symbolic implementation.*

The construction of individual DFAs for finite trace subformulas in Steps 1. and 2. is based on the powerset construction which is amenable to symbolic implementation, and taking the product of the individual automata in Step 3. is an inherently symbolic operation. Finally, the game solution in Step 4. can be implemented using the symbolic algorithm for EL game solution from (Hausmann, Lehaut, and Piterman 2024).

Remark 3. *Transducers for realizable specifications Ψ are obtained by extracting a winning strategy for the system player in the EL game induced by $\mathcal{A}_\Psi$ (see Remark 1).*

4 MP Automata and Games

Next, we introduce automata and games with objectives that support, in addition to the Boolean combinations of recurrence and persistence allowed in EL objectives, also combinations with guarantee and safety objectives. We call such objectives Manna-Pnueli (MP) objectives. Adding guarantee and safety properties to EL objectives enables a direct translation from LTL$_f$+ (resp. PPLTL+) formulas to automata, as the structure of the resulting objectives directly corresponds to the high-level structure of formulas. In Section 5 below, MP automata and the solution of MP games will be instrumental to an alternative solution of the synthesis problem for LTL$_f$+ (resp. PPLTL+).

Given a finite set Γ of events, a *Manna-Pnueli formula* is a positive Boolean formula over atoms of the shape $\mathsf{GF}\, a$, $\mathsf{FG}\, a$, $\mathsf{F}\, a$, or $\mathsf{G}\, a$, where $a \in \Gamma$. We refer to events that occur in a concrete formula only in atoms of the shape $\mathsf{F}\, a$ or $\mathsf{G}\, a$ as *local events*, and as *Emerson-Lei events* to all other events. Given $L_1L_2 \ldots \in (2^\Gamma)^\omega$, we put

$$L_1L_2 \ldots \models \mathsf{F}\, a \quad \Leftrightarrow \quad \exists i.\, a \in L_i$$
$$L_1L_2 \ldots \models \mathsf{G}\, a \quad \Leftrightarrow \quad \forall i.\, a \in L_i$$

All other operators are evaluated in the same way as for EL formulas; denote $\gamma(q_0)\gamma(q_1)\ldots \models \varphi$ by $\pi \models \varphi$, where $\gamma : Q \to 2^\Gamma$ is a labelling function, and $\pi = q_0 q_1 \ldots \in Q^\omega$.

A *Manna-Pnueli objective* $O = (\Gamma, \gamma, \varphi)$ on a finite set Q consists of a set Γ of events, a labeling function $\gamma : Q \to 2^\Gamma$ and an MP formula φ over Γ; assume that $|\Gamma| \leq |Q|$. A sequence $\pi \in Q^\omega$ is contained in O if and only if $\pi \models \varphi$.

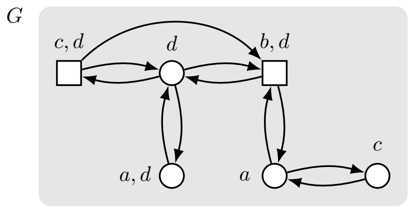
We point out that like EL objectives, MP objectives are ω-regular (that is, they can be transformed to parity objectives); however, unlike EL objectives, MP objectives are not prefix independent (that is, there may be $u_1, u_2 \in Q^*$, $v \in Q^\omega$ such that $u_1 v \in O$ but $u_2 v \notin O$).

A *Manna-Pnueli automaton* $\mathcal{A} = (T, O)$ is a transition system T with set Q of states together with a Manna-Pnueli objective on Q. Then $\mathcal{A}$ recognizes the (ω-regular) language $L(\mathcal{A})$ consisting of all infinite words for which there is a run π of T such that $\pi \in O$.

A *Manna-Pnueli game* $G = (A, O)$ consists of a game arena $A = (V, E)$ together with a Manna-Pnueli objective O on A. By definition, MP games are determined.

Example 1. *Consider the game G with events a, b, c, d, with Manna-Pnueli objective $\varphi = (\mathsf{GF}\, a \wedge \mathsf{GF}\, b \wedge \mathsf{G}\, d) \vee (\mathsf{FG}\, \neg b \wedge \mathsf{F}\, c)$ and with node ownership indicated by circles (system player) and boxes (environment player).*

In this example the system player wins every node by a strategy that uses memory to alternatingly move from the node labelled with d to the nodes labelled with a, d and b, d, respectively; the strategy always moves from the node labelled with a to the node labelled with c.



Any play following this strategy either avoids the node labelled with just a forever and infinitely often visits the nodes labeled with a, d and b, d (and then satisfies $\mathsf{GF}\, a \wedge \mathsf{GF}\, b \wedge \mathsf{G}\, d$), or it eventually only visits the two bottom right nodes (and then satisfies $\mathsf{FG}\, \neg b \wedge \mathsf{F}\, c$).

5 Synthesis via MP Games

We now show how LTL$_f$+ and PPLTL+ synthesis both reduce to the solution of MP games. To this end, we consider an input LTL$_f$+ or PPLTL+ formula Ψ given in positive normal form over k finite trace formulas, as in Section 3. The synthesis algorithm transforms Ψ to an equivalent deterministic MP automaton and solves the MP game induced by the automaton. We point out that the transformation from Ψ to an MP automaton is immediate (and does, unlike the construction in Section 3, not transform any states into sink states), as MP automata natively embed the high-level structure of LTL$_f$+ and PPLTL+ formulas.

Step 1a. Convert the finite trace formulas Φ_i into equivalent DFAs (D_i, F_i) with $D_i = (2^{AP}, Q_i, \iota_i, \delta_i)$. If $\mathbb{Q}_i = \forall$, then add ι_i to F_i; if $\mathbb{Q}_i = \exists$, then remove ι_i from F_i. Construct the product transition system $D'_\Psi = \prod_{i \in [k]} D_i$.

Step 2a. Define the Manna-Pnueli objective $O' = (\Gamma, \gamma, \varphi)$ by putting $\Gamma = [k]$ and $\gamma(q_1, \ldots, q_k) = \{i \in [k] \mid q_i \in F_i\}$ for $(q_1, \ldots, q_k) \in Q$. Depending on the shape of $\mathbb{Q}_i$, define φ_i to be $\mathsf{F}\, i$ (if $\mathbb{Q}_i = \exists$), $\mathsf{G}\, i$ (if $\mathbb{Q}_i = \forall$), $\mathsf{GF}\, i$ (if $\mathbb{Q}_i = \forall\exists$), or $\mathsf{FG}\, i$ (if $\mathbb{Q}_i = \exists\forall$). The Manna-Pnueli formula φ is obtained from the input formula Ψ by replacing each atom $\mathbb{Q}_i \Phi_i$ with φ_i. Define the deterministic Manna-Pnueli automaton $\mathcal{A}'_\Psi = (D'_\Psi, O')$.

Step 3a. Solve the Manna-Pnueli game induced by $\mathcal{A}'_\Psi$. If the system player wins the initial state $(\iota_1, \ldots, \iota_k)$ of D'_Ψ, then extract a witnessing strategy.

We show correctness of the simpler automata construction in Step 1a. To this end, define the objective $O_i = (\Gamma_i, \gamma_i, \varphi_i)$ on D_i, where $\Gamma_i = [1]$ and $\gamma_i : D_i \to [1]$ maps nodes from F_i to $\{1\}$, and all other nodes to $\emptyset$. Then showing the following lemma is immediate.

Lemma 2. *Let π be an infinite trace. For all $i \in [k]$, we have $\pi \models \mathbb{Q}_i \Phi_i$ if and only if $\pi \in L(D_i, O_i)$.*

Proposition 2. *The formula Ψ is equivalent to the MP automaton $\mathcal{A}'_\Psi$.*

The proof is by induction over Ψ, using Lemma 2 for the base cases.

Theorem 3. *The LTL$_f$+ (resp. PPLTL+) synthesis problem can be decided symbolically via MP games in* 2EXPTIME *(resp.* EXPTIME*).*

In more detail, consider an input formula Ψ of size n and with k recurrence and persistence formulas, and d guarantuee and safety formulas. The constructed game over $\mathcal{A}'_\Psi$ is of size at most $n' = 2^{2^n}$ (resp. $n' = 2^n$) and has k EL events and d local events.

While the MP automata construction described in Step 1a. is natural and straightforward, it has the drawback that the resulting automata do not incorporate memory for local events. A more efficient procedure is obtained by using the automata construction from Section 3 (Step 1.) that adds sink states and incorporates memory for local events at no extra cost (Lemma 1). In what follows, we assume that the latter automata construction is used in the construction of $\mathcal{A}'_\Psi$. Then the synthesis game can be solved in time $2^{\mathcal{O}(k \log n')}$ by reduction to a composition of EL games (as described in Section 6 below, see Theorem 4). This indicates that the proposed synthesis method can handle guarantee and safety properties "for free". It improves significantly over the reduction to EL games from Section 3 which solves the synthesis problem in time $2^{\mathcal{O}((d+k) \log n')}$ (Corollary 1).

Remark 4. *For realizable specifications Ψ, transducers are obtained by extracting a winning strategy for the system player from the MP game induced by $\mathcal{A}'_\Psi$ (see Remark 5).*

6 Solution of MP Games

We show how MP games can be reduced (symbolically) to EL games. Such reductions enable the use of the solution algorithm for EL games from (Hausmann, Lehaut, and Piterman 2024) to symbolically solve MP games.

Fix a Manna-Pnueli game $G = (A, O)$ with arena $A = (V, E)$ and objective $O = (\Gamma, \gamma, \varphi)$, and assume w.l.o.g. that each atom in φ uses a unique event. Let G have $n = |V|$ nodes, m edges, k EL events, and d local events.

In the first step, equip the arena A with additional memory to store the local events that have been visited (or violated) in a play so far. If A already has memory for the local events, then this step can be skipped. This is the case in our synthesis algorithm when using the automata construction given in Section 3, Step 1.: by Lemma 1, each state q in D_Ψ comes with memory L_q for the local events.

An immediate, but costly, reduction from MP games to EL games then is to simply treat all local events as Emerson-Lei events. The synthesis algorithm described in Section 3 above is based on this reduction.

This first reduction can be improved by instead reducing MP games to directed acyclic graphs (DAGs) of EL games with simplified objectives. The simplified EL objectives are obtained by partially evaluating the original MP objective according to the local events from the auxiliary memory. Then the reduced DAG of EL games can be solved by solving the individual EL games in a bottom-up fashion. The alternative synthesis algorithm given in Section 5 leverages this improved reduction.

Arena transformation. In the transformed arena A', the nodes from the MP game G are annotated with sets of local events, acting as memory values that keep track of the F-events that have occurred so far and the G-events that have occurred in every game step so far. The moves in A' are the same as in A, but update the memory whenever an F-event occurs for the first time, or a G-event does not occur for the first time.

Let Γ_{F} and Γ_{G} denote the sets of F- and G-events in φ, respectively and put $\Gamma_{\mathsf{F,G}} = \Gamma_{\mathsf{F}} \cup \Gamma_{\mathsf{G}}$. Let Γ_{EL} denote the set of Emerson-Lei events in φ.

We define a memory update function upd that takes as input a game node $v \in V$ and a current memory value $L \subseteq \Gamma_{\mathsf{F,G}}$, and computes the set $\mathsf{upd}(v, L)$ obtained from L by removing all G-events that do not occur at v, and adding all F-events that occur at v. Formally, we put

$$\mathsf{upd}(v, L) = (L \cap \gamma(v)) \cap \Gamma_{\mathsf{G}} \cup (L \cup \gamma(v)) \cap \Gamma_{\mathsf{F}}.$$

Then the reduced game arena $A' = (V', E')$ incorporating the auxiliary memory is defined by putting $V' = V \times 2^{\Gamma_{\mathsf{F,G}}}$ and $E'(v, L) = E(v) \times \{\mathsf{upd}(v, L)\}$; nodes $(v, L) \in V'$ are owned by the owner of v in G.

The memory update is defined in such a way that (not) visiting a local event once in a play on A corresponds to eventually forever (not) having the event in the auxiliary memory in a play on A'.

We also define a labelling function $\gamma'(v, L) = (\gamma(v) \cap \Gamma_{\mathsf{EL}}) \cup \{L\}$, marking nodes $(v, L) \in V'$ with the union of the Emerson-Lei events of v and the local events from the auxiliary memory L.

Reduction to EL Games. Define the EL objective $O_1 = (\Gamma, \gamma', \varphi_1)$, where the EL formula φ_1 is obtained from φ by replacing all atoms of shape $\mathsf{F}\, a$ or $\mathsf{G}\, a$ in φ with $\mathsf{GF}\, a$. The objective O_1 consists of plays for which the EL events visited by the corresponding play in the original game together with the events from the auxiliary memory satisfy the adapted formula. We obtain an EL game $G_1 = (A', O_1)$ of size $\mathcal{O}(n \cdot 2^d)$ that has $k + d$ EL events.

Lemma 3. *The games G and G_1 are equivalent.*

Sketch. The claim follows by showing that (not) visiting a local event *once* in G corresponds to eventually (not) having this event in the auxiliary memory *forever* in G_1 (similar to Lemma 1). Thus atoms $\mathsf{F}\, a$ and $\mathsf{G}\, a$ in G correspond to atoms $\mathsf{GF}\, a$ in G_1. In fact, we could also choose $\mathsf{FG}\, a$ to represent local atoms in G_1. ☐

Using Theorem 1, we thus obtain

Corollary 1. *Manna-Pnueli games with n nodes, k Emerson-Lei events and d local events can be solved in time $\mathcal{O}((k+d)! \cdot (n \cdot 2^d)^{k+d+2}) \in 2^{\mathcal{O}(d(k+d) \log n)}$; winning strategies require at most $(k + d)!$ memory values.*

The above bound on solution time improves to $\mathcal{O}((k+d)! \cdot n^{k+d+2}) \in 2^{\mathcal{O}((k+d) \log n)}$ for games over arenas that have memory for local events (such as the arenas constructed in Sections 3 and 5). In such cases, the arena transformation is not required so that G_1 has just n nodes and $k + d$ EL events.

Reduction to Compositions of EL Games. Whenever the memory value L changes by a move in the arena A', at least one G-event is permanently removed from L, or at least one F-event is permanently added to L. As there are finitely many events, the memory changes finitely often. Consequently, the memory values partition the arena A' into $2^{|\Gamma_{\mathsf{F,G}}|}$ subarenas; we denote the subarena with the memory values fixed to L by A'_L. Together with the memory changing moves, the partition of A' into the arenas A'_L forms a DAG with top and bottom subarenas $A'_{\Gamma_{\mathsf{G}}}$ and $A'_{\Gamma_{\mathsf{F}}}$, respectively.

Given an MP formula ψ and a set $L \subseteq \Gamma_{\mathsf{F,G}}$ of local events, we let ψ_L denote the formula that is obtained from ψ by evaluating F- and G-atoms according to L, that is by replacing atoms $\mathsf{F}\, a$ and $\mathsf{G}\, a$ with $\top$ if $a \in L$, and with $\bot$ otherwise. We point out that ψ_L does not contain any F- and G-atoms and hence is an EL formula.

Given a play $\pi \in \mathsf{plays}(A)$, we define the set of F- and G-events that are satisfied by π by

$$\Gamma_{\mathsf{F,G}}(\pi) = \{c \in \Gamma_{\mathsf{F}} \mid \pi \models \mathsf{F}\, c\} \cup \{c \in \Gamma_{\mathsf{G}} \mid \pi \models \mathsf{G}\, c\}.$$

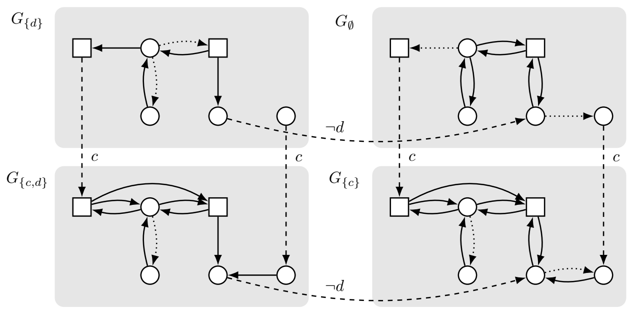
The following lemma relates MP objectives to (simplified) EL objectives; the proof is immediate.

Lemma 4. *For all plays $\pi \in \mathsf{plays}(A)$ we have $\pi \models \varphi$ if and only if $\pi \models \varphi_{\Gamma_{\mathsf{F,G}}(\pi)}$.*

Based on Lemma 4, we simplify the objective formula φ_1 from G_1 by using, for each subarena A'_L, the simpler objective φ_L. Formally, we define $\varphi_2 = \bigvee_{L \in 2^{\Gamma_{\mathsf{F,G}}}} (\bigwedge_{a \in L} \mathsf{Inf}\, a \wedge \varphi_L)$. This formula expresses the existence of some memory value L such that the auxiliary memory eventually stabilizes to L and such that φ_L is satisfied. Plays satisfying this formula eventually stay within some subarena A'_L forever and satisfy φ_L. We put $O_2 = (\Gamma, \gamma', \varphi_2)$ and let G_2 denote the EL game (A', O_2).

Example 2. *We demonstrate the reduction of MP games to DAGs of EL games for the game from Example 1 (not depicting labelings with events for readability). The objectives of the individual subgames are as follows.*

$$\begin{aligned}\varphi_\emptyset &= \varphi[\mathsf{G}\, d \mapsto \bot, \mathsf{F}\, c \mapsto \bot] = \bot\\ \varphi_{\{c\}} &= \varphi[\mathsf{G}\, d \mapsto \bot, \mathsf{F}\, c \mapsto \top] = \mathsf{FG}\,\neg b\\ \varphi_{\{d\}} &= \varphi[\mathsf{G}\, d \mapsto \top, \mathsf{F}\, c \mapsto \bot] = \mathsf{GF}\, a \wedge \mathsf{GF}\, b\\ \varphi_{\{c,d\}} &= \varphi[\mathsf{G}\, d \mapsto \top, \mathsf{F}\, c \mapsto \top] = (\mathsf{GF}\, a \wedge \mathsf{GF}\, b) \vee \mathsf{FG}\,\neg b\end{aligned}$$



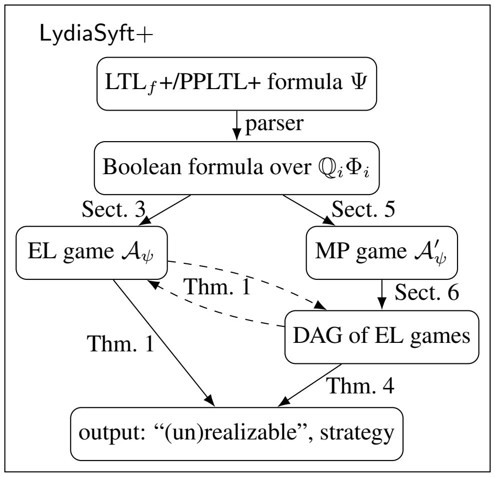
Dashed edges depict changes in the memory for local events (corresponding to first-time visits of event c or ¬d, respectively); they descend in the DAG. Dotted edges depict winning strategies for each subgame. An overall winning strategy is obtained by using additional memory to keep track of the current subgame and by always moving according to the winning strategy for the current subgame.

Lemma 5. *The games G and G_2 are equivalent.*

Sketch. For one direction, use a winning strategy for G to play in G_2, simply ignoring memory values. By construction, the resulting plays in G_2 eventually stay within one subgame G_L and are winning by Lemma 4. For the converse direction, a winning strategy for G_2 provides, for each set L of local events, a strategy to play in the subgame G_L. In G, let system player always follow the strategy for the subgame that corresponds to the local events visited so far. The resulting plays in G are again winning by Lemma 4. □

Remark 5 (Strategy extraction). *For each EL subgame G_L, construct a winning strategy σ_L (according to Remark 1). An overall strategy σ for G uses auxiliary memory $2^{\Gamma_{\mathsf{F},\mathsf{G}}}$ to keep track of the local events that have been satisfied / violated so far. This memory identifies, at each point, a subgame G_L. Define σ to always play according to σ_L, where L is the current content of the auxiliary memory.*

Due to its particular DAG structure, the game G_2 can be solved by solving the subgames G_L (played over arena A'_L with objective φ_L) individually, starting from the bottom game G_{Γ_F}. Once a subgame G_L has been solved, all edges in the remaining subgames that lead to a node $v \in G_L$ are marked as winning or losing, depending on whether v is winning or losing in G_L. Then G_L is removed from the DAG and one of the remaining bottom subgames is picked and solved in turn. Thus G_2 can be solved by solving at most 2^d EL games, each of which has at most n nodes, m edges, and an objective with at most k EL events. The overall time complexity of solving G_2 in this way is in $\mathcal{O}(2^d \cdot m \cdot k! \cdot n^k)$. Note that we have $m \leq n^2$.



LydiaSyft+
LTL$_f$+/PPLTL+ formula Ψ
parser
Boolean formula over $\mathbb{Q}_i\Phi_i$
Sect. 3
Sect. 5
EL game $\mathcal{A}_\psi$
MP game $\mathcal{A}'_\psi$
Thm. 1
Sect. 6
DAG of EL games
Thm. 1
Thm. 4
output: "(un)realizable", strategy

Figure 1: Overall structure of the implementation

Theorem 4. *Manna-Pnueli games with n nodes, k Emerson-Lei events and d local events can be solved in time $\mathcal{O}(k! \cdot n^{k+2} \cdot 2^d) \in 2^{\mathcal{O}(d+k\log n)}$; winning strategies require at most $k! \cdot 2^d$ memory values.*

The above bounds improve to a solution time $\mathcal{O}(m \cdot k! \cdot n^k) \in 2^{\mathcal{O}(k \log n)}$ and strategy size $k!$ for games over arenas that have memory for local events. Thus the solution complexity for MP games with memory for local events is the same as for EL games.

The dependency on the number of events in Theorem 1 (and in Corollary 1) is factorial while the dependency on the number of local events in Theorem 4 is exponential.

7 Implementation

We implemented a prototype LydiaSyft+ (Hausmann et al. 2025) (see Figure 1) that realizes both our method and the existing approach based on a reduction to Emerson-Lei games as described in Section 3. While LydiaSyft+ is the first implementation of an LTL$_f$+/PPLTL+ synthesizer, it also enables a direct empirical comparison between the two approaches.

Synthesis via EL solver. We implemented an LTL$_f$+/PPLTL+ synthesis procedure LydiaSyft+-EL based on a reduction to EL games, using the symbolic EL solution algorithm. For LTL$_f$+ synthesis, we leverage the existing LTL$_f$-to-DFA translator LydiaSyft (Zhu and Favorito 2025) as a backend to translate LTL$_f$ formula components to explicit-state deterministic finite automata (DFAs). These DFAs are tailored according to the reduction in (Aminof et al. 2025b) (cf. Step 1. in Section 3) and transformed into symbolic representation using Binary Decision Diagrams (BDDs) (Bryant 1992; Zhu et al. 2017b). For PPLTL+ synthesis, we implemented a direct symbolic DFA construction based on the method described in (Bonassi et al. 2023b), which also serves as the first direct translator from PPLTL+ to symbolic

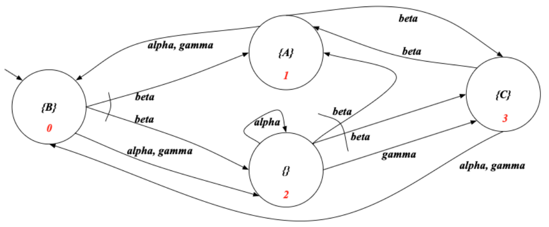


Figure 2: A simple planning domain

DFA. Since all DFAs are in symbolic representation, we can avoid explicitly computing their cross-product. Instead, we perform an on-the-fly product construction during the EL game solving, which is carried out using symbolic fixpoint computations as in (Zhu et al. 2017b; Hausmann, Lehaut, and Piterman 2024).

Synthesis via MP solver. We also implemented the MP based approach to LTL$_f$+/PPLTL+ synthesis, described in Section 5. Given an MP game $G = (V, E, (\Gamma, \gamma, \varphi))$, the core component in solving the game involves constructing a directed acyclic graph (DAG) of EL games. These EL games are then solved in a bottom-up order along the DAG. Note that each EL game in the DAG is obtained by evaluating the local events in the MP condition φ. We encode φ symbolically using BDDs for efficient evaluation and simplification.

8 Experiments

All experiments were conducted on a laptop running 64-bit Ubuntu 22.04.4 LTS, with an i5-1245U CPU with 12 cores and 32GB of memory. Time-out was set to one hour.

Comparing LTL$_f$+ and LTL$_f$. We construct the benchmark example based on a simple planning problem in a nondeterministic domain, illustrated in Figure 2. In this domain, a robot starts at state 0 and chooses an action from the set $\{$*alpha, beta, gamma*$\}$ to move to the next state. However, the actual next state also depends on the response of the environment, which is nondeterministic to the robot. For example, if the robot is in state 0 and takes action *beta*, the robot may move to either state 1 or state 2. We can describe this domain in either LTL$_f$ or LTL$_f$+. The core is that we just use the formula to specify safety conditions that represent the transitions of the domain following the robot action and the environment response. In this case, we have an LTL$_f$ formula Φ_D caturing the traces of the domain, and an LTL$_f$ formula Φ_{act} specifying that exactly one action is executed at each step. The objective of the robot is given by the LTL$_f$ formula $\Phi_{goal} = \mathsf{F}(A \wedge \mathsf{F}(B \wedge \mathsf{X}\,\mathsf{false}))$, specifying that the robot must eventually visit A and then visit B at the end of the finite trace; recall that X stands for "weak next". Ultimately, we have the overall LTL$_f$ specification describing the planning problem as: $\Phi = \Phi_{act} \wedge (\Phi_D \rightarrow \Phi_{goal})$.

To construct the corresponding LTL$_f$+ specification for the same domain, we note that both Φ_{act} and Φ_D represent safety conditions. This allows us to simply add universal quantification $\forall$ in front of Φ_{act} and Φ_D. For the objective, instead of simply requiring the robot to visit A and then (or meanwhile) B once, we make it more difficult by requiring that the robot should "visit A and subsequently (or simultaneously) visit B" infinitely often. This leads to the LTL$_f$+ specification $\Psi = \forall\Phi_{act} \wedge (\forall\Phi_D \rightarrow \forall\exists\Phi_{goal})$.

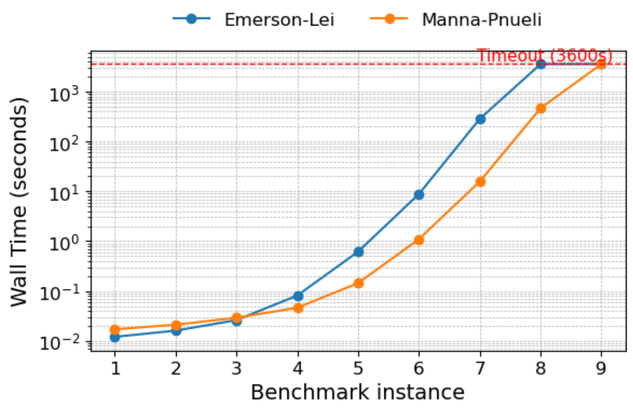


Figure 3: Runtime of LTL$_f$+ synthesis, counter-games

We run both problems using LydiaSyft for LTL$_f$ synthesis on Φ and LydiaSyft+ (both using the MP solver and the EL solver) for LTL$_f$+ synthesis on Ψ, and observe that both solvers complete within a few milliseconds. Recall that Ψ encodes a more complex objective involving a recurrence formula. This confirms that there is no overhead in constructing the arena for LTL$_f$+ respect to LTL$_f$, and that the more complex fixpoint computation required for LTL$_f$+ does not compromise performance.

Comparing MP solver and the EL solver. We next compare the scalability of the MP solver and the EL solver. The first benchmark is inspired by the counter-game introduced in (Zhu et al. 2020). We adapt the basic game setting which involves an n-bit binary counter. We use $\forall\Phi_B$ to describe the behaviour of the counter, including both the value transitions and the response of the environment to grant requests to increment the counter. The formula $\forall\exists\Phi_{add}$ represents agent requests to increase the counter value, while $\exists\Phi_g$ captures the goal condition that the counter eventually reaches its maximum value, i.e., all bits are set to 1. The LTL$_f$+ formulas in this series then are defined as

$$\Psi = \forall(\Phi_{init} \wedge \Phi_{inc} \wedge \Phi_{B_i}) \rightarrow (\forall\exists\Phi_{add} \rightarrow \exists\Phi_g), \text{ where}$$

$$\Phi_{init} = \neg c_0 \wedge \ldots \wedge \neg c_{n-1} \wedge \neg b_0 \wedge \ldots \neg b_{n-1}$$

$$\Phi_{inc} = \mathsf{G}(add \rightarrow (\mathsf{X}(c_0) \wedge \mathsf{XX}(c_0) \wedge \mathsf{XXX}(c_0))$$

$$\Phi_{B_i} = \begin{cases} ((\neg c_i \wedge \neg b_i) \rightarrow \mathsf{X}(\neg b_i \wedge \neg c_{i+1})) \wedge \\ ((\neg c_i \wedge b_i) \rightarrow \mathsf{X}(b_i \wedge \neg c_{i+1})) \wedge \\ ((c_i \wedge \neg b_i) \rightarrow \mathsf{X}(b_i \wedge \neg c_{i+1})) \wedge \\ ((c_i \wedge b_i) \rightarrow \mathsf{X}(\neg b_i \wedge c_{i+1})) \end{cases}$$

$$\Phi_{add} = \mathsf{F}(add \wedge \mathsf{X}(\mathsf{false}))$$

$$\Phi_g = \mathsf{F}(b_0 \wedge \ldots \wedge b_{n-1} \wedge \mathsf{X}(\mathsf{false}))$$

In this benchmark, all variables b_i and c_i are environment variables, since they are used to describe the status of the counter. The variable add is an agent variable, used to request an increment of the counter. All the formulas in this

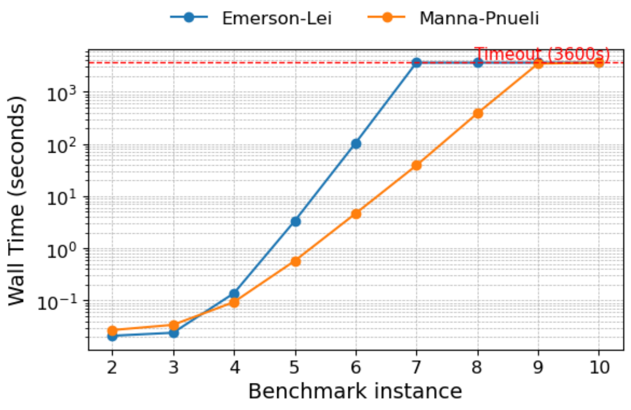


Figure 4: Runtime of LTL$_f$+ synthesis, $(\forall\exists \rightarrow \exists)$-*Pattern*

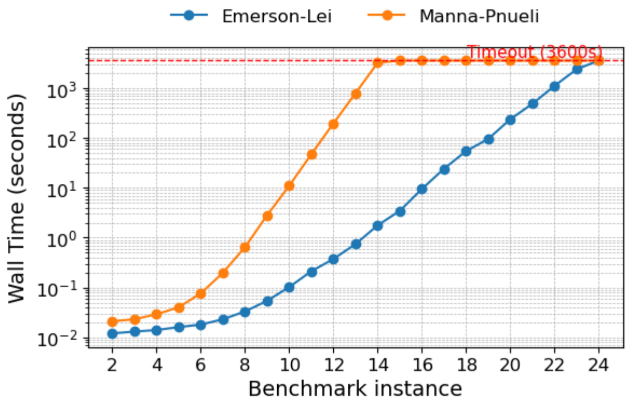


Figure 5: Runtime of LTL$_f$+ synthesis, $\exists$-*Pattern*

benchmark are realizable as well. Figure 3 shows that the MP solver significantly outperforms the EL solver, solving more instances with lower runtime.

To check scaling as the game conditions increase, we construct a benchmark consisting of formulas formed as conjunctions of subformulas in the form of $\forall\exists \rightarrow \exists$, referred to as $(\forall\exists \rightarrow \exists)$-*Pattern*, and is defined as:

$$\Psi_n = \bigwedge_{1\leq i\leq n}(\,\forall\exists\,\mathsf{F}(e_i \wedge \mathsf{X}(\mathsf{false})) \rightarrow \exists\,\mathsf{F}(a_i \wedge \mathsf{X}(\mathsf{false}))\,)$$

Each LTL$_f$ formula component is very simple. But the number of recurrence and guarantee formulas increases with n. Each a_i is an agent variable, while each e_i is an environment variable; all formulas in this series are realizable.

Figure 4 shows the running time on the $(\forall\exists \rightarrow \exists)$-Pattern benchmarks. The MP solver is able to solve more instances within the time limit, and consistently takes less time than the EL solver, demonstrating the superior performance of the MP solver.

A natural question is whether there are cases for which the EL solver performs better than the MP solver in practice. We construct a specific benchmark, the instances of which contain only guarantee formulas ($\exists$). More specifically, the formulas, referred to as $\exists$-*Pattern*, are defined as follows:

$$\Psi_n = \bigwedge_{1\leq i\leq n}\exists\,\mathsf{F}((a_i \vee e_i) \wedge \mathsf{X}(\mathsf{false}))$$

Since instances contain only local events, one might expect the MP solver to outperform the EL solver. However, the results in Figure 5 show that the EL solver is able to solve a substantially larger number of instances within the time limit and consistently takes less time than the MP solver on instances that both solvers can handle. To understand why this happens, one has to consider that while the EL solver computes a nested fixpoint, having to reduce $\exists(\cdot)$ to $\forall\exists(\cdot)$, the automata construction for $\exists(\cdot)$ introduces loops on the accepting states, thus shortcutting the corresponding nested fixpoint. On the other hand, the MP solver must first construct a DAG, where each node corresponds to a relatively (but in this case not significantly) simpler EL game. Moreover, the MP solver can only conclude "realizable" when all these games have been solved, resulting in greater overall computational overhead.

9 Discussion

We have implemented synthesizers for LTL$_f$+ (and PPLTL+) based on reductions to EL and MP games. We have introduced MP games and showed that theoretically solvers based on MP games have better computational characteristics than EL solvers. In practice, experimental results indicate that while the MP solver often performs better than the EL solver, this is not always the case, since there is a trade-off between efficiency gains obtained by solving DAGs of simpler EL games versus the higher cost associated to the construction of these DAGs. This indicates that fine-tuning may be needed to strike the perfect balance between which local events should be treated by the DAG construction and which should be left to the EL solver. We leave this for future work.

We also plan to explore a recently proposed alternative game construction that combines the benefits of the EL and the MP reductions, eliding the DAG construction by integrating local events into the existing EL events, instead of adding auxiliary EL events (Duret-Lutz 2025).

We conclude by briefly mentioning *obligation properties*, that is, Boolean combinations of guarantee, $\exists(\cdot)$, and safety, $\forall(\cdot)$ formulas. Synthesis for these formulas reduces to the solution of MP games without EL events; asymptotically, the solution of such games is not harder than the solution of reachability games, indicating that particularly good performance can be obtained for the corresponding fragments of LTL$_f$+ (and PPLTL+). Again, we leave this for future work.

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, the EPSRC project LaST (No. EP/Z003121/1), and by the Swedish research council (VR) through project No. 2020-04963. Gianmarco Parretti is supported by the Italian National Phd on AI at the University of Rome "La Sapienza".

References

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

Aminof, B.; De Giacomo, G.; Di Stasio, A.; Francon, H.; Rubin, S.; and Zhu, S. 2025a. LTL synthesis under environment specifications for reachability and safety properties. *Inf. Comput.* 303:105255.

Aminof, B.; De Giacomo, G.; Rubin, S.; and Vardi, M. Y. 2025b. LTLf+ and PPLTL+: Extending LTLf and PPLTL to infinite traces. In *IJCAI*.

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

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

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

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

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

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

Bansal, S.; De Giacomo, G.; Di Stasio, A.; Li, Y.; Vardi, M. Y.; and Zhu, S. 2022. Compositional safety LTL synthesis. In *VSTTE*.

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

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

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

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* 24(3):293–318.

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

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

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

De Giacomo, G., and Favorito, M. 2021. Compositional approach to translate LTL$_f$/LDL$_f$ into deterministic finite automata. In *ICAPS*.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of fond planning for LTL$_f$/LDL$_f$ goals. In *IJCAI*.

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

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

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

De Giacomo, G.; Stasio, A. D.; Tabajara, L. M.; Vardi, M. Y.; and Zhu, S. 2022. Finite-trace and generalized-reactivity specifications in temporal synthesis. *Formal Methods Syst. Des.* 61(2):139–163.

Duret-Lutz, A. 2025. Personal communication.

Dziembowski, S.; Jurdzinski, M.; and Walukiewicz, I. 1997. How much memory is needed to win infinite games? In *LICS*.

Ehlers, R.; Lafortune, S.; Tripakis, S.; and Vardi, M. Y. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discret. Event Dyn. Syst.* 27(2):209–260.

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

Fijalkow, N.; Bertrand, N.; Bouyer-Decitre, P.; Brenguier, R.; Carayol, A.; Fearnley, J.; Gimbert, H.; Horn, F.; Ibsen-Jensen, R.; Markey, N.; Monmege, B.; Novotný, P.; Randour, M.; Sankur, O.; Schmitz, S.; Serre, O.; and Skomra, M. 2023. Games on graphs. *arXiv* 2305.10546.

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

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

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

Hausmann, D.; Zhu, S.; Parretti, G.; Weinhuber, C.; De Giacomo, G.; and Piterman, N. 2025. Lydiasyft+. https://github.com/Shufang-Zhu/LydiaSyftPlus. Version 1.0.0.

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

Kankariya, Y., and Bansal, S. 2024. Decompositions in compositional translation of LTLf to DFA. In *AAAI*.

Klarlund, N.; Møller, A.; and Schwartzbach, M. I. 2002. MONA implementation secrets. *Int. J. Found. Comput. Sci.* 13(4):571–586.

Lichtenstein, O.; Pnueli, A.; and Zuck, L. D. 1985. The glory of the past. In *Logic of Programs*, 196–218.

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

McNaughton, R. 1993. Infinite games played on finite graphs. *Ann. Pure Appl. Logic* 65(2):149–184.

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

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

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

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

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

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

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

Zhu, S., and Favorito, M. 2025. LydiaSyft: A compositional symbolic synthesis framework for LTL$_f$ specifications. In *TACAS*.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017a. A symbolic approach to Safety LTL synthesis. In *HVC*.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017b. Symbolic LTL$_f$ Synthesis. In *IJCAI*.

Zhu, S.; De Giacomo, G.; Pu, G.; and Vardi, M. Y. 2020. LTLf synthesis with fairness and stability assumptions. In *AAAI*.

Zhu, S.; Tabajara, L. M.; Pu, G.; and Vardi, M. Y. 2021. On the power of automata minimization in temporal synthesis. In *GandALF*.

Zielonka, W. 1998. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.* 200(1-2):135–183.