Incremental LTL$_f$ Synthesis

Giuseppe De Giacomo[1,2], Yves Lespérance[3], Gianmarco Parretti[2], Fabio Patrizi[2], Moshe Y. Vardi[4]

[1] University of Oxford
[2] University of Rome "La Sapienza"
[3] York University
[4] Rice University
giuseppe.degiacomo@cs.ox.ac.uk, lesperan@yorku.ca, {parretti, patrizi}@diag.uniroma1.it, vardi@cs.rice.edu

Abstract

In this paper, we study incremental LTL$_f$ synthesis – a form of reactive synthesis where the goals are given incrementally while in execution. In other words, the protagonist agent is already executing a strategy for a certain goal when it receives a new goal: at this point, the agent has to abandon the current strategy and synthesize a new strategy still fulfilling the original goal, which was given at the beginning, as well as the new goal, starting from the current instant. In this paper, we formally define the problem of incremental synthesis and study its solution. We propose a solution technique that efficiently performs incremental synthesis for multiple LTL$_f$ goals by leveraging auxiliary data structures constructed during automata-based synthesis. We also consider an alternative solution technique based on LTL$_f$ formula progression. We show that, in spite of the fact that formula progression can generate formulas that are exponentially larger than the original ones, their minimal automata remain bounded in size by that of the original formula. On the other hand, we show experimentally that, if implemented naively, i.e., by actually computing the automaton of the progressed LTL$_f$ formulas from scratch every time a new goal arrives, the solution based on formula progression is not competitive.

1 Introduction

In this paper, we study *incremental* LTL$_f$ synthesis – a form of reactive synthesis where the goals are given incrementally while in execution. Reactive synthesis was originally studied in Formal Methods for Linear Temporal Logic on infinite traces (LTL) (Pnueli 1977; Pnueli and Rosner 1989) and uses techniques drawn from model checking. More recently, synthesis has been studied for Linear Temporal Logic on finite traces (De Giacomo and Vardi 2013), where the agent must eventually stop after completing its goal (De Giacomo and Vardi 2015). LTL$_f$ synthesis has some nice computational characteristics that allow one to exploit symbolic techniques typical of model checking (Baier and Katoen 2008), to obtain symbolic solvers significantly more scalable than those for LTL, see (Zhu et al. 2017; Bansal et al. 2020; De Giacomo and Favorito 2021; Kankariya and Bansal 2024; Zhu and Favorito 2025; Duret-Lutz et al. 2025).

In AI, reactive synthesis is essentially a variant of strong planning for temporally extended goals in Fully Observable Nondeterministic (FOND) domains (Cimatti et al. 2003; Bacchus and Kabanza 2000; Baier, Fritz, and McIlraith 2007; De Giacomo and Rubin 2018; Camacho and McIlraith 2019; De Giacomo, Parretti, and Zhu 2023).

Incremental synthesis deals with the case where an agent is already executing a strategy for a certain goal when it gets an additional new goal: at this point, the agent has to abandon the strategy for the original goal that it is executing and synthesize a new strategy that fulfills both the original and new goal. Crucially, the original goal has to be fulfilled starting from the beginning, through the current history, while the new goal starts from the current instant. In general, incremental synthesis may be needed several times during the execution.

Incremental synthesis shares some similarities with compositional approaches to synthesis, where the LTL$_f$ specification is a conjunction of subformulas that are handled separately (Bansal et al. 2020, 2022). Here, however, the conjunction involves formulas to be evaluated over traces that start at different time points. Incremental synthesis also shares some similarities with live synthesis, where the synthesis is performed after a certain history and not in the initial state (Finkbeiner, Klein, and Metzger 2022; Zhu and De Giacomo 2022).

Obviously, when adopting a new goal, we must guarantee the realizability of the new goal together with the previous ones. In case of conflict, we need to decide whether to forfeit the adoption of the new goal or drop some of the old ones to keep the adopted goals realizable. While we do not discuss how to resolve this conflict in the paper, leaving it to future work, our techniques do indeed assess whether the conflict is present.

In this paper, we formally define the problem of incremental synthesis and study its solution. In particular, we propose a solution technique that leverages auxiliary data structures constructed during automata-based synthesis. We show its soundness, completeness, and use it to characterize the worst-case computational complexity of the incremental LTL$_f$ synthesis as 2EXPTIME-complete – as for standard LTL$_f$ synthesis. We then evaluate its effectiveness experimentally using a prototype implementation.

We also study an alternative way of solving incremental synthesis by exploiting LTL$_f$ formula progression, i.e., the fixpoint characterization of temporal formulas (Gabbay et al. 1980; Manna 1982; Emerson 1990; Bacchus and Kabanza 2000; De Giacomo et al. 2022), which allows to recursively split an LTL_f formula into a propositional formula on the *current* instant and a temporal formula to be checked at the *next* instant. By applying formula progression, we can reduce the problem of incremental synthesis to standard synthesis for the conjunction of the progressed original goal and the new one. Progressed formulas can become exponentially larger than the initial ones, so one could think that incremental synthesis using this technique is 3EXPTIME. Instead, here we show that the automata of progressed formulas (once minimized) are bounded by the automata of their corresponding original formulas. Hence, also using LTL_f progression, incremental synthesis can be solved in 2EXPTIME. However, the LTL_f progression-based technique, if implemented naively, i.e., by actually computing the automata of the progressed formulas every time a new goal arrives, is not really competitive, as we show empirically in Section 7. In fact, the automata-based solution technique we present here can be seen as a clever implementation of the solution based on formula progression which takes advantage of automata caching.

2 Preliminaries

We consider specifications (aka *formulas*) in Linear Temporal Logic on *finite traces* (LTL_f) (De Giacomo and Vardi 2013). LTL_f is a formalism widely used in AI to specify temporal properties over finite-time horizons, e.g., in planning, temporally extended goals (Camacho et al. 2017; De Giacomo and Rubin 2018; Camacho and McIlraith 2019) as well as state/action trajectory constraints (Bonassi et al. 2021; Bonassi, Gerevini, and Scala 2022).

We require LTL_f specifications to be in Negation Normal Form (NNF): negation can appear in front of atomic propositions only. This does not cause any loss of generality: every LTL_f formula can be transformed into NNF in linear time. Formally, given a set of *atomic propositions* (aka *atoms*) P, the syntax of LTL_f formulas φ in NNF is:

$$\varphi := p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \circ\varphi \mid \bullet\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \varphi_1 \mathcal{R} \varphi_2$$

Where $p \in P$. Boolean operators include: *negation* ($\neg$); *disjunction* ($\vee$); and *conjunction* ($\wedge$). Temporal operators include: *next* ($\circ$); *weak next* ($\bullet$); *until* ($\mathcal{U}$); and *release* ($\mathcal{R}$). Additional operators include: standard operators of propositional logic, such as $true$, $false$, *implication* ($\supset$), and *equivalence* ($\equiv$); and temporal operators defined as abbreviations, such as *eventually* ($\Diamond\varphi \doteq true\,\mathcal{U}\,\varphi$), and *always* ($\Box\varphi \doteq false\,\mathcal{R}\,\varphi$). The size of φ, denoted $|\varphi|$, is the number of subformulas in its abstract syntax tree.

LTL_f formulas are interpreted over finite traces $\tau \in (2^P)^*$ of propositional interpretations over P. The empty trace is ϵ. We denote by $\tau_i \in 2^P$ the propositional interpretation in the i-th time step of τ and by $|\tau|$ its length. Given the traces $\tau = \tau_0 \cdots \tau_n$ and $\tau' = \tau'_0 \cdots \tau'_m$, their concatenation is the trace $\tau \cdot \tau' = \tau_0 \cdots \tau_n \cdot \tau'_0 \cdots \tau'_m$. Given an LTL_f formula φ, a trace τ, and an index i such that $0 \leq i < |\tau|$, the following inductive definition formalizes when τ *satisfies* φ at i:

- $\tau, i \models p$ iff $p \in \tau_i$, for $p \in P$;
- $\tau, i \models \neg p$ iff $p \notin \tau_i$, for $p \in P$;
- $\tau, i \models \varphi_1 \vee \varphi_2$ iff $\tau, i \models \varphi_1$ or $\tau, i \models \varphi_2$;
- $\tau, i \models \varphi_1 \wedge \varphi_2$ iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \circ\varphi$ iff $i < |\tau| - 1$ and $\tau, i+1 \models \varphi$;
- $\tau, i \models \bullet\varphi$ iff $i < |\tau| - 1$ implies $\tau, i+1 \models \varphi$;
- $\tau, i \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists j such that $i \leq j < |\tau|$ and $\tau, j \models \varphi_2$, and, for every k such that $i \leq k < j$ we have $\tau, k \models \varphi_1$;
- $\tau, i \models \varphi_1 \mathcal{R} \varphi_2$ iff either: (1) there exists j such that $i \leq j < |\tau|$ and $\tau, j \models \varphi_1$ and, for every k such that $i \leq k \leq j$, we have $\tau, k \models \varphi_2$; *or* (2) for every k such that $i \leq k < |\tau|$, we have $\tau, k \models \varphi_2$.

A finite trace τ satisfies φ, denoted $\tau \models \varphi$, if $\tau, 0 \models \varphi$.

LTL_f *(reactive) synthesis* concerns finding a strategy to satisfy an LTL_f goal specification. Goals are expressed as LTL_f formulas over $P = \mathcal{Y} \cup \mathcal{X}$, where $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of atoms under the control of the agent and environment respectively. Traces over $\mathcal{Y} \cup \mathcal{X}$ are denoted $\tau = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots$, where $Y_i \in 2^{\mathcal{Y}}$ and $X_i \in 2^{\mathcal{X}}$ for every $i \geq 0$. Infinite traces of this form are called *plays*; finite traces are also called *histories*.

An *(agent) strategy* is a function $\sigma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ that maps (possibly empty) sequences of environment moves to an agent move. The domain of σ includes the empty trace ϵ as we assume that the agent moves first – as in planning. A play $\tau = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots$ is σ-consistent if: *(i)* $Y_0 = \sigma(\epsilon)$; and *(ii)* $Y_i = \sigma(X_0 \cdots X_{i-1})$ for every $i > 0$. That a history h is σ-consistent is defined analogously. The set of σ-consistent plays is denoted $Play(\sigma)$.

A strategy σ is winning for φ if, for every play $\tau \in Play(\sigma)$, there exists a finite prefix τ^k that satisfies φ, i.e., $\tau^k \models \varphi$. LTL_f synthesis is the problem of finding a winning strategy for φ, if any exists. If there exists a winning strategy for φ, we say that synthesis is *realizable*; otherwise, it is *unrealizable*. LTL_f synthesis is 2EXPTIME-complete wrt φ (De Giacomo and Vardi 2015).

3 Incremental Synthesis

In this paper, we study *incremental* LTL_f *synthesis*. In incremental synthesis, the agent does not know the goals in advance – as in standard synthesis – but receives them online, during execution.

Specifically, consider an agent that is executing a strategy σ – previously synthesized for a goal φ_{org} – which has so far generated a history h. Assume that at the end of h a new goal φ_{new} arrives. On the one hand, the agent cannot rely on the original strategy σ, as it was synthesized for φ_{org}, without even knowing φ_{new}; on the other hand, the agent cannot simply synthesize a new strategy σ' for both φ_{org} and φ_{new}, since σ' would restart satisfying φ_{org} from the current instant, without taking into account that after the history h only part of the original goal φ_{org} remains to be realized. Instead, we want σ' to: (R1) guarantee the satisfaction of φ_{new}; and (R2) continue with the satisfaction of φ_{org} from where σ left, i.e., taking into account the history h. We formalize this intuition below.

Requirement (R1) can be easily captured by requiring the new strategy to be winning for φ_{new}. Regarding requirement (R2): we formalize a new notion that captures what it means for a strategy σ to be winning for φ taking into account that a history h has occurred. That is, σ executed at h only generates plays that satisfy φ from the beginning. Formally:

Definition 1 *Let φ be an* LTL_f *goal and h a history. A strategy σ is winning for φ* assuming h *if, for every $\tau \in Play(\sigma)$, we have that $\tau' = h \cdot \tau$ has a finite prefix that satisfies φ.*

Having notions that capture requirements (R1) and (R2), we can formally define the incremental synthesis problem:

Definition 2 *Let: φ_{org} be an original* LTL_f *goal; h a history; and φ_{new} a new goal. Incremental synthesis is the problem of computing a strategy σ such that:*
1. σ is winning for φ_{org} assuming h; and
2. σ is winning for φ_{new};
if any exists. In this latter case, we say that incremental synthesis is realizable*; otherwise, it is* unrealizable*.*

Incremental synthesis generalizes standard synthesis. Indeed, we have the following: synthesis for φ is equivalent to incremental synthesis for $\varphi_{org} = \varphi$, $h = \epsilon$, and $\varphi_{new} = true$ (or alternatively for $\varphi_{org} = true$, $h = \epsilon$, and $\varphi_{new} = \varphi_{org}$). Given that standard synthesis is 2EXPTIME-hard (De Giacomo and Vardi 2015), we obtain the hardness of incremental synthesis:

Theorem 1 *Incremental synthesis is 2EXPTIME-hard wrt φ_{org} and φ_{new}.*

In fact, incremental synthesis also preserves the realizability for the original goal. Say that σ_{org} is a winning strategy for φ_{org} that the agent was using when h occurred and φ_{new} arrived; also, say that σ_{new} is a strategy that satisfies Definition 2. Consider the overall strategy obtained by combining the executions of σ_{org} and σ_{new}: we have that this strategy is winning for φ_{org} – i.e., the *realizability of φ_{org} is preserved*. The intuition is as follows: before h occurred, the agent was using σ_{org}, that always guarantees the satisfaction of φ_{org}; after h occurred, the agent uses σ_{new}, that guarantees the satisfaction of φ_{org} in every extension of h. Formally: denote by $\sigma_{org}[h \leftarrow \sigma_{new}]$ the strategy that agrees with σ_{org} everywhere, except in h and all its extensions, where it agrees with σ_{new}. Then we have:

Theorem 2 *The strategy $\sigma_{org}[h \leftarrow \sigma_{new}]$, where σ_{org} is winning for φ_{org} and consistent with h, and σ_{new} is a strategy that satisfies Definition 2, is winning for φ_{org}.*

We conclude this section by observing that Definition 2 can be easily generalized to an arbitrary number of goals. For instance, consider the original goal φ_{org}, and two new successive goals φ_{new_1}, arriving at history h_1 and φ_{new_2} arriving at history $h_1 \cdot h_2$. Incremental synthesis in this case consists computing a strategy that is: (*i*) winning for φ_{org} assuming $h_1 \cdot h_2$; (*ii*) winning for φ_{new_1} assuming h_2; and (*iii*) winning for φ_{new_2}.

4 Automata-Based Synthesis Technique

In this section, we present an automata-based technique to solve incremental synthesis. The technique is based on solving reachability games played over suitable deterministic finite automata, which we briefly review below.

A *deterministic finite automaton* (DFA) is a tuple $\mathcal{A} = (\Sigma, S, \iota, \delta, F)$, where: Σ is a finite input alphabet; S is a finite set of states; $\iota \in S$ is the initial state; $\delta : S \times \Sigma \to S$ is the transition function; and $F \subseteq S$ is the set of final states. The size of $\mathcal{A}$ is $|S|$. For convenience, we extend δ to a function $\delta : S \times \Sigma^* \to S$ over finite traces $\alpha = \alpha_0 \cdots \alpha_n$, inductively defined as follows: (*i*) $\delta(s, \epsilon) = s$; and (*ii*) $\delta(s, \alpha_0 \cdots \alpha_n) = \delta(\delta(s, \alpha_0 \cdots \alpha_{n-1}), \alpha_n)$. A trace α is *accepted* if $\delta(\iota, \alpha) \in F$. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of traces that $\mathcal{A}$ accepts. A DFA $\mathcal{A}$ is *minimal* if there is no other DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and $|\mathcal{A}'| < |\mathcal{A}|$. A DFA can be minimized in polynomial time by using an algorithm that repeatedly collapses into one states that are indistinguishable wrt to all possible future inputs. For the languages that we consider in this paper, called *regular languages*, there exists a unique minimal DFA (Sipser 1997).

Theorem 3 *(De Giacomo and Vardi 2015) Given an* LTL_f *formula φ, we can construct a* DFA *$\mathcal{A}_\varphi$ of size at most double exponential in that of φ and whose language is the set of finite traces that satisfy φ.*

We note, however, that the worst-case doubly-exponential blowup while constructing DFAs of LTL_f formulas is rare in practice, and the construction scales well, see, e.g., (De Giacomo and Vardi 2015; Zhu et al. 2017; Bansal et al. 2020; De Giacomo and Favorito 2021).

For synthesis, we see such a DFA as a game. Specifically, a DFA game is a DFA $\mathcal{G} = (\Sigma, S, \iota, \delta, F)$ with input alphabet $\Sigma = 2^{\mathcal{Y} \cup \mathcal{X}}$, where $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of *atoms* under control of agent and environment, respectively. A *game strategy* is a function $\kappa : S \to 2^{\mathcal{Y}}$ that maps game states to agent moves. The notion of play in Section 2 also applies to DFA games. A play $\tau = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots$ is κ-consistent if: (*i*) $Y_0 = \kappa(\iota)$; and (*ii*) $Y_i = \kappa(\delta(\iota, \tau^{i-1}))$ for every $i > 0$, where $\tau^{i-1} = (Y_0 \cup X_0) \cdots (Y_{i-1} \cup X_{i-1})$. The set of κ-consistent plays is $Play(\kappa)$. A game strategy κ is winning in $\mathcal{G}$ if, for every play $\tau \in Play(\kappa)$, there exists a finite prefix τ^k that is accepted by $\mathcal{G}$. That is: in a DFA game $\mathcal{G}$ the agent has to visit at least once the set of final states – thus generating only traces accepted by $\mathcal{G}$. The winning region W is the set of states s where the agent has a winning strategy in the game $\mathcal{G}' = (\Sigma, S, s, \delta, F)$, i.e., the same game as $\mathcal{G}$, but with new initial state s. There exists a winning game strategy in $\mathcal{G}$ iff $\iota \in W$. Solving a DFA game $\mathcal{G}$ is the problem of computing the winning region W and a winning game strategy κ, if any exists. DFA games can be solved in polynomial time by a backward-induction algorithm that performs a least fixpoint computation over the state space of the game (Apt and Grädel 2011). Given a game $\mathcal{G}$ and a winning game strategy $\kappa : S \to 2^{\mathcal{Y}}$, we can construct in polynomial time a winning strategy $\sigma_{(\mathcal{G},\kappa)} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ for the corresponding LTL_f formula as follows: (*i*) $\sigma_{(\mathcal{G},\kappa)}(\epsilon) = \kappa(\iota)$; and (*ii*) for every $\tau = (Y_0 \cup X_0) \cdots (Y_n \cup X_n)$, define $\sigma_{(\mathcal{G},\kappa)}(X_0 \cdots X_n) = \kappa(\delta(\iota, \tau))$. The pair $(\mathcal{G}, \kappa)$ is a *transducer* that implements the strategy $\sigma_{(\mathcal{G},\kappa)}$.

Now, we present the algorithm for incremental synthesis. Intuitively, the algorithm consists of four steps that do the following: Step (1) constructs the DFAs that accept the traces that satisfy the original and new goal; Steps (2) and (3) manipulate and combine these DFAs so to obtain a DFA game that admits a winning strategy iff incremental synthesis is realizable; Step (4) solves this game to decide whether incremental synthesis is realizable – in which case it returns a strategy to satisfy both the original and new goal.

The algorithm uses two polynomial operators to manipulate DFAs, *product* and *progression*: the former combines several DFAs into one whose language is the intersection of the languages of its operands; the latter progresses the initial state of a DFA through a history h, so that the progressed DFA accepts exactly the traces accepted by the input DFA and that contain h as a prefix. Formally:

Definition 3 (DFA Product) *Let* $\mathcal{A}_1, \cdots, \mathcal{A}_n$ *be* DFA*s where* $\mathcal{A}_i = (\Sigma, S_i, \iota_i, \delta_i, F_i)$ *for every* i. *Their product is the* DFA $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n = (\Sigma, S, \iota, \delta, F)$, *with: alphabet* Σ*; state space* $S = S_1 \times \cdots \times S_n$*; initial state* $\iota = (\iota_1, \cdots, \iota_n)$*; transition function* $\delta((s_1, \cdots, s_n), w) = (\delta(s_1, w), \cdots, \delta(s_n, w))$ *for every* $w \in \Sigma$*; final states* $F = F_1 \times \cdots \times F_n$*; and language* $\mathcal{L}(\mathcal{A}) = \bigcap_i \mathcal{L}(\mathcal{A}_i)$.

Definition 4 (DFA Progression) *Let* $\mathcal{A} = (\Sigma, S, \iota, \delta, F)$ *be a* DFA *and* h *a history. The* DFA *progression of* $\mathcal{A}$ *through* h *is* DFA $\mathcal{A}_h = (\Sigma, S, \delta(\iota, h), \delta, F)$ *– i.e., the same as* $\mathcal{A}$*, but with initial state* $\delta(\iota, h)$*, so that its language is* $\mathcal{L}(\mathcal{A}_h) = \{h \cdot \tau \mid \tau \in \mathcal{L}(\mathcal{A})\}$.

We give below the algorithm for incremental synthesis.

Algorithm 1. Consider an instance of incremental synthesis: φ_{org} is the original LTL$_f$ goal; h the history; and φ_{new} the new LTL$_f$ goal.

(1) Construct the DFAs $\mathcal{A}_{org}$ and $\mathcal{A}_{new}$ that accept the traces that satisfy φ_{org} and φ_{new}, respectively;
(2) Construct the DFA $\mathcal{A}_{(org,h)}$ as the DFA progression of $\mathcal{A}_{org}$ through h;
(3) Construct the DFA game $\mathcal{G} = \mathcal{A}_{(org,h)} \times \mathcal{A}_{new}$. Say ι is the initial state of $\mathcal{G}$;
(4) Solve $\mathcal{G}$. Let W be the winning region and κ a winning game strategy, if any: if $\iota \notin W$, return that incremental synthesis is *unrealizable*; else, return the strategy $\sigma_{(\mathcal{G},\kappa)}$ as the transducer $(\mathcal{G}, \kappa)$.

We show below the correctness of Algorithm 1. Recall that we are interested in constructing a game that admits a winning strategy iff incremental synthesis is realizable.

The DFA game is obtained as $\mathcal{G} = \mathcal{A}_{(org,h)} \times \mathcal{A}_{new}$; by Definition 3, the traces that $\mathcal{G}$ accepts are those accepted by both $\mathcal{A}_{(org,h)}$ and $\mathcal{A}_{new}$. As a result, $\mathcal{G}$ accepts the traces τ such that: (P1) $\tau \models \varphi_{new}$, by Theorem 3; (P2) $h \cdot \tau \models \varphi_{org}$, by Theorem 3 and Definition 4.

Next, we show that $\mathcal{G}$ admits a winning strategy iff incremental synthesis is realizable. Assume that $\mathcal{G}$ admits a winning strategy $\sigma_{(\mathcal{G},\kappa)}$: by (P1) and (P2), we have that $\sigma_{(\mathcal{G},\kappa)}$ generates only plays τ such that τ has a finite prefix that satisfies φ_{new} and $\tau' = h \cdot \tau$ has a finite prefix that satisfies φ_{org} – i.e., $\sigma_{(\mathcal{G},\kappa)}$ satisfies Definition 2 and incremental synthesis is realizable. Conversely, assume that $\mathcal{G}$ does not admit a winning strategy: by (P1) and (P2), there is no strategy that satisfies Definition 2 – i.e., incremental synthesis is unrealizable. As a result, we proved that Algorithm 1 is sound and complete, which is formally established in the following:

Theorem 4 *Incremental synthesis is realizable iff Algorithm 1 returns a strategy.*

Regarding complexity: Algorithm 1 is polynomial in the length of h, as it updates the initial state of $\mathcal{A}_{org}$ by executing its transition function through all of h. Wrt φ_{org} and φ_{new}: Algorithm 1 solves a game that is worst-case doubly-exponential in their sizes; solving this game can be done in polynomial time – so that Algorithm 1 establishes membership of incremental synthesis in 2EXPTIME wrt φ_{org} and φ_{new}. Having established the hardness of incremental synthesis in Theorem 1, we obtained the computational complexity of incremental synthesis:

Theorem 5 *Incremental synthesis for* φ_{org}*,* h*, and* φ_{new} *is:*
- 2EXPTIME*-complete wrt* φ_{org} *and* φ_{new}*;*
- *Polynomial wrt* h*.*

Finally, we show Algorithm 1 can be extended to an arbitrary number of goals. Let φ_{org} be the original goal, $h = h_1 \cdot \ldots \cdot h_n$ a history, and φ_i is a new goal that arrives at the end of $h_1 \cdot \ldots \cdot h_i$ for every $i \leq n$. The key modifications to Algorithm 1 are: (1) construct the DFAs $\mathcal{A}_{org}$ that accepts the traces that satisfy φ_{org}, and $\mathcal{A}_i$ that accepts the traces that satisfy φ_i for every $i \leq n$; (2) compute the DFA progressions $\mathcal{A}_{(org,h_1 \cdot \ldots \cdot h_n)}$, and $\mathcal{A}_{(i,h_{i+1} \cdot \ldots \cdot h_n)}$ for every $i < n$ (recall the DFA for the last new goal φ_n is not progressed); (3) construct and solve DFA game $\mathcal{G} = \mathcal{A}_{(org,h_1 \cdot \ldots \cdot h_n)} \times \mathcal{A}_{(1,h_2 \cdot \ldots \cdot h_n)} \times \cdots \times \mathcal{A}_{(n-1,h_n)} \times \mathcal{A}_n$. The arguments for soundness and completeness above can be extended by induction to the case where multiple goals are considered; the complexity wrt goals and histories remains unchanged.

Obviously, an agent that uses incremental synthesis does not need to compute from scratch the DFAs of the original goals every time it is requested to add a new goal – an operation that would cost 2EXPTIME for every goal. Instead, the agent can store in its memory the DFAs of the original goal and progress their initial states at each time step during execution; when the agent is requested to add a new goal, it only needs to compute the DFA for the new goal. Recall: *the bottleneck of* LTL$_f$ *synthesis is the* DFA *construction* (Zhu et al. 2017; Bansal et al. 2020; De Giacomo and Favorito 2021; De Giacomo et al. 2022); hence, storing the DFA of the goals in memory has the potential to improve significantly the performance of incremental synthesis.

These features make Algorithm 1 well-suited for efficient implementation, as confirmed empirically in Section 7.

5 Formula Progression

In this section, we show that incremental synthesis can be solved by LTL$_f$ *formula progression* (aka LTL$_f$ progression) (De Giacomo et al. 2022; Gabbay et al. 1980; Manna 1982; Emerson 1990; Bacchus and Kabanza 2000). Intuitively, the progression of an LTL$_f$ formula is a syntactic rewriting that captures, given a history, what remains to be satisfied of that formula after the history has occurred.

We start by reviewing LTL_f progression. First: LTL_f is interpreted over finite traces; it is necessary to clarify when traces end. To do so: we introduce two formulas $\Box(false)$ and $\Diamond(true)$, which, intuitively, denote *finite trace ends* and *finite trace does not end*, respectively. Let φ be an LTL_f formula in NNF over a set of atoms P and $w \in 2^P$ a propositional interpretation. The progression of φ through w, denoted $prog(\varphi, w)$, is (De Giacomo et al. 2022):

$$
\begin{aligned}
&prog(p, w) \doteq true \text{ if } p \in w \text{ and } false \text{ otherwise}\\
&prog(\neg p, w) \doteq true \text{ if } p \notin w \text{ and } false \text{ otherwise}\\
&prog(\varphi_1 \vee \varphi_2, w) \doteq prog(\varphi_1, w) \vee prog(\varphi_2, w)\\
&prog(\varphi_1 \wedge \varphi_2, w) \doteq prog(\varphi_1, w) \wedge prog(\varphi_2, w)\\
&prog(\circ\varphi, w) \doteq \varphi \wedge \Diamond(true)\\
&prog(\bullet\varphi, w) \doteq \varphi \vee \Box(false)\\
&prog(\varphi_1 \,\mathcal{U}\, \varphi_2, w) \doteq\\
&\quad prog(\varphi_2, w) \vee (prog(\varphi_1, w) \wedge prog(\circ(\varphi_1 \,\mathcal{U}\, \varphi_2), w))\\
&prog(\varphi_1 \,\mathcal{R}\, \varphi_2, w) \doteq\\
&\quad prog(\varphi_2, w) \wedge (prog(\varphi_1, w) \vee prog(\bullet(\varphi_1 \,\mathcal{R}\, \varphi_2), w))
\end{aligned}
$$

Let $h = \alpha_0 \cdots \alpha_n \in (2^P)^*$ be a history. We extend the LTL_f progression of φ to h, denoted $prog(\varphi, h)$, as follows: (*i*) $prog(\varphi, \epsilon) = \varphi$; and (*ii*) $prog(\varphi, \alpha_0 \cdots \alpha_n) = prog(prog(\varphi, \alpha_0 \cdots \alpha_{n-1}), \alpha_n)$. Intuitively, the LTL_f progression of φ through h captures what remains to be satisfied of φ after h has occurred.

Consider an instance of incremental synthesis: φ_{org} is the original LTL_f goal; h is the history; and φ_{new} is the new LTL_f goal. Using LTL_f progression, we can solve incremental synthesis by standard synthesis:

Theorem 6 *A strategy σ solves incremental synthesis for φ_{org}, h, and φ_{new} iff σ solves standard synthesis for $prog(\varphi_{org}, h) \wedge \varphi_{new}$.*

Next, we investigate the complexity of solving incremental synthesis by LTL_f progression. We observe that progressing φ_{org} through h generates an LTL_f formula $prog(\varphi_{org}, h)$ that is worst-case exponential in the size of φ_{org} (De Giacomo et al. 2022); this observation, together with the complexity of LTL_f synthesis, may hint that the complexity of solving incremental synthesis by LTL_f progression is 3EXPTIME wrt φ_{org} and 2EXPTIME wrt φ_{new}. However, *the bound wrt φ_{org} is too pessimistic*. In fact, we show below that the size of the DFA of any progressed formula is indeed bounded by that of the DFA of the original formula – so that also solving incremental synthesis by LTL_f progression establishes 2EXPTIME membership wrt φ_{org}.

To show this, we analyze the relation between LTL_f progression and DFA progression, see Definition 4. Let $\mathcal{A}_\varphi$ be the DFA for an LTL_f goal φ; say that ι and δ are initial state and transition function of $\mathcal{A}_\varphi$, respectively. If we progress φ one step through w, this corresponds to performing that same step in $\mathcal{A}_\varphi$ as well: the corresponding DFA is $\mathcal{A}_\varphi$, but with its initial state progressed to $\delta(\iota, w)$. This result can be extended by induction to histories of arbitrary length: if we progress φ through h, its corresponding DFA is $\mathcal{A}_{(\varphi,h)}$ – i.e., the same as $\mathcal{A}_\varphi$, but with its initial state progressed to $\delta(\iota, h)$, as in Definition 4. Hence, we obtain:

Theorem 7 *Let: φ be an LTL_f goal; $\mathcal{A}_\varphi$ its corresponding DFA; and h an history. The DFA progression $\mathcal{A}_{(\varphi,h)}$ of $\mathcal{A}_\varphi$ through h accepts exactly the traces that satisfy $prog(\varphi, h)$.*

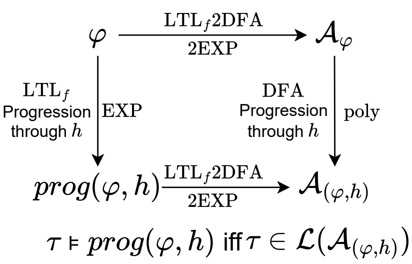

Figure 1: Relation between LTL_f progression and DFA progression through a history h

Theorem 7 shows that while $prog(\varphi, h)$ is worst-case exponential in the size of φ, its corresponding DFA can be obtained in 2EXPTIME – vs 3EXPTIME – as the DFA progression $\mathcal{A}_{(\varphi,h)}$ of $\mathcal{A}_\varphi$ through h. In other words: the worst-case exponential blowup introduced by the LTL_f progression $prog(\varphi, h)$ is not reflected in its corresponding minimal DFA, which is essentially the same as $\mathcal{A}_\varphi$. This result is summarized in Figure 1.

In fact, progressing the initial state of $\mathcal{A}_\varphi$ to get $\mathcal{A}_{(\varphi,h)}$ may cause parts of $\mathcal{A}_\varphi$ to be unreachable, which gives opportunities for further minimization. As a result:

Theorem 8 *Let $\mathcal{A}_\varphi$ be the DFA for an LTL_f formula φ and h a history. We have that $|\mathcal{A}_{(\varphi,h)}| \leq |\mathcal{A}_\varphi|$.*

Finally, we show that Algorithm 1 can be seen as a clever implementation of incremental synthesis by LTL_f progression. Indeed, this follows by Theorems 1, 6, and 7, as well as Definitions 3 and 4:

Corollary 1 *Consider an instance of incremental synthesis: φ_{org} is the original LTL_f goal; h the history; φ_{new} the new LTL_f goal. The DFA $\mathcal{G} = \mathcal{A}_{(org,h)} \times \mathcal{A}_{new}$ constructed (in polynomial time) in Step (3) of Algorithm 1 accepts exactly the traces that satisfy $prog(\varphi_{org}, h) \wedge \varphi_{new}$.*

6 Implementation

We implemented the automata-based technique (Algorithm 1) extended to multiple goals in a tool called ISABEL-DP[1] (***I****ncremental* ***S****ynthesis by* **D**FA ***P****rogression*). ISABEL-DP employs the symbolic synthesis framework in (Zhu et al. 2017) – which is integrated into state-of-the-art LTL_f synthesis tools (Zhu and Favorito 2025; Duret-Lutz et al. 2025). We use LYDIA (De Giacomo and Favorito 2021), which is among the best-performing LTL_f-to-DFA conversion tools, to generate DFAs from LTL_f formulas; in turn, LYDIA uses MONA (Henriksen et al. 1995) to represent DFAs and perform relevant DFA manipulations, such as, e.g., product and minimization. MONA uses a semi-symbolic representation of DFAs: states are represented explicitly; transitions are represented symbolically. We use SYFT to represent DFAs fully symbolically and solve symbolic DFA games (Zhu et al. 2017). SYFT represents state space and transition function of symbolic DFAs using Binary Decision Diagrams (Bryant

[1]https://github.com/GianmarcoDIAG/ISabel/

1992), with CUDD-3.0.0 (Somenzi 1998) as the underlying BDD library.

Our implementation employs both MONA's and SYFT's representations of DFAs in order to gain the maximal benefit during incremental synthesis. Specifically: (1) The MONA DFA representation is employed during game construction and DFA progression, so as to gain the maximal benefit from DFA minimization; (2) The SYFT DFA representation is employed during game solving, so as to gain the maximal benefit from symbolic game-resolution techniques.

Following the approach outlined in Section 4, ISABEL-DP maintains a data structure that stores the MONA DFAs of the added goals in a set V; their initial states are progressed incrementally during execution. When a new goal φ_{new} arrives, the implementation retrieves the DFAs of the previously added goals from V; doing so enables us to minimize the number of LTL_f-to-DFA conversions – thus saving 2EXPTIME for each stored goal. Specifically, the implementation takes the following steps: (1) transform φ_{new} into the MONA DFA $\mathcal{A}_{new}$; (2) construct the minimal MONA DFA game $\mathcal{G}$ as the product of the DFAs in V and $\mathcal{A}_{new}$; (3) transform $\mathcal{G}$ into a SYFT symbolic DFA game $\mathcal{G}^s$; (4) solve $\mathcal{G}^s$ and return a new strategy iff incremental synthesis is realizable, in which case $\mathcal{A}_{new}$ is also added to V.

7 Empirical Analysis

In this section, we present an empirical analysis for incremental synthesis. The goal of the analysis is to establish whether incremental synthesis is feasible in settings where standard synthesis is. Moreover, we want to determine whether the automata-based solution is more effective than the solution based on formula progression, when implemented directly.

To do so, we compare the performance of ISABEL-DP to that of a *baseline* implementation that solves incremental synthesis by LTL_f progression on some scalable benchmarks. The latter implementation is called ISABEL-FP (***I****ncremental* ***S****ynthesis by* LTL_f ***F****ormula* ***P****rogression*) and, for a fair comparison, it also uses LYDIA for LTL_f-to-DFA conversion and SYFT for symbolic game-resolution. We will show that ISABEL-DP *performs consistently and significantly better than* ISABEL-FP – *thus showing the effectiveness of the automata-based approach*.

Experimental Methodology. We considered four *benchmarks*: two of our own invention, PLANTS and REQUESTS; two adapted from existing benchmarks used in LTL_f synthesis and strong Fully Observable Nondeterministic (FOND) planning, COUNTER and TIREWORLD.[2] These benchmarks have been selected as they enable the agent to add many goals during execution – which makes them suited for evaluating incremental synthesis implementations. In each benchmark, an *instance* consists of the following: the implementation is given up to 3600 seconds until it times out; we evaluate the number of goals added by both ISABEL implementations before the timeout, as well as the time required to add each goal. To reproduce the incremental setting, the addition of new goals is interleaved with the execution of the strategy synthesized for the previous goals: at each time step, the agent performs an action and adds a new goal. All experiments were run on a laptop with an operating system 64-bits Ubuntu 22.04, 3.6 GHz CPU, and 8 GBs memory.

Benchmarks. We give a description for each benchmark. Recall from the synthesis settings in Sections 2 and 3: atoms are divided into agent atoms $\mathcal{Y}$ and environment atoms $\mathcal{X}$.

In our TIREWORLD benchmarks (Muise, McIlraith, and Beck 2012), the agent must navigate between several locations dealing with nondeterministic outcomes while moving. Specifically: from location ℓ the agent can move to any location ℓ'; as the agent moves, one its tires can nondeterministically become flat (in ℓ'); when a tire is flat, the agent cannot move, but can change the flat tire (we assume that spare tires are always available). Formally, TIREWORLD benchmarks are modeled as follows. The agent atoms are: (*i*) at_i, denotes that the current location of the agent is i; (*ii*) $move_i$, denotes that the agent moves to location i; and (*iii*) $change\text{-}tire$, denotes that the agent changes a flat tire. There is only one environment atom: (*iv*) $make\text{-}flat$, denotes that the environment caused one of the agent tires to become flat. Instances in TIREWORLD benchmarks are parametrized by the number of locations ℓ. We assume that the environment can make a tire flat only when the agent moves (*not* when the agent changes a flat tire), written in LTL_f $\alpha \doteq \Box(make\text{-}flat \supset \bigvee_{i<\ell} move_i)$. In each instance, we generate goals φ_n, with $n \leq 30$, as follows: if the assumption α holds, the agent must traverse locations from 0 to $\ell-1$ in sequence, for a total of $2n+1$ visits, written in LTL_f $\varphi_n \doteq$

$$\alpha \supset (\Diamond(at_0 \wedge \circ(\Diamond at_1 \wedge \cdots \wedge \circ(\Diamond at_{\ell-1} \wedge \circ(\Diamond at_0 \wedge \cdots \wedge \circ(\Diamond at_{2n+1 \ mod \ \ell}) \cdots)))))$$

where n is the goal parameter. The original goal ($n = 0$) also requires the agent to satisfy the specification of the domain in which it operates – which in this case is a FOND planning domain where nondeterminism is captured by the environment atom $make\text{-}flat$. The domain specification includes **(C1)** *Initial state*: the agent is at location ℓ_0, written:

$$\varphi_{init} \doteq at_0 \wedge \bigwedge_{i.0<i<\ell} \neg at_i$$

(C2) *Action preconditions*: at each time step, the agent can move only if it does not have a flat tire, and can change a tire only if it has a flat tire, written $\varphi_{pre} \doteq$

$$\Box(\bigwedge_{i<\ell}(move_i \supset \neg flat\text{-}tire)) \wedge \Box(change\text{-}tire \supset flat\text{-}tire)$$

(C3) *Mutual exclusion axiom*: at each time step, the agent must perform exactly one action, written

[2]In many AI applications – including planning – the agent has a model of the world in which it operates, e.g., a FOND domain.World models are typically considered specifications of the environment and can often be expressed in LTL_f, e.g., for the case of a FOND domain, see (Aminof et al. 2019). To avoid the introduction of explicit environment specifications, we exploit the following: once the agent action has been performed and the environment reaction observed, e.g., the effect of a PDDL `one-of` condition has occurred, the fluents describing the successor domain state are *completely determined* (De Giacomo, Parretti, and Zhu 2023). It follows that we can assign fluents as agent atoms and allow the environment reaction to be completely free (possibly with some care in handling all environment reactions appropriately). As a result, we can represent the world model as part of the original agent goal φ_{org}. This is the approach we follow in TIREWORLD benchmarks.

$$\varphi_{mutex} \doteq \Box(\textit{change-tire} \lor \bigvee_{i<\ell} move_i) \land \Box(\textit{change-tire} \supset \neg \bigvee_{i<\ell} move_i) \land \Box(\bigwedge_{i<\ell} move_i \supset \neg \textit{change-tire} \land \neg \bigvee_{j\neq i} move_j)$$

(C4) *Transition function*: at each time step, the agent must follow the transition function of the domain, written

$$\varphi_{trans} \doteq \Box(\bigwedge_{i\leq\ell}\bigwedge_{j\neq i} at_i \land move_j \land \textit{make-flat} \supset \bullet(at_j \land \textit{flat-tire} \land \bigwedge_{k\neq j} \neg at_k)) \land \Box(\bigwedge_{i\leq\ell}\bigwedge_{j\neq i} at_i \land move_j \land \neg\textit{make-flat} \supset \bullet(at_j \land \neg\textit{flat-tire} \land \bigwedge_{k\neq j} \neg at_k)) \land \Box(\bigwedge_{i\leq\ell} at_i \land \textit{change-tire} \supset \bullet(at_i \land \neg\textit{flat-tire} \land \bigwedge_{j\neq i} \neg at_j))$$

As a result, the original goal is:

$$\varphi_{org} \doteq \varphi_{init} \land \varphi_{pre} \land \varphi_{mutex} \land \varphi_{trans} \land \varphi_0$$

For every goal, a winning strategy for the agent is the following: navigate locations 0 to $\ell-1$ in order to accomplish a total of $2n+1$ visits; if the environment makes a tire flat, change the tire and resume navigation. We consider TIREWORLD instances with varying numbers of locations ℓ such that $1 \leq \ell \leq 10$, denoted TIREWORLD-ℓ; in each instance, we consider goals with $n \leq 30$, for a total of 300 goals.

We remark that there are more effective ways to generate the DFA of a FOND domain than converting its LTL$_f$ specification into DFA. For instance, (De Giacomo, Di Stasio, and Parretti 2025) show a FOND-to-DFA conversion technique that does not incur in the 2EXPTIME blowup resulting from the LTL$_f$-to-DFA conversion, but only costs EXPTIME in the size of the domain. Nonetheless, we represent the domain specification as an LTL$_f$ formula in order to be consistent with the synthesis settings introduced in Sections 2 and 3.

In COUNTER benchmarks (Zhu et al. 2020), the specification is as follows: the agent maintains a k-bits counter, with all bits initially set to 0; at each time step, the environment chooses whether to issue an increment request for the counter, captured by the environment atom add; the agent chooses whether to accept the request, and then increment the counter. Instances in COUNTER benchmarks are parametrized by the number of bits k in the counter. In each instance, we generate goals φ_n as follows: if the environment always issues increment requests, the agent must eventually set the counter value to $2n+1$, where n is the goal parameter. One such goal is written in LTL$_f$ as $\varphi_n \doteq (\Box add) \supset \Diamond\varphi^{counter}_{2n+1}$, where $\varphi^{counter}_{2n+1}$ is a conjunction of agent atoms that sets the counter value to $2n+1$. The original goal also requires the agent to satisfy the counter specification. For every goal, a winning strategy is to accept all increment requests and increment the counter accordingly. We consider COUNTER instances with number of bits k s.t. $6 \leq k \leq 10$, denoted COUNTER-k; in each instance, we consider goals with $n \leq 30$, for a total of 150 goals.

In PLANTS benchmarks, the specification is as follows: the agent has to take care of several plants; the agent can water the plants, captured by the agent atom $water$; that a plant i is alive is captured by the environment atom $alive_i$; the environment can also rain, captured by the environment atom $rain$. Instances in PLANTS benchmarks are parametrized by the number of plants p. At every time step. a plant i will be alive iff i was alive in the previous time step and either the agent waters it or the environment rains, written:

$$\alpha_i \doteq \Box(\bullet(alive_i) \equiv alive_i \land (water \lor rain))$$

In each instance, we generate goals as follows: if the assumption α_i holds, the agent must take care of plant i for $3(n+1)$ days, each day being a distinct time step, where n is the goal parameter. One such a goal is written

$$\varphi_n \doteq \bigwedge_{i<p} \alpha_i \supset \varphi^{alive_i}_{3(n+1)} \text{ with } \varphi^{alive_i}_{3(n+1)} \doteq \Diamond(\bigwedge_{j<3(n+1)} \circ^j alive_i)$$

where $\circ^k$ denotes k nested $\circ$ operators. For every goal, a winning strategy is to always water the plants. We consider PLANTS instances with number of plants p s.t. $1 \leq p \leq 10$, denoted PLANTS-p; in each instance, we consider goals with $n \leq 30$, for a total of 300 goals.

In REQUESTS benchmarks, the specification is as follows: the agent provides a set of services that the environment, at each time step, can request; to satisfy a request, the agent must perform a specific sequence of actions; the environment issues a finite number of consecutive requests, and the agent must satisfy all of them. Instances in REQUESTS benchmarks are parametrized by: (1) the number i of services s_i the agent provides; and (2) the number j of actions $a_{i,j}$ that the agent must perform to provide service i in response to a request for it. In each instance, we generate goals as follows: the agent must satisfy $2n+1$ environment requests, where n is the goal parameter. For every goal, a winning strategy is to perform all actions to satisfy the requests issued by the environment. We consider REQUESTS instances with: (*i*) number of services i s.t. $1 \leq i \leq 3$; (*ii*) number of actions j s.t. $1 \leq j \leq 4$; such instances are denoted REQUESTS-i-j; in each instance, we consider goals with $n \leq 40$, for a total of 480 goals.

Empirical Results. Table 1 compares the number of goals added by ISABEL-DP and ISABEL-FP, as well as their average runtime to add a new goal; for a fair comparison, in each instance, the average times are computed wrt the goals that are added by both. First, we note that both ISABEL-DP and ISABEL-FP failed to add any goal in COUNTER-9 and COUNTER-10: this is because both timed out during the construction of the DFA of the counter specification – whose size grows exponentially in the number of bits. We have an analogous result in REQUESTS-3-4: both ISABEL-DP and ISABEL-FP timed out during the construction of the DFA of the 7-th goal. These results confirm that DFA construction is the bottleneck of incremental synthesis, as in standard synthesis.

In the remaining instances, we have the following: (*i*) in terms of number of added goals, ISABEL-DP performs consistently better or the same as ISABEL-FP; (*ii*) in terms of average time to add a new goal, ISABEL-DP is significantly better than ISABEL-FP – *up to gaining one or two orders of magnitudes better performance in average*. This is evident in PLANTS and REQUESTS: in their instances, the size of the progressed LTL$_f$ goals may grow exponentially. To add a new goal, ISABEL-FP employs an LTL$_f$-to-DFA conversion for the LTL$_f$ formula obtained as the conjunction of the progressed LTL$_f$ goals – which may then cause an unnecessary overhead; on the other hand, ISABEL-DP employs an LTL$_f$-to-DFA conversion for the new LTL$_f$ goal only, and generates the DFAs of the progressed LTL$_f$ goals in polynomial time using DFA progression – which prevents any unnecessary overhead to occur.

We have analogous results in COUNTER instances. How-

Instance	Added Goals		Avg. Runtime (s)	
	ISABEL-DP	ISABEL-FP	ISABEL-DP	ISABEL-FP
PLANTS-1	**30**/30	19/30	**9.12**	155.12
PLANTS-2	**25**/30	16/30	**15.68**	172.00
PLANTS-3	**22**/30	14/30	**18.70**	179.43
PLANTS-4	**20**/30	13/30	**19.60**	173.17
PLANTS-5	**19**/30	12/30	**29.94**	265.45
PLANTS-6	**18**/30	12/30	**24.85**	216.06
PLANTS-7	**17**/30	11/30	**35.37**	268.26
PLANTS-8	**16**/30	11/30	**35.37**	268.26
PLANTS-9	**15**/30	10/30	**32.49**	190.29
PLANTS-10	**15**/30	10/30	**42.09**	237.39
COUNTER-6	**25**/30	20/30	**25.02**	29.00
COUNTER-7	**21**/30	12/30	**3.98**	13.18
COUNTER-8	**19**/30	10/30	**14.57**	41.11
COUNTER-9	**0**/30	**0**/30	-	-
COUNTER-10	**0**/30	**0**/30	-	-
REQUESTS-1-1	**36**/40	28/40	**27.53**	107.01
REQUESTS-1-2	**36**/40	29/40	**31.91**	117.05
REQUESTS-1-3	**36**/40	29/40	**31.41**	115.76
REQUESTS-1-4	**36**/40	29/40	**31.80**	111.20
REQUESTS-2-1	**37**/40	29/40	**24.39**	112.03
REQUESTS-2-2	**37**/40	29/40	**23.83**	112.27
REQUESTS-2-3	**36**/40	29/40	**24.49**	105.88
REQUESTS-2-4	**36**/40	29/40	**27.97**	105.64
REQUESTS-3-1	**37**/40	29/40	**24.12**	113.95
REQUESTS-3-2	**36**/40	29/40	**24.85**	108.55
REQUESTS-3-3	**35**/40	29/40	**33.81**	111.29
REQUESTS-3-4	**6**/40	**6**/40	7.09	**6.43**
TIREWORLD-1	**30**/30	26/30	**5.52**	118.14
TIREWORLD-2	**30**/30	26/30	**5.42**	121.49
TIREWORLD-3	**30**/30	25/30	**5.06**	141.56
TIREWORLD-4	**30**/30	24/30	**9.73**	117.57
TIREWORLD-5	**28**/30	21/30	**17.89**	152.42
TIREWORLD-6	**21**/30	19/30	**32.04**	151.25
TIREWORLD-7	**17**/30	**17**/30	**41.77**	157.20
TIREWORLD-8	**15**/30	**15**/30	**34.05**	98.96
TIREWORLD-9	**13**/30	**13**/30	**18.95**	47.23
TIREWORLD-10	**12**/30	**12**/30	**15.97**	36.65
Total	**892**/1230	692/1230		

Table 1: Comparison of the number of goals added and average time to add a goal achieved by ISABEL-DP and ISABEL-FP. Results of the best-performing implementation in bold.

ever, it is important to note the following: *in* COUNTER *instances, the size of the progressed* LTL_f *goals is almost the same as that of the non-progressed goals*. Nonetheless, ISABEL-DP outperformed ISABEL-FP both in terms of added goals and average time to add a goal. The reason is as follows: ISABEL-DP benefits from retrieving the DFAs of the goals from memory, whereas ISABEL-FP incurs an unnecessary overhead by employing an LTL_f-to-DFA conversion for the LTL_f formula obtained as the conjunction of the progressed LTL_f goals. Hence, storing the DFAs of the goals in memory improves the performance of incremental synthesis.

Regarding TIREWORLD benchmarks, we have the following. (1) Instances TIREWORLD-1 to TIREWORLD-6 show that ISABEL-DP performs better than ISABEL-FP both in terms of added goals and average time to add a goal – *up to gaining one or two orders of magnitude better performance in average* – since ISABEL-DP avoids the unnecessary overhead resulting by using LTL_f progression. (2) Instance TIREWORLD-7 to TIREWORLD-10 show that ISABEL-DP and ISABEL-FP add the same number of goals, and, while ISABEL-DP remains much faster than ISABEL-DP, the performance gap between ISABEL-DP and ISABEL-FP becomes smaller. This latter result is due to memory limitation (8 GBs of RAM) of the machine on which the experiments are run. Indeed, as we approach the memory limit, memory management overhead becomes significant for both ISABEL-DP and ISABEL-FP, until *they both run out of memory during the construction of the* DFA *game arena for computing the new strategy*. Recall that ISABEL-DP and ISABEL-FP construct the same DFA game arena (see Theorem 6 and Corollary 1): hence, the observed performance difference stems from the more efficient DFA game arena construction employed by ISABEL-DP – which, in turn, suggests ISABEL-DP would scale better than ISABEL-FP in large TIREWORLD instances when run on more powerful hardware.

Overall, the empirical analysis shows the superiority of ISABEL-DP to ISABEL-FP – which confirms the effectiveness of our approach to incremental synthesis and its feasibility in settings where also standard synthesis is feasible.

8 Conclusion

In addition to being of theoretical and general interest, this work also has a direct practical impact, providing an effective approach for revising an agent's strategy when goals dynamically change. Our results provide an efficient incremental technique for revising strategies for autonomous agents that operate in nondeterministic domains, reducing as much as possible the synthesis effort required when goals change dynamically. In this context, the proposed approach can be directly integrated into a Goal/Intention Management System, such as the one sketched in (De Giacomo et al. 2025).

Observe that a new goal might be inconsistent with the current ones, leading to a situation where no strategy exists that guarantees the realization of all the agent's goals. Although the present paper does not address this issue, our technique allows for effective conflict detection, a first step toward resolution. Then, if all the agent's goals have the same importance, one can look into finding a *maximal realizable subset*, though this is an exponential problem itself, since it may require trying exponentially many combinations, each of which is an incremental synthesis problem. A simple alternative is to assume that the adopted goals are totally ordered with respect to priority, and given this, keep the set of highest priority goals that are jointly realizable.

In future work, we plan to investigate forward-search approaches for incremental synthesis based on those adopted in LTL_f synthesis and planning (Camacho and McIlraith 2019; Bonassi et al. 2023; Muise, McIlraith, and Beck 2024; Duret-Lutz et al. 2025). Incremental synthesis could also be extended in further directions, e.g., synthesizing best-effort/good-enough strategies when there is no winning strategy (Aminof, De Giacomo, and Rubin 2021; De Giacomo, Parretti, and Zhu 2023; Almagor and Kupferman 2020; Aminof et al. 2024) or the partial observability setting (Tabajara and Vardi 2020).

Acknowledgments

This work has been supported by the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI (No. EP/Y028872/1), the Italian National Ph.D. in Artificial Intelligence at Sapienza University of Rome, the Natural Sciences and Engineering Research Council of Canada, and York University.

References

Almagor, S.; and Kupferman, O. 2020. Good-Enough Synthesis. In *CAV*.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL Environment Specifications. In *ICAPS*.

Aminof, B.; De Giacomo, G.; Parretti, G.; and Rubin, S. 2024. Effective Approach to LTL_f Best-Effort Synthesis in Multi-Tier Environments. In *IJCAI*.

Aminof, B.; De Giacomo, G.; and Rubin, S. 2021. Best-Effort Synthesis: Doing Your Best Is Not Harder Than Giving Up. In *IJCAI*.

Apt, K. R.; and Grädel, E., eds. 2011. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press.

Bacchus, F.; and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *AIJ*.

Baier, C.; and Katoen, J. 2008. *Principles of model checking*. MIT Press.

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

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

Bonassi, L.; De Giacomo, G.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2023. FOND Planning for Pure-Past Linear Temporal Logic Goals. In *ECAI*.

Bonassi, L.; Gerevini, A. E.; Percassi, F.; and Scala, E. 2021. On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away. In *ICAPS*.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2022. Planning with Qualitative Action-Trajectory Constraints in PDDL. In *IJCAI*, 4606–4613.

Bryant, R. E. 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*

Camacho, A.; and McIlraith, S. A. 2019. Strong Fully Observable Non-Deterministic Planning with LTL and LTL_f Goals. In *IJCAI*.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *AAAI*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *AIJ*.

De Giacomo, G.; Di Stasio, A.; and Parretti, G. 2025. PDDL to DFA: A Symbolic Transformation for Effective Reasoning. In *TIME*.

De Giacomo, G.; and Favorito, M. 2021. Compositional Approach to Translate LTL_f/LDL_f into Deterministic Finite Automata. In *ICAPS*.

De Giacomo, G.; Favorito, M.; Li, J.; Vardi, M. Y.; Xiao, S.; and Zhu, S. 2022. LTL_f Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In *IJCAI*.

De Giacomo, G.; Lespérance, Y.; Parretti, G.; Patrizi, F.; and Schram, R. 2025. Managing an Agent's Changing Intentions Using LTLf Synthesis. In *AAMAS*.

De Giacomo, G.; Parretti, G.; and Zhu, S. 2023. LTL_f Best-Effort Synthesis in Nondeterministic Planning Domains. In *ECAI*.

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTL_f and 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*, 854–860.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *IJCAI*, 1558–1564.

Duret-Lutz, A.; Zhu, S.; Piterman, N.; De Giacomo, G.; and Vardi, M. Y. 2025. Engineering an LTL_f Synthesis Tool. In *CIAA*.

Emerson, E. A. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Chapter 16*.

Finkbeiner, B.; Klein, F.; and Metzger, N. 2022. Live synthesis. *Innov. Syst. Softw. Eng.*

Gabbay, D. M.; Pnueli, A.; Shelah, S.; and Stavi, J. 1980. On the Temporal Analysis of Fairness. In *POPL*.

Henriksen, J. G.; Jensen, J. L.; Jørgensen, M. E.; Klarlund, N.; Paige, R.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic Second-Order Logic in Practice. In *TACAS*.

Kankariya, Y.; and Bansal, S. 2024. Decompositions in Compositional Translation of LTLf to DFA. In *AAAI*.

Manna, Z. 1982. *Verification of Sequential Programs: Temporal Axiomatization*, 53–102. Springer.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2024. PRP Rebooted: Advancing the State of the Art in FOND Planning. In *AAAI*.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *ICAPS*.

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

Pnueli, A.; and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *POPL*.

Sipser, M. 1997. *Introduction to the theory of computation*. PWS Publishing Company.

Somenzi, F. 1998. CUDD: CU decision diagram package release 2.3.0. In *University of Colorado at Boulder*.

Tabajara, L. M.; and Vardi, M. Y. 2020. LTL_f Synthesis under Partial Observability: From Theory to Practice. In *GandALF*.

Zhu, S.; and De Giacomo, G. 2022. Act for Your Duties but Maintain Your Rights. In *KR*.

Zhu, S.; De Giacomo, G.; Pu, G.; and Vardi, M. Y. 2020. LTL$_f$ Synthesis with Fairness and Stability Assumptions. In *AAAI*.

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. 2017. Symbolic LTL$_f$ Synthesis. In *IJCAI*.