Synthesis Foundations for Online LTL$_f$ Goal Management

Giuseppe De Giacomo[1,2] , **Yves Lespérance**[3] , **Gianmarco Parretti**[2] , **Fabio Patrizi**[2]
[1]University of Oxford
[2]Sapienza University of Rome
[3]York University
giuseppe.degiacomo@cs.ox.ac.uk lesperan@eecs.yorku.ca {parretti, patrizi}@diag.uniroma1.it

Abstract

Autonomous agents' goals typically change as they operate. Handling this is particularly challenging when the environment is nondetermnistic and the goals are temporally extended. Here, we assume that the agent operates in a fully observable nondeterministic (FOND) domain and uses Linear Temporal Logic over finite traces (LTL$_f$) to represent goals. We use LTL$_f$ synthesis notions to formalize this problem of online agent goal management, handling goal adoption, goal dropping, and performing steps of the synthesized strategy, while ensuring that the agent's goals always remain realizable. We propose automata-based and formula progression-based methods to manage LTL$_f$ goals. We implement these methods and evaluate their effectiveness experimentally.

1 Introduction

Autonomous agents' goals generally change as they operate. They adopt new goals and drop existing ones as a result of interactions with others and changes in their own motivations. In this paper we develop a new approach to managing goal change and planning/strategic reasoning for agents that builds on *reactive synthesis for Linear Temporal Logic on finite traces (*LTL$_f$*)* (De Giacomo and Vardi 2015; Zhu et al. 2017). We propose an *agent goal management system (*AGMS*)* framework where we assume that the agent has a dynamically changing set of goals, which are specified by arbitrary LTL$_f$ formulas. We also assume that the agent operates in a fully observable nondeterministic (FOND) domain, where the agent does not fully control the outcomes of its actions. We use LTL$_f$ synthesis to check that all the agent's goals remain *realizable*, i.e., jointly achievable no matter how the environment behaves, and to generate strategies to achieve them. This means that the developer only has to specify the FOND domain and the goals that the agent should adopt and drop. It also ensures that the agent behaves with a high degree of rationality. Our approach to goal management can be viewed as an advanced form of planning in which the agent generates and executes a plan/strategy for satisfying its current goals, but is also ready to revise the set of goals and replan during execution.

The AGMS is assumed to be a core component of the agent's architecture that provides planning/strategic reasoning and goal management capabilities.[1] We use LTL$_f$ to specify agent's goals, a very expressive language that has proven useful for many applications. This allows complex constraints on the agent behavior to be specified, thus supporting framed autonomy and safety guarantees. This is a key feature of the framework. We also assume that the environment is fully observable, which enables exploiting FOND planning/synthesis techniques, and leave handling partial observability for future work. Finally, we assume that the agent wants to ensure it achieves its goals no matter how the environment behaves, i.e., that plans are robust.

We first present a formal specification of an AGMS based on LTL$_f$ synthesis notions. After this, we show how we can obtain a AGMS realization by exploiting progression-based and automata-based techniques. The AGMS needs to check realizability and recompute strategies when new goals are adopted. We show how this can be done more efficiently by exploiting synthesis auxiliary data structures, and present an implemented prototype tool. We evaluate our approach empirically and show that it can be used to effectively implement agents that operate in non-trivial domains and adopt numerous goals over the course of a run while also performing steps of the synthesized strategies.

2 Preliminaries

We consider specifications (aka *formulas*) in Linear Temporal Logic on *finite traces* (LTL$_f$) (De Giacomo and Vardi 2013) – a formalism that can be used to represent temporal properties over finite-time horizons. It is widely employed in many areas of Computer Science (CS) and Artificial Intelligence (AI), including: Planning, to specify temporally extended goals (De Giacomo and Rubin 2018; Camacho and McIlraith 2019; De Giacomo, Parretti, and Zhu 2023) as well as state/action trajectory constraints (Bonassi et al. 2021; Bonassi, Gerevini, and Scala 2022); and Formal Methods, to specify safety dynamic properties in system verification (Baier and Katoen 2008).

[1]There would normally also be an *executor* component that actually performs the action prescribed by the strategy synthesised by the AGMS, observes the resulting environment state, and informs the AGMS of it so that the agent goals state can be updated accordingly. As well, there would typically be another *goal revision* component that decides when to adopt new goals and drop current goals, mostly as a result of interaction with other agents but possibly due to intrinsic motivation change.

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:

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

where $p \in P$. Boolean operators include: *negation* ($\neg$); *disjunction* ($\vee$); and *conjunction* ($\wedge$). Temporal operators include: *next* ($\bigcirc$); *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 $|\tau|$ the length of τ. Given 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, denoted $\tau, i \models \varphi$:

- $\tau, i \models p$ 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 $\tau, i \models \varphi_2$;
- $\tau, i \models \bigcirc\varphi$ 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 Synthesis in Fully Observable Nondeterministic Domains. Following (Geffner and Bonet 2013), a *Fully Observable Nondeterministic (*FOND*) Domain* is a tuple $\mathcal{D} = (2^{\mathcal{F}}, Act, \delta, \alpha)$, where $\mathcal{F}$ is a finite set of *fluents*, and $2^{\mathcal{F}}$ is the *state space*; Act is a finite set of actions; $\alpha : 2^{\mathcal{F}} \to 2^{Act}$ represents *action preconditions*; and $\delta \subseteq 2^{\mathcal{F}} \times Act \times 2^{\mathcal{F}}$ is the *transition relation* such that $\delta\,(s, a, s')$ only if $a \in \alpha(s)$. Considering the domain represented in PDDL (Haslum et al. 2019), we identify its size with $|\mathcal{F}|$.

Given an initial state $s_0 \in 2^{\mathcal{F}}$, the domain evolves as follows: at each time step, the agent performs an action a among those that satisfy actions preconditions in the current domain state s; the environment chooses a successor state s' among those s.t. $(s, a, s') \in \delta$.

Formally, a *domain trace* starting from s_0 over $\mathcal{D}$ is a (finite or infinite) sequence $\tau = (\texttt{start}, s_0)(a_1, s_1)\cdots$ where `start` is a dummy action and $(s_i, a_{i+1}, s_{i+1}) \in \delta$ for every $i \geq 0$. Infinite domain traces are also called *plays*; finite domain traces are also called *histories*.

An *(agent) strategy* is a function $\sigma : (2^{\mathcal{F}})^* \to Act$, mapping (possibly empty) sequences of domain states to agent actions. Given a strategy σ and a domain $\mathcal{D}$ with initial state s_0, a play $(\texttt{start}, s_0)(a_1, s_1)\cdots$ is σ-consistent if: (*i*) $\sigma(\epsilon) = \texttt{start}$; (*ii*) $a_{i+1} = \sigma(s_0 \cdots s_i) \in \alpha(s_i)$ for every $i \geq 0$ – i.e., the agent always follows action preconditions; and (*iii*) $(s_i, a_{i+1}, s_{i+1}) \in \delta$ for every $i \geq 0$ – i.e., the environment always follows the transition function of the domain. The set of σ-consistent plays in $\mathcal{D}$ starting from s_0 is $Play(\sigma, \mathcal{D}, s_0)$ and is always non-empty if we assume that the agent can perform at least one action in every state. [2]

A *goal* in $\mathcal{D}$ is an LTL_f formula φ over $\mathcal{F}$. Every domain trace $\tau = (\texttt{start}, s_0)(a_1, s_1)\cdots$ induces a trace $\tau|_{2^{\mathcal{F}}} = s_0 s_1 \cdots$, over which, when finite, we can evaluate LTL_f formulas. Given a history $h = (\texttt{start}, s_0)\cdots(a_n, s_n)$, we denote by $\texttt{lst}(h)$ its last state s_n.

LTL_f synthesis in FOND domains is the problem of computing a strategy, called *winning strategy* (or *strong plan*) that achieves the goal, irrespective of how the domain's nondeterminism is resolved. Formally, a strategy σ is *winning* for φ in $\mathcal{D}$ from s_0 if, for every play $\tau \in Play(\sigma, \mathcal{D}, s_0)$ there exists a finite prefix of $\tau|_{2^{\mathcal{F}}}$ that satisfies φ. If such a strategy exists, we say that φ is *realizable* in $\mathcal{D}$ from s_0. When φ, $\mathcal{D}$, and/or s_0 are clear from the context, we simply say that σ is winning or φ is realizable. LTL_f synthesis is the problem of finding a winning strategy for φ in $\mathcal{D}$ from s_0, if any. The problem is 2EXPTIME-complete wrt $|\varphi|$ and EXPTIME-complete wrt $\mathcal{D}$ (i.e. $|\mathcal{F}|$), respectively (De Giacomo and Rubin 2018).

3 Agent Goal Management System

In this paper, we provide the formal specification of an Agent Goal Management System (AGMS) based on LTL_f synthesis in FOND domains. Generally speaking, the purpose of the AGMS is to *provide autonomous agents with the capability to synthesize strategies and rationally manage and revise multiple prioritized goals while operating in a* FOND *domain*.

Consider an agent that has been assigned/adopted an LTL_f goal φ_1 and is executing a strategy σ_1 to realize it. Suppose that during strategy execution, which so far has generated a history h, the agent adopts/is assigned another LTL_f goal φ_2. It is easy to see that the agent cannot rely on σ_1, as it was computed before φ_2 was known. Yet, synthesizing from scratch a new strategy σ_2 for both φ_1 and φ_2 would not be an effective solution: σ_2 would restart satisfying φ_1 and φ_2 since the beginning, thus discarding the progress that σ_1 made for φ_1 during h. Instead, we require σ_2 to ensure the realization of φ_2 as well as the realization of φ_1 and for the latter taking into account that history h has occurred. However, suppose that the realizability of φ_1 and φ_2 is *conflicting*, i.e., it is not possible to realize both of them. Then, depending on the priority of φ_1 and φ_2, the agent has to make a choice: (*i*) continue realizing φ_1; or (*ii*) drop φ_1 and begin realizing φ_2. An AGMS has to provide

[2] If this assumption does not hold, we can add a no-operation action nop s.t. $(s, nop, s) \in \delta$ and $nop \in \alpha(s)$ for every $s \in 2^{\mathcal{F}}$.

autonomous agents with the capability of handling scenarios like those described above, i.e., execute strategies, adopt new goals taking into account strategy execution and existing goals as well as possible conflicts among goals, and drop goals if necessary.

Background Concepts and Notation. First, the AGMS needs to manage multiple LTL_f goals adopted at the end of distinct histories: we need to define what it means for a strategy σ to be winning for a goal φ assuming that a certain history h has occurred. Intuitively, this means that σ can extend h and generate only domain traces that satisfy φ. This intuition is formalized as follows:

Definition 1. *Let: $\mathcal{D}$ be a* FOND *domain; φ an* LTL_f *goal; h a history; and σ a strategy. We say that σ is winning for φ in $\mathcal{D}$* assuming *h if, for every play $\tau \in Play(\sigma, \mathcal{D}, \texttt{lst}(h))$, we have that $(h \cdot \tau)|_{2^{\mathcal{F}}}$ has a finite prefix that satisfies φ.*

We also need to refer to histories generated after a certain history has occurred. Let $h = (\texttt{start}, s_0) \cdots (a_n, s_n)$ be a *history* and $h^k = (\texttt{start}, s_0) \cdots (a_k, s_k)$ a *prefix* of h: we denote by $h \setminus h^k = (a_{k+1}, s_{k+1}) \cdots (a_n, s_n)$ the *suffix* of h containing the action-state pairs observed after h^k.

Finally, we use lists to represent ordered sets of objects. A list is denoted $L = [o_1, \cdots, o_n]$, where o_i is the i-th object in the list. By $L[i..j] = [o_i, \cdots, o_j]$ we denote the sublist containing the elements of L from the i-th to the j-th index, where $i \leq j \leq n$. Given two lists $L = [o_1, \cdots, o_n]$ and $L' = [o'_1, \cdots, o'_m]$, their concatenation is the list $L \circ L' = [o_1, \cdots, o_n, o'_1, \cdots, o'_m]$.

Intuitively, the AGMS is formalized as a state-based system that evolves as the agent operates. The AGMS state includes: the FOND domain the agent operates in (assumed fixed); the list of LTL_f goals the agent has adopted so far – i.e., the LTL_f goals that the agent must realize – together with the history in which each goal was adopted; a winning strategy for all the goals adopted so far, together with the history in which the strategy was selected; and the history generated since the beginning of the AGMS execution. Formally, an AGMS *state* is a tuple:

$$\mathcal{I}^{\text{AGMS}} = (\mathcal{D}, L, (\sigma, h_\sigma), h)$$

where: $\mathcal{D}$ is the FOND domain the agent operates in; $L = [(\varphi_1, h_1), \ldots, (\varphi_n, h_n)]$ is a list of pairs (φ_i, h_i), where φ_i is the currently adopted i-th LTL_f goal and h_i is the history in which φ_i was adopted; (σ, h_σ) is a pair consisting of a strategy σ and the history h_σ in which σ was selected; and h is the history generated since the beginning of the AGMS execution. Note that both h_σ and every h_i are prefixes of h. The list L is sorted in decreasing order of goal priority, i.e., φ_1 is the goal at the highest priority and φ_n the goal at the lowest. At each time step, we require the strategy σ to be *winning for* L – i.e., for every $(\varphi_i, h_i) \in L$, σ must be winning for φ_i assuming $h \setminus h_i$, so that σ realizes φ_i taking into account what has been observed since φ_i has been adopted. We say that L is *realizable* when a winning strategy for L exists. We assume that the AGMS state is initialized as $\mathcal{I}_0^{\text{AGMS}} = (\mathcal{D}, [\,], (\sigma, s_0), s_0)$, where s_0 is an initial domain state and σ is an arbitrary strategy (we assume that the empty list of goals $[\,]$ is always realizable).

We now give the specification for three operations that can modify the AGMS state. Informally, these operations are: *adoption*, to adopt a new goal, if possible; *dropping*, to drop a currently adopted goal; and *step*, to execute the next action prescribed by the currently selected strategy, observe the resulting domain state (nondeterministically chosen by the environment), and update the AGMS state accordingly.

Adopt. The first operator that we define is $adopt$. This is formally denoted as $adopt(\phi, k)$, where ϕ is an LTL_f goal and k is such that $1 \leq k \leq n+1$ and denotes the priority index at which ϕ should be adopted. Intuitively, a goal should be adopted at a certain priority index only if it is realizable jointly with all goals with higher priority. That is, there are two possible outcomes resulting from the execution of $adopt(\phi, k)$ in history h: (*i*) ϕ is realizable jointly with all goals with higher priority, i.e., $L[1..(k-1)] \circ [(\phi, h)]$ is realizable, in which case ϕ is adopted; or (*ii*) ϕ is not realizable jointly with all goals with higher priority, in which case it is not adopted and the AGMS state remains unchanged. Furthermore, in case (*i*) holds, the AGMS must drop the goals φ_i with priority lower than k (i.e., $i \geq k$) that became unrealizable after the adoption of ϕ. More specifically: the AGMS should keep the maximal set of highest priority goals that is realizable. For example, assume that the agent's currently adopted goals are $L = [(\varphi_1, h_1), (\varphi_2, h_2), (\varphi_3, h_3), (\varphi_4, h_4)]$ and we instruct $adopt(\varphi_{new}, 2)$ in h_{new}: if $\{(\varphi_1, h_1), (\varphi_{new}, h_{new})\}$ is realizable, $\{(\varphi_1, h_1), (\varphi_{new}, h_{new}), (\varphi_2, h_2)\}$ is also realizable, $\{(\varphi_1, h_1), (\varphi_{new}, h_{new}), (\varphi_2, h_2), (\varphi_3, h_3)\}$ is not realizable, and $\{(\varphi_1, h_1), (\varphi_{new}, h_{new}), (\varphi_2, h_2), (\varphi_4, h_4)\}$ is realizable, then (φ_3, h_3) must be dropped and the agent goals afterwards should be the latter, i.e., $L' = [(\varphi_1, h_1), (\varphi_{new}, h_{new}), (\varphi_2, h_2), (\varphi_4, h_4)]$.

Formally, the specification of $adopt(\phi, k)$ is as follows. Given the AGMS state $\mathcal{I}_m^{\text{AGMS}} = (\mathcal{D}, L, (\sigma, h_\sigma), h)$, return the successor state $\mathcal{I}_{m+1}^{\text{AGMS}} = \mathcal{I}_m^{\text{AGMS}}$ in case $L[1..(k-1)] \circ [(\phi, h)]$ is unrealizable, otherwise, return a succcessor AGMS state defined as follows:

$$\mathcal{I}_{m+1}^{\text{AGMS}} = (\mathcal{D}, R(L, (\phi, h), n), (\sigma', h), h)$$

where: $\mathcal{D}$ and h are unchanged wrt AGMS; σ' is a winning strategy for $R(L, (\phi, h), n)$; and $R(L, (\phi, h), n)$ is the list of goal-history pairs obtained by adding (ϕ, h) at priority index k and removing all goal-history pairs for lower priority goals that became unrealizable after ϕ has been adopted, which is defined recursively. Formally, $R(L, (\phi, h), n)$ is defined as follows: $R(L, (\phi, h), j) = [(\varphi_1, h_1), \cdots, (\varphi_{k-1}, h_{k-1}), (\phi, h)]$, if $j < k$; otherwise (2.1) $R(L, (\phi, h), j) = R(L, (\phi, h), j-1) \circ [(\varphi_j, h_j)]$ if the latter is realizable; otherwise (2.2) $R(L, \phi, h, j) = R(L, \phi, h, j-1)$.]. Intuitively, the definition of R can be split in two parts: the base case returns the list consisting of all goal-history pairs (φ_j, h_j) with priority greater than k and (φ, h), Case (1). The recursive part checks whether each goal φ_j with priority lower than k remains realizable after ϕ has been adopted: if yes (φ_j, h_j) is added to $R(L, (\phi, h), n)$, Case (2.1); otherwise, it is not added, Case (2.2). That is, $R(L, (\phi, h), n)$ contains the list of all goals that are realizable after ϕ has been adopted in h.

Drop. The second operator that we consider is $drop$. This is formally denoted as $drop(k)$, where k is such that $1 \leq$

$k \leq |L|$ and denotes the index of the k-th currently adopted goal φ_k. Intuitively, $drop(k)$ requires the AGMS to drop the k-th goal φ_k and to recompute a winning strategy for the remaining goals.

Formally, the specification of $drop(k)$ is as follows. Given the AGMS state $\mathcal{I}_m^{\text{AGMS}} = (\mathcal{D}, L, (\sigma, h_\sigma), h)$, return a successor AGMS state $\mathcal{I}_{m+1}^{\text{AGMS}} = (\mathcal{D}, R, (\sigma', h), h)$, where: $\mathcal{D}$ and h are unchanged wrt AGMS; $R = [L(1..k-1)] \circ L[k+1..n]$, i.e., the same as L, except that the goal-history pair (φ_k, h_k) at the k-th index has been removed; and σ' is a winning strategy for R. [3]

Step. The third operator that we define is *step*. This is formally denoted as $step(s')$, where $s' \in 2^{\mathcal{F}}$ is a domain state. Intuitively, *step* requests the AGMS to perform the next action prescribed by the currently selected strategy and update the AGMS state based on the observed successor domain state s' – nondeterministically chosen by the environment.

Its specification is as follows. Given the AGMS state $\mathcal{I}_m^{\text{AGMS}} = (\mathcal{D}, L, (\sigma, h_\sigma), h)$, return the successor AGMS state $\mathcal{I}_{m+1}^{\text{AGMS}} = (\mathcal{D}, L, (\sigma, h_\sigma), h \cdot (\sigma((h \setminus h_\sigma)|_{2^{\mathcal{F}}}), s')))$. That is: $\mathcal{I}_{m+1}^{\text{AGMS}}$ is the same as $\mathcal{I}_m^{\text{AGMS}}$, except that the history h is extended with the next action $(\sigma((h \setminus h_\sigma)|_{2^{\mathcal{F}}})$ prescribed by σ and the resulting successor domain state s', where $(\texttt{lst}(h), (\sigma((h \setminus h_\sigma)|_{2^{\mathcal{F}}}), s') \in \delta$.

AGMS Problem Specification. We conclude this section by giving a formal definition of the computational problem that the AGMS must solve. Intuitively, the AGMS problem is as follows: given the current AGMS state and the next update operation – i.e., either $adopt(\phi, k)$, $step(s)$, or $drop(k)$ – compute a successor AGMS state. Formally:

Definition 2 (AGMS Problem)**.** *Let $\mathcal{I}_m^{\text{AGMS}}$ be the current* AGMS *state and* op *the next update operation. Compute a successor* AGMS *state $\mathcal{I}_{m+1}^{\text{AGMS}}$ that is the result of performing operation* op *in* AGMS *state $\mathcal{I}_m^{\text{AGMS}}$.*

Obviously, an agent that manages multiple goals using an AGMS needs to perform many update problems during execution. However, Definition 2 suffices to characterize formally the computational problems that an AGMS must solve.

Indeed, the AGMS problem generalizes LTL$_f$ synthesis in FOND domains. Let $\mathcal{D}$ be a FOND domain with initial state s_0 and φ an LTL$_f$ goal. Consider the initial AGMS state $\mathcal{I}_0^{\text{AGMS}} = (\mathcal{D}, [\,], [(\sigma, s_0)], s_0)$ and the operation $\pi = adopt(\varphi, 1)$. By definition of *adopt*, the AGMS will adopt φ iff φ is realizable in $\mathcal{D}$ from s_0. Formally:

Theorem 3. *Let $\mathcal{D}$ be a* FOND *domain with initial state s_0 and φ an* LTL$_f$ *goal: φ is realizable in $\mathcal{D}$ from s_0 iff the* AGMS *state $\mathcal{I}_0^{\text{AGMS}} = (\mathcal{D}, [\,], (\sigma, s_0), s_0)$ after operation* $\texttt{op} = adopt(\varphi, 1)$ *has a successor* AGMS *state of the form $\mathcal{I}_1^{\text{AGMS}} = (\mathcal{D}, [(\varphi, s_0)], (\sigma_\varphi, s_0), s_0)$, where σ_φ is a winning strategy for φ in $\mathcal{D}$ from s_0.*

Given the complexity of LTL$_f$ synthesis in FOND domains (De Giacomo and Rubin 2018), we can establish the hardness of the AGMS problem:

[3] It is not strictly necessary to recompute the strategy, as the original agent strategy achieves all the remaining goals after dropping. However, there may be a better strategy, i.e., one that achieves all the remaining goals in fewer steps in the worst-case, which is why we recompute the strategy after dropping.

Theorem 4. *The* AGMS *problem is* EXPTIME*-hard wrt $\mathcal{D}$ and* 2EXPTIME*-hard wrt the adopted/dropped goals φ_i.*

4 Progression-Based Agent Goal Management System

In this section, we provide an alternative specification for an Agent Goal Management System that is completely *memoryless*: the states of the system do not need to store histories for neither adopted goals, the selected strategy, nor the history generated during execution. We base this alternative specification on 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) – hence, we call it *Progression-Based Agent Goal Management System* (PAGMS). The motivation for introducing the PAGMS is that it admits algorithms for adoption and dropping that simply reduce to standard LTL$_f$ synthesis in FOND domains. As a first step, we review LTL$_f$ progression.

Background on LTL$_f$ Progression. Intuitively, the LTL$_f$ progression of an LTL$_f$ formula φ is a syntactic rewriting of φ that captures what remains to be satisfied of φ after a history has occurred. This intuition is formalized below.

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 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):

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

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*) $prog(\varphi, \alpha_0 \cdots \alpha_n) = prog(prog(\varphi, \alpha_0 \cdots \alpha_{n-1}), \alpha_n)$.

Lemma 5. *Let φ be an* LTL$_f$ *formula and h a history. For every trace $\tau \in (2^{\mathcal{P}})^*$, we have that $h \cdot \tau \models \varphi$ iff $\tau \models prog(\varphi, h)$ (De Giacomo et al. 2022).*

We provide the specification of the PAGMS. Intuitively, a PAGMS state retains the same structure of an AGMS state, with two differences: (*i*) it does not store histories for adopted goals, selected strategy, or the history generated during execution; and (*ii*) it stores the current domain state. Formally, a PAGMS state is defined as follows:

$$\mathcal{I}^{\text{PAGMS}} = (\mathcal{D}, L, \sigma, s)$$

where: $\mathcal{D}$ is the FOND domain the agent operates in (assumed fixed) and s is the current domain state; $L = [\varphi_1, \cdots, \varphi_n]$ is the list of currently adopted LTL$_f$ goals, sorted in decreasing order of goal priority; and σ is the currently selected strategy. At each time step, the strategy σ

must be winning for L from state s – i.e., for every $\varphi_i \in L$, we must ensure that σ is winning for φ_i from s. We also say that L is realizable from state s iff there exists a strategy σ that is winning for L from s. As for the AGMS, the PAGMS state is initialized as $\mathcal{I}_0^{\text{PAGMS}} = (\mathcal{D}, [\,], \sigma, s_0)$, where $\mathcal{D}$ is the FOND domain with initial state s_0 and σ is an arbitrary strategy (the empty list of goals $[\,]$ is always realizable).

Adopt. The specification of $adopt(\phi, k)$ is defined analogously to that for the AGMS: ϕ is an LTL$_f$ goal and k is a priority index such that $1 \leq k \leq n+1$. Its specification is as follows. Given the PAGMS state $\mathcal{I}_m^{\text{PAGMS}} = (\mathcal{D}, L, \sigma, s)$, return the successor PAGMS state $\mathcal{I}_{m+1}^{\text{PAGMS}} = \mathcal{I}_m^{\text{PAGMS}}$ in case $L[(1..k-1)] \circ [\phi]$ is unrealizable; otherwise, return a successor PAGMS state $\mathcal{I}_{m+1}^{\text{PAGMS}} = (\mathcal{D}, R(L, \phi, n), \sigma', s)$. where: $\mathcal{D}$, s, and L are unchanged wrt $\mathcal{I}_m^{\text{PAGMS}}$; σ' is a winning strategy for $R(L, \phi, n)$; and $R(L, \phi, n)$ is the list of goals obtained by adding ϕ at priority index k and removing all lower priority goals that became unrealizable after ϕ has been adopted, which is defined recursively. Specifically, $R(L, \phi, j)$ is defined as: (1) $R(L, \phi, j) = [\varphi_1, \cdots, \varphi_{k-1}]$, if $j < k$; otherwise (2.1) $R(L, \phi, j) = R(L, \phi, j-1) \circ [\varphi_j]$ if the latter is realizable; or (2.2) $R(L, \phi, j) = R(L, \phi, j-1)$, otherwise.

Drop. The PAGMS specification of $drop(k)$ is also similar to that for the AGMS. Its specification is as follows. Given the PAGMS state $\mathcal{I}_m^{\text{PAGMS}} = (\mathcal{D}, L, \sigma, s)$, return a successor PAGMS state $\mathcal{I}_{m+1}^{\text{PAGMS}} = (\mathcal{D}, R, \sigma', s)$ where: $\mathcal{D}$ and s are unchanged wrt $\mathcal{I}_m^{\text{PAGMS}}$; $R = L[(1..k-1)] \circ L[k+1..n]$ – i.e., the same as L, except that goal φ_k at priority index k has been removed; and σ' is a winning strategy for R.

Step. Intuitively, the PAGMS specification of $step(s')$ is analogous to that for the AGMS: it instructs the PAGMS to perform the next action prescribed by the currently selected strategy, read the successor domain state s', and update the PAGMS state accordingly. However, the PAGMS does not store any history: we use the transition relation of the domain and LTL$_f$ progression to define the successor PAGMS state. In this way, the PAGMS is completely *memoryless*.

Formally, the PAGMS specification of $step(s')$ is as follows. Given the PAGMS state: $\mathcal{I}_m^{\text{PAGMS}} = (\mathcal{D}, L, \sigma, s)$, return the successor PAGMS state: $\mathcal{I}_{m+1}^{\text{PAGMS}} = (\mathcal{D}, L', \sigma', s')$, where: $\mathcal{D}$ is unchanged wrt $\mathcal{I}_m^{\text{PAGMS}}$; $L' = [\varphi_1', \cdots, \varphi_n']$ is the list of LTL$_f$ goals obtained as $\varphi_i' = prog(\varphi_i, s')$ for every $\varphi_i \in L$; σ' is the strategy obtained as $\sigma'(s_1 \cdots s_n) = \sigma(s \cdot s_1 \cdots s_n)$ for every sequence $s_1 \cdots s_n$ of domain states; and s' is the successor domain state (nondeterministically chosen by the environment) such that $(s, \sigma(s), s') \in \delta$.

PAGMS Problem Specification. Analogously to what was done for the AGMS, we define the PAGMS problem.

Definition 6 (PAGMS Problem)**.** *Let $\mathcal{I}_m^{\text{PAGMS}}$ be the current* PAGMS *state and* op *the next update operation. Compute a successor* PAGMS *state $\mathcal{I}_{m+1}^{\text{PAGMS}}$ that is the result of performing operation* op *in* PAGMS *state $\mathcal{I}_m^{\text{PAGMS}}$.*

We can easily obtain the PAGMS counterparts of the results stated in Theorems 3 and 4, specifically as follows.

Theorem 7. *Let $\mathcal{D}$ be a* FOND *domain with initial state s_0 and φ an* LTL$_f$ *goal: φ is realizable in $\mathcal{D}$ from s_0 iff the* PAGMS *state $\mathcal{I}_0^{\text{PAGMS}} = (\mathcal{D}, [\,], \sigma, s_0)$ after operation* $\mathtt{op} = adopt(\varphi, 1)$ *has a successor* PAGMS *state of the form $\mathcal{I}_1^{\text{PAGMS}} = (\mathcal{D}, [\varphi], \sigma_\varphi, s_0)$, where σ_φ is a winning strategy for φ in $\mathcal{D}$ from s_0.*

Theorem 8. *The* PAGMS *problem is* EXPTIME*-hard wrt $\mathcal{D}$ and 2*EXPTIME*-hard wrt adopted/dropped goals φ_i.*

Algorithms for the PAGMS. One important consequence of the PAGMS being memoryless is that it admits algorithms for $adopt(\phi, k)$ and $drop(k)$ that simply reduce to solving instances of LTL$_f$ synthesis in FOND domains. In what follows, we give a sound and complete algorithm for each of the three update operations in the PAGMS problem.

We consider operation $adopt(\phi, k)$ first. Its algorithm consists of three steps, which intuitively do the following. Step (1) checks whether ϕ is realizable jointly with higher priority intentions: if not, the successor PAGMS state is the same as the current PAGMS state; if yes, the algorithm proceeds to Step (2). The inner loop in Step (2.1) checks, one by one, whether intentions φ_i with priority lower than k are still realizable after the adoption of ϕ at priority index k: if yes, φ_i is added to set of goals R in the successor PAGMS state. Each check in Steps (1) and (2.1) is performed by solving an instance of LTL$_f$ synthesis in FOND domains. Step (3) returns the successor PAGMS state.

Adopt (PAGMS). Let $\mathcal{I}_m^{\text{PAGMS}} = [D, L, \sigma, s]$ be the current PAGMS state and $\mathtt{op} = adopt(\phi, k)$

(1) Compute a winning strategy for $L[(1..k-1)] \circ [\phi]$ in $\mathcal{D}$ from s, if any. If yes, proceed to Step 2; else, return $\mathcal{I}_{m+1}^{\text{PAGMS}} = \mathcal{I}_m^{\text{PAGMS}}$

(2) Define $R = L[(1..k-1)] \circ [\phi]$. For every i such that $k \leq i \leq n$, do the following:
   (2.1) Compute a winning strategy σ' for $R \circ [\varphi_i]$ in $\mathcal{D}$ from s, if any exists. If yes, update $R = R \circ [\varphi_i]$;

(3) Return $\mathcal{I}_{m+1}^{\text{PAGMS}} = [\mathcal{D}, R, \sigma', s]$, where σ' is a winning strategy for R in $\mathcal{D}$ from s.

We turn to operation $drop(k)$. Its algorithm essentially reduces to standard LTL$_f$ synthesis in FOND domains for the LTL$_f$ goal obtained as the conjunction of the currently adopted goals except that at index k (observe that in this case, synthesis is always realizable, since it is realizable for the conjunction of all currently adopted goals).

Drop (PAGMS). Let $\mathcal{I}_m^{\text{PAGMS}} = [D, L, \sigma, s]$ be the current PAGMS state and consider $\mathtt{op} = drop(k)$.

(1) Return $\mathcal{I}_{m+1}^{\text{PAGMS}} = [\mathcal{D}, R, \sigma', s]$ with: (*i*) $R = L[(1..k-1)] \circ L[(k+1..n)]$, i.e., R is the same as L, but with φ_k removed; (*ii*) σ' is a winning strategy for R in $\mathcal{D}$ from s.

We consider operation $step(s')$. Its algorithm computes the successor PAGMS state as follows: the list of goals is obtained by progressing each adopted goal in the current PAGMS state, Step (1), Item (*i*); the strategy is advanced one step forward from the current domain state, Step (1), Item (*ii*); and the successor domain state is set to s'.

Step (PAGMS). Let $\mathcal{I}_m^{\text{PAGMS}} = [D, L, \sigma, s]$ be the current PAGMS state and consider $\mathtt{op} = step(s')$.

(1) Return $\mathcal{I}_{m+1}^{\text{PAGMS}} = [\mathcal{D}, L', \sigma', s']$ with: (*i*) $L' = [\varphi_1', \cdots, \varphi_n']$, where $\varphi_i' = prog(\varphi_i, s')$ for every $\varphi_i \in L$; and (*ii*) $\sigma'(s_1 \cdots s_n) = \sigma(s \cdot s_1 \cdots s_n)$ for every sequence of domain states $s_1 \cdots s_n$.

It is easy to see that the algorithms for $adopt(\phi, k)$, $drop(k)$, and $step(s')$ are sound and complete wrt the

PAGMS specifications.

We analyze the complexity of the PAGMS problem. The algorithm for $step(s')$ computes progressed LTL_f goals – whose size is worst-case exponential in that of the non-progressed LTL_f goals (De Giacomo et al. 2022); the algorithms for $adopt(\phi, k)$ and $drop(k)$ reduces to LTL_f synthesis in FOND domains, which is EXPTIME- and 2EXPTIME-complete wrt domain and goal, respectively (De Giacomo and Rubin 2018). As a result, we obtain membership of the PAGMS problem in EXPTIME and 3EXPTIME wrt domain and goals, respectively.

Theorem 9. *The* PAGMS *problem is in* EXPTIME *wrt the domain $\mathcal{D}$ and 3*EXPTIME *wrt the adopted/dropped goals φ_i.*

Relation with AGMS. We conclude this section by investigating the relation between the PAGMS and AGMS. Specifically, we show that the AGMS and PAGMS are *bisimilar*, which intuitively means the following: despite their different internal representations, the AGMS and PAGMS yield the same results wrt operations $adopt(\phi, k)$, $drop(k)$, and $step(s')$. We formalize this result below.

First, we define when an AGMS and a PAGMS state are *isomorphic*. This means the following: (*i*) the AGMS state and PAGMS state are defined over the same underlying FOND domain; (*ii*) the LTL_f goals in the PAGMS state are exactly the progressed LTL_f goals in the AGMS state; (*iii*) the output prescribed by their strategies is the same in the current state for every sequence of domain states; and (*iv*) the domain state in the PAGMS state is the last reached domain state in the execution history of the AGMS state. Formally:

Definition 10. *Let $\mathcal{I}^{\text{AGMS}} = [\mathcal{D}_{\text{AGMS}}, L_{\text{AGMS}}, (\sigma_{\text{AGMS}}, h_\sigma), h]$ be an* AGMS *state and $\mathcal{I}^{\text{PAGMS}} = [\mathcal{D}_{\text{PAGMS}}, L_{\text{PAGMS}}, \sigma_{\text{PAGMS}}, s]$ be a* PAGMS *state. We say that $\mathcal{I}^{\text{AGMS}}$ and $\mathcal{I}^{\text{PAGMS}}$ are* isomorphic*, written $\mathcal{I}^{\text{AGMS}} \approx \mathcal{I}^{\text{PAGMS}}$, if:*

1. $\mathcal{D}_{\text{AGMS}} = \mathcal{D}_{\text{PAGMS}}$;
2. $s = \texttt{lst}(h)$;
3. $\varphi_i' = prog(\varphi, (h \backslash h_i)|_{2^{\mathcal{F}}}) \in L_{\text{PAGMS}}$ *iff* $(\varphi_i, h_i) \in L_{\text{AGMS}}$;
4. $\sigma_{\text{AGMS}}((h \setminus h_\sigma)|_{2^{\mathcal{F}}} \cdot \tau) = \sigma_{\text{PAGMS}}(\tau)$ *for every sequence of domain states τ.*

Next, we observe that, given an AGMS state and a PAGMS state that are isomorphic, regardless of the operation performed, the successor AGMS state has a successor PAGMS state that is isomorphic, and vice versa. This result can be proved constructively and is formally established below.

Theorem 11. *Let $\mathcal{I}_m^{\text{AGMS}}$ and $\mathcal{I}_m^{\text{PAGMS}}$ be an* AGMS *and* PAGMS *state such that $\mathcal{I}_m^{\text{AGMS}} \approx \mathcal{I}_m^{\text{PAGMS}}$. For every operation $\texttt{op} \in \{adopt(\phi, k), drop(k), step(s')\}$:*

1. *For every successor* AGMS *state $\mathcal{I}_{m+1}^{\text{AGMS}}$ there exists a successor* PAGMS *state $\mathcal{I}_{m+1}^{\text{PAGMS}}$ such that $\mathcal{I}_{m+1}^{\text{AGMS}} \approx \mathcal{I}_{m+1}^{\text{PAGMS}}$;*
2. *For every successor* PAGMS *state $\mathcal{I}_{m+1}^{\text{PAGMS}}$ there exists a successor* AGMS *state $\mathcal{I}_{m+1}^{\text{AGMS}}$ such that $\mathcal{I}_{m+1}^{\text{AGMS}} \approx \mathcal{I}_{m+1}^{\text{PAGMS}}$;*

We now define what it means for the AGMS and PAGMS to be *bisimilar*. Denote by Δ_{AGMS} and Δ_{PAGMS} the set of all AGMS states and PAGMS states, respectively.

Definition 12. *A relation $B \subseteq \Delta_{\text{AGMS}} \times \Delta_{\text{PAGMS}}$ is a* bisimulation *between* AGMS *and* PAGMS *if $(\mathcal{I}_m^{\text{AGMS}}, \mathcal{I}_m^{\text{PAGMS}}) \in B$ implies:*

1. *$\mathcal{I}_m^{\text{AGMS}} \approx \mathcal{I}_m^{\text{PAGMS}}$;*
2. *For every successor* AGMS *state $\mathcal{I}_{m+1}^{\text{AGMS}}$ that is the result of performing operation $\texttt{op} \in \{adopt(\phi, k), drop(k), step(s')\}$ in $\mathcal{I}_m^{\text{AGMS}}$, there exists a successor* PAGMS *state $\mathcal{I}_{m+1}^{\text{PAGMS}}$ that is the result of performing* op *in $\mathcal{I}_m^{\text{PAGMS}}$ and such that $(\mathcal{I}_{m+1}^{\text{AGMS}}, \mathcal{I}_{m+1}^{\text{PAGMS}}) \in B$;*
3. *For every successor* PAGMS *state $\mathcal{I}_{m+1}^{\text{PAGMS}}$ that is the result of performing operation $\texttt{op} \in \{adopt(\phi, k), drop(k), step(s')\}$ in $\mathcal{I}_m^{\text{PAGMS}}$, there exists a successor* AGMS *state $\mathcal{I}_{m+1}^{\text{AGMS}}$ that is the result of performing* op *in $\mathcal{I}_m^{\text{AGMS}}$ and such that $(\mathcal{I}_{m+1}^{\text{AGMS}}, \mathcal{I}_{m+1}^{\text{PAGMS}}) \in B$;*

Definition 13. *The* AGMS *and* PAGMS *are bisimilar if there exists a bisimulation relation B between them such that:*

1. *For every initial* AGMS *state $\mathcal{I}_0^{\text{AGMS}}$, there exists an initial* PAGMS *state $\mathcal{I}_0^{\text{PAGMS}}$ such that $(\mathcal{I}_0^{\text{AGMS}}, \mathcal{I}_0^{\text{PAGMS}}) \in B$;*
2. *For every initial* PAGMS *state $\mathcal{I}_0^{\text{PAGMS}}$, there exists an initial* AGMS *state $\mathcal{I}_0^{\text{AGMS}}$ such that $(\mathcal{I}_0^{\text{AGMS}}, \mathcal{I}_0^{\text{PAGMS}}) \in B$;*

The AGMS and PAGMS are indeed bisimilar. Consider the bisimulation B with $(\mathcal{I}^{\text{AGMS}}, \mathcal{I}^{\text{PAGMS}}) \in B$ for every AGMS state $\mathcal{I}^{\text{AGMS}}$ and PAGMS state $\mathcal{I}^{\text{PAGMS}}$ such that $\mathcal{I}^{\text{AGMS}} \approx \mathcal{I}^{\text{PAGMS}}$. Observe that B is indeed a bisimulation: Item 1 above holds since $(\mathcal{I}^{\text{AGMS}}, \mathcal{I}^{\text{PAGMS}}) \in B$ implies $\mathcal{I}^{\text{AGMS}} \approx \mathcal{I}^{\text{PAGMS}}$ by construction of B; Items 2 and 3 hold as a consequence of Theorem 11. It remains to show that B satisfies Definition 13. To see this: observe that every initial AGMS state $\mathcal{I}_0^{\text{AGMS}} = (\mathcal{D}, [\,], (\sigma, s_0), s_0)$ is isomorphic to the initial PAGMS state $\mathcal{I}_0^{\text{PAGMS}} = (\mathcal{D}, [\,], \sigma, s_0)$, and vice versa, so that $(\mathcal{I}_0^{\text{AGMS}}, \mathcal{I}_0^{\text{PAGMS}}) \in B$. As a result, we obtain the following:

Theorem 14. *The* AGMS *and* PAGMS *are bisimilar.*

Theorem 14 gives us a reduction of the AGMS problem to the PAGMS problem, meaning that any implementation of the PAGMS is also an implementation of the AGMS. Furthermore, by Theorem 9, we obtain membership of the AGMS problem in EXPTIME and 3EXPTIME wrt domain and adopted/dropped goals, respectively. However, the bound wrt goals is not tight, as we will show in the next section.

5 Automata-Based Agent Goal Management System

In this section, we provide a specification for a goal management system based on deterministic finite automata (DFA) instead of LTL_f formulas, which we call Automata-Based Agent Goal Management System (AAGMS).

Background on Deterministic Finite Automata. A *deterministic transition system* is a tuple $\mathcal{T} = (\Sigma, S, \delta, F)$, where: Σ is a finite input alphabet; S is a finite set of states; $\delta : S \times \Sigma \to S$ is the transition function; and $F \subseteq S$ is the set of final states. 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 *deterministic finite automaton* (DFA) is a pair $\mathcal{A} = (\mathcal{T}, \iota)$, where $\mathcal{T}$ is a deterministic transition system and $\iota \in S$ is the initial state. The size of $\mathcal{A}$ is $|S|$. 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. Observe that, for convenience, our notion of DFAs separates the initial state from

the rest of the automaton, but it is equivalent to the standard notion of DFA defined, e.g., in (Sipser 1997).

Theorem 15. *(De Giacomo and Rubin 2018) Given a* FOND *domain $\mathcal{D}$ and a state s_0, we can construct a* DFA*, denoted $\mathcal{A}_\mathcal{D} = (\mathcal{T}_\mathcal{D}, s_0)$ with size at most exponential in that of $\mathcal{D}$ and whose language is the set of domain traces from s_0.*

Theorem 16. *(De Giacomo and Vardi 2015) Given an* LTL_f *formula φ, we can construct a* DFA *$\mathcal{A}_\varphi = (\mathcal{T}_\varphi, \iota_\varphi)$ of size at most double exponential in that of φ and whose language is the set of finite traces that satisfy φ.*

A DFA game is a DFA $\mathcal{G} = (\mathcal{T}_\mathcal{G}, \iota_\mathcal{G})$ with $\mathcal{T}_\mathcal{G} = (Act \times React, S, \delta, F)$, where where Act and $React$ are disjoint sets of actions and reactions under control of agent and environment, respectively. A game strategy is a function $\kappa : S \to Act$ that maps states of the game to agent actions. Given a game strategy κ, a sequence of environment reactions $\vec{r} = r_0 r_1 \ldots \in React^\omega$, and a DFA game $\mathcal{G}$, we denote by $Run(\kappa, \vec{r}, \mathcal{G})$, the *run* $s_0 s_1 \ldots \in S^\omega$ of states of $\mathcal{G}$ *induced* by (or *consistent with*) κ and $\vec{r}$ as follows: s_0 is the initial state of $\mathcal{G}$, and for every $i \geq 0$, we have $s_{i+1} = \delta(s_i, (a_i, r_i))$, where $a_i = \kappa(s_i)$. A game strategy is winning in $\mathcal{G}$ if, for every $\vec{r} \in React^\omega$, there exists some i in $Run(\kappa, \vec{r}, \mathcal{G}) = s_0 s_1 \cdots$ such that $s_i \in F$. That is, DFA games requires the set of final states to be visited at least once. The *winning region* W is the set of states s in which the agent has a winning strategy in the game $\mathcal{G}' = (\mathcal{T}_\mathcal{G}, s)$, i.e., the same game as $\mathcal{G}$, but with initial state s. *Solving* a DFA game is the problem of computing the winning region and a winning game strategy, if any. DFA games can be solved in polynomial time in their size by a backward search algorithm that performs a least fixpoint computation over the state space of the game (Apt and Grädel 2011). There exists a winning game strategy in $\mathcal{G} = (\mathcal{T}_\mathcal{G}, \iota_\mathcal{G})$ iff $\iota_\mathcal{G} \in W$.

We combine DFAs using (synchronous) product (Baier and Katoen 2008). We denote by $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ the product of $\mathcal{A}_1, \cdots, \mathcal{A}_n$ such that $\mathcal{L}(\mathcal{A}) = \bigcap_{1 \leq i \leq n} \mathcal{L}(\mathcal{A}_i)$. The product automaton can be computed in polynomial time in the size of its state space, i.e., $S_1 \times \cdots \times S_n$, where S_i is the state space of $\mathcal{A}_i$.

Theorem 17. *(De Giacomo and Rubin 2018) Let $\mathcal{D}$ be a* FOND *domain with initial state s and φ an* LTL_f *goal. There exists a winning strategy for φ in $\mathcal{D}$ from s iff there exists a winning game strategy in the* DFA *game $\mathcal{A}_\mathcal{D} \times \mathcal{A}_\varphi$ where agent and environment control Act and $2^\mathcal{F}$, respectively.*

A *transducer* is defined as a tuple $\Pi = (\mathcal{T}, \iota, \kappa)$, where: $\mathcal{T} = (Act \times React, S, \delta, F)$ is a deterministic transition system; $\iota \in S$ is the initial state; and κ is a game strategy (aka *output function* in the context of transducers). A transducer Π is equivalent to an agent strategy of the form $\sigma : (React)^* \to Act$ defined as follows: $\sigma(r_0 r_1 \cdots) = \kappa(\delta(\tau))$ for every $\tau = (a_0, r_0)(a_1, r_1) \cdots \in (Act \times React)^*$. Let: $\mathcal{D}$ be a FOND domain with initial state s; φ an LTL_f goal; $\mathcal{A}_\mathcal{D} \times \mathcal{A}_\varphi = (\mathcal{T}, \iota)$ their corresponding DFA game; and κ a winning game strategy. The transducer $\Pi = (\mathcal{T}, \iota, \kappa)$ is a winning strategy σ for φ in $\mathcal{D}$ from s. In the remainder of this section, we assume strategies represented as transducers. This does not cause any loss of generality wrt synthesis setting in Section 2: there exists a winning strategy for φ in $\mathcal{D}$ from s iff there exists one represented as a transducer (De Giacomo and Rubin 2018).

We now give the specification for the AAGMS. Intuitively, the formal definition of AAGMS state is along the same lines as those for the AGMS and PAGMS state, with two exceptions: (*i*) it uses DFAs to represent the FOND domain and LTL_f goals; and (*ii*) it represents the strategy as a transducer. Formally, an AAGMS state is a tuple:

$$\mathcal{I}_m^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma)$$

where: $(\mathcal{T}_\mathcal{D}, s_\mathcal{D})$ is the DFA of the FOND domain; $L = [(\mathcal{T}_{\varphi_1}, \iota_{\varphi_1}), \cdots, (\mathcal{T}_{\varphi_n}, \iota_{\varphi_n})]$ is a list of DFAs $(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})$ representing goals, where φ_i is the currently adopted i-th LTL_f goal, sorted in decreasing order of priority; and $\Pi_\sigma = (\mathcal{T}_\sigma, \iota_\sigma, \kappa_\sigma)$ is the currently selected strategy represented as a transducer. At each time step, we require Π_σ to be winning for L, i.e., κ_σ is a winning game strategy in the DFA game $(\mathcal{T}_\mathcal{D}, s_\mathcal{D}) \times (\mathcal{T}_{\varphi_1}, \iota_{\varphi_1}) \times \cdots \times (\mathcal{T}_{\varphi_n}, \iota_{\varphi_n})$. We also say that L is realizable if there exists a winning strategy Π_σ for it. The AAGMS initial state is initialized as $\mathcal{I}_0^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), [\,], \Pi_\sigma)$, where Π_σ is an arbitrary strategy (the empty list of goals is always realizable).

Adopt. The specification of $adopt(\phi, k)$ is similar to that for AGMS and PAGMS: ϕ is an LTL_f goal and k is a priority index such that $1 \leq k \leq n + 1$. Formally, its specification is as follows. Given the PAGMS state $\mathcal{I}_m^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma)$, return the successor PAGMS state $\mathcal{I}_{m+1}^{\text{AAGMS}} = \mathcal{I}_m^{\text{PAGMS}}$ in case $L[(1..k-1)] \circ [(\mathcal{T}_\phi, \iota_\varphi)]$ is unrealizable, where $(\mathcal{T}_\phi, \iota_\phi)$ is the DFA that accepts exactly the traces that satisfy ϕ; otherwise, return a successor PAGMS state $\mathcal{I}_{m+1}^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), R(L, (\mathcal{T}_\phi, \iota_\phi), n), \Pi_{\sigma'})$ where: $\mathcal{T}_\mathcal{D}$ and $s_\mathcal{D}$ are unchanged wrt $\mathcal{I}_m^{\text{AAGMS}}$; $\Pi_{\sigma'}$ is a winning strategy for $R(L, (\mathcal{T}_\phi, \iota_\phi), n)$; and $R(L, (\mathcal{T}_\phi, \iota_\phi), n)$ is the list of DFAs obtained by adding $(\mathcal{T}_\phi, \iota_\phi)$ at priority index k and removing all lower priority goals that became unrealizable after ϕ has been adopted, which is defined recursively. Specifically, $R(L, (\mathcal{T}_\phi, \iota_\phi), j)$ is defined as follows: (1) $R(L, \phi, j) = [(\mathcal{T}_{\varphi_1}, \iota_{\varphi_1}), \cdots, (\mathcal{T}_{\varphi_{k-1}}, \iota_{\varphi_{k-1}}), (\mathcal{T}_\phi, \iota_\phi)]$, if $j < k$; otherwise (2.1) $R(L, (\mathcal{T}_\phi \iota_\phi), j) = R(L, (\mathcal{T}_\phi, \iota_\phi), j - 1) \circ [(\mathcal{T}_{\varphi_j}, \iota_{\varphi_j})]$ if the latter is realizable; or (2.2) $R(L, (\mathcal{T}_\phi \iota_\phi), j) = R(L, (\mathcal{T}_\phi, \iota_\phi), j - 1)$.

Drop. The AAGMS specification of $drop(k)$ is also similar to that for the AGMS and PAGMS. Its specification is as follows. Given the PAGMS state $\mathcal{I}_m^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma)$, return a successor PAGMS state $\mathcal{I}_{m+1}^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), R, \Pi_{\sigma'})$, where: $\mathcal{T}_\mathcal{D}$ and $s_\mathcal{D}$ are unchanged wrt $\mathcal{I}_m^{\text{AAGMS}}$; $R = L[(1..k-1)] \circ L[k+1..n]$ – i.e., the same as L, except that DFA $(\mathcal{T}_{\varphi_k}, \iota_{\varphi_k})$ at priority index k has been removed; and $\Pi_{\sigma'}$ is a winning strategy for R.

Step. Intuitively, the AAGMS specification of $step(s')$ is analogous to that for the AGMS and PAGMS: it instructs the AAGMS to perform the next action prescribed by the currently selected strategy, read the successor domain state s', and update the AAGMS state accordingly. In this case, however, we use the transition function of the DFAs of the domain and LTL_f goals to define the successor AAGMS state.

Formally, the AAGMS specification of $step(s')$ is as follows. Given the AAGMS state $\mathcal{I}_m^{\text{PAGMS}} = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma)$, return the successor AAGMS state

$\mathcal{I}^{\text{AAGMS}}_{m+1} = ((\mathcal{T}_\mathcal{D}, s'), L', \Pi'_\sigma)$, where: $\mathcal{T}_\mathcal{D}$ is unchanged wrt $\mathcal{I}^{\text{AAGMS}}_{m+1}$; s' is the successor domain state (nondeterministically chosen by the environment) such that $(s, \kappa_\sigma(\iota_\sigma), s') \in \delta$; $L' = [(\mathcal{T}_{\varphi_1}, \iota'_{\varphi_1}), \cdots, (\mathcal{T}_{\varphi_n}, \iota'_{\varphi_n})]$ is the list of DFAs obtained by advancing the initial state ι_{φ_i} to $\iota'_{\varphi_i} = \delta_{\varphi_i}(s_{\varphi_i}, s')$, where δ_{φ_i} is the transition function of $\mathcal{T}_{\varphi_i}$, for every $(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i}) \in L$; and $\Pi'_\sigma = (\mathcal{T}_\sigma, \iota'_\sigma, \kappa_\sigma)$ is obtained by advancing the initial state ι_σ to $\iota'_\sigma = \delta_\sigma(\iota_\sigma, (\kappa_\sigma(\iota_\sigma), s'))$, where δ_σ is the transition function of $\mathcal{T}_\sigma$.

AAGMS Problem Specification. As for the AGMS and PAGMS, we provide the AAGMS problem specification.

Definition 18 (AAGMS Problem)**.** *Let $\mathcal{I}^{\text{AAGMS}}_m$ be the current* AAGMS *state and* op *the next update operation. Compute a successor* AAGMS *state $\mathcal{I}^{\text{AAGMS}}_{m+1}$ that is the result of performing operation* op *in* AAGMS *state $\mathcal{I}^{\text{AAGMS}}_m$.*

Theorem 19. *Let $\mathcal{D}$ be a* FOND *domain with initial state s_0 and φ an* LTL_f *goal: φ is realizable in $\mathcal{D}$ from s_0 iff the* AAGMS *state $\mathcal{I}^{\text{AAGMS}}_0 = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), [\,], \Pi_\sigma)$ after operation $\texttt{op} = adopt(\varphi, 1)$ has a successor* AAGMS *state of the form $\mathcal{I}^{\text{AGMS}}_1 = ((\mathcal{T}_\mathcal{D}, s_\mathcal{D}), [(\mathcal{T}_\varphi, \iota_\varphi)], \Pi_{\sigma_\varphi}$, where $(\mathcal{T}_\varphi, \iota_\varphi)$ is the* DFA *of φ and σ_φ is a winning strategy for φ in $\mathcal{D}$ from s_0.*

Theorem 20. *The* AAGMS *problem is* EXPTIME*-hard wrt $\mathcal{D}$ and* 2EXPTIME*-hard wrt adopted/dropped goals φ_i.*

Algorithms for the AAGMS. We now give sound and complete algorithms for each of the three update operations in the AAGMS problem. The main intuition for the algorithms for $adopt(\phi, k)$ and $drop(k)$ is to reduce to solving DFA games based on Theorems 15, 16, and 17. An analysis of the complexity of these algorithms allows us to establish the computational complexity for the AAGMS problem.

We consider operation $adopt(\phi, k)$ first. Its algorithm consists of four steps, which intuitively do the following. Step (1) constructs the DFA $(\mathcal{T}_\phi, \iota_\phi)$ that accepts exactly the traces that satisfy ϕ. Step (2) solves a DFA game obtained from the DFAs of the domain and higher priority goals as well as $(\mathcal{T}_\phi, \iota_\phi)$ to decide whether the latter can be adopted: if yes, the algorithm proceeds to Step (3). The inner loop in Step (3.1) solves a DFA obtained to decide whether the DFA $(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})$ of a goal with priority lower than k can be kept after the adoption of $(\mathcal{T}_\phi, \iota_\phi)$ at priority index k: if yes $(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})$ is added to the list of DFAs R in the successor AAGMS state. Step (4) returns the successor AAGMS state.

Adopt (AAGMS). Let $\mathcal{I}^{\text{AAGMS}}_m = [(\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma]$ be the current AAGMS state and $\texttt{op} = adopt(\phi, k)$:

(1) Construct the DFA $(\mathcal{T}_\phi, \iota_\phi)$ that accepts exactly the traces that satisfy φ;
(2) Construct DFA game $L[(1..k-1)] \circ [(\mathcal{T}_\varphi, \iota_\phi)]$ and compute a winning strategy $\Pi_{\sigma'}$, if any. If yes, proceed to Step (3); else, return $\mathcal{I}^{\text{AAGMS}}_{m+1} = \mathcal{I}^{\text{AAGMS}}_m$
(3) Define $R = L[(1..k-1)] \circ [(\mathcal{T}_\phi, \iota_\phi)]$. For every i such that $k \leq i \leq n$, do the following:
  (3.1) Construct DFA game $R \circ [(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})]$ and compute a winning strategy $\Pi_{\sigma'}$, if any. If yes, update $R = R \circ [(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})]$;
(4) Return $\mathcal{I}^{\text{PAGMS}}_{m+1} = [(\mathcal{T}_\mathcal{D}, s_\mathcal{D}), R, \Pi_{\sigma'}]$, where σ' is a winning strategy in R.

We turn to operation $drop(k)$. Its algorithm essentially reduces to solving a DFA game obtained from the DFAs of the FOND domain and currently adopted goals, exception made for that at index k.

Drop (AAGMS). Let $\mathcal{I}^{\text{AAGMS}}_m = [(\mathcal{T}_\mathcal{D}, s_\mathcal{D}), L, \Pi_\sigma]$ be the current AAGMS state and $\texttt{op} = drop(k)$:

(1) Return $\mathcal{I}^{\text{AAGMS}}_{m+1} = [\mathcal{D}, R, \Pi_{\sigma'}]$ with: (*i*) $R = L[(1..k-1)] \circ L[(k+1..n)]$, i.e., R is the same as L, but with φ_k removed; and (*ii*) $\Pi_{\sigma'}$ is a winning strategy in R.

The algorithm for **Step (AAGMS)** is immediate by the specification of $step(s')$.

We analyze the complexity of the AAGMS problem. The algorithm for $step(s')$ progresses the initial state of the DFAs of the FOND domain and LTL_f goals, which can be done in polynomial time and does not modify their sizes; the algorithms for $adopt(\phi, k)$ and $drop(k)$ solve games that are worst-case exponential and doubly-exponential in the size of the domain and added/dropped goals, respectively – which can be done in polynomial time in the size of the games. As a result, we obtain membership of the AAGMS problem in EXPTIME and 2EXPTIME wrt the domain and goals, respectively. Together with Theorem 20, we establish tight bounds for the AAGMS problem.

Theorem 21. *The* AAGMS *problem is* EXPTIME*-complete wrt the domain $\mathcal{D}$ and* 2EXPTIME*-complete wrt the adopted/dropped goals φ_i.*

Relation with PAGMS and AGMS. We now investigate the relation between the AAGMS and AGMS. Specifically, we show that the AAGMS and AGMS are bisimilar: they yield the same results wrt any sequence of operations $adopt(\phi, k)$, $drop(k)$, and $step(s')$. The same can be shown for the AAGMS and PAGMS. Based on these results, we establish tight complexity bounds for the AGMS and PAGMS problems.

First, we define when an AGMS and AAGMS state are *isomorphic*. This means the following: (*i*) The DFAs stored in the AAGMS accepts exactly the domain traces that satisfy the LTL_f goals in the AGMS, the latter taking into account what has been observed since the goal has been adopted; and (*ii*) the output prescribed by their strategies is the same in the current state for every sequence of domain states. Formally:

Definition 22. *Let $\mathcal{I}^{\text{AGMS}} = [\mathcal{D}_{\text{AGMS}}, L_{\text{AGMS}}, (\sigma, h_\sigma), h]$ be an* AGMS *state and $\mathcal{I}^{\text{AAGMS}} = [(\mathcal{T}_{\text{AAGMS}}, s_{\text{AAGMS}}), L_{\text{AAGMS}}, \Pi]$ an* AAGMS *state. We say that $\mathcal{I}^{\text{AGMS}}$ and $\mathcal{I}^{\text{AAGMS}}$ are* isomorphic*, written $\mathcal{I}^{\text{AGMS}} \approx \mathcal{I}^{\text{AAGMS}}$, if:*

1. *$(\mathcal{T}_{\text{AAGMS}}, s_{\text{AAGMS}})$ is the* DFA *that accepts exactly the domain traces of $\mathcal{D}_{\text{AGMS}}$ from $\texttt{lst}(h)$;*
2. *$s_{\text{AAGMS}} = \texttt{lst}(h)$;*
3. *$(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i}) \in L_{\text{AAGMS}}$ and $\tau \in \mathcal{L}(\mathcal{T}_{\varphi_i}, \iota_{\varphi_i})$ iff $(\varphi_i, h_i) \in L_{\text{AGMS}}$ and $((h \setminus h_i) \cdot \tau)|_{2^\mathcal{F}} \models \varphi_i$;*
4. *$\sigma_{\text{AGMS}}((h \setminus h_\sigma)|_{2^\mathcal{F}} \cdot \tau) = \Pi_{\text{AAGMS}}(\tau)$ for every sequence of domain states τ.*

The notions of bisimulation and bisimilarity for the AGMS and AAGMS can be obtained by adapting those for the AGMS and PAGMS: it is sufficient to replace PAGMS with AAGMS in Definitions 12 and 13. In order to establish bisimilarity, we observe that, given an AGMS state and an AAGMS state that are isomorphic, regardless of the operation performed, the successor AGMS state has a successor AAGMS state that is isomorphic, and vice versa. This result can be proved constructively and is analogous to Theorem 11 for the AGMS

and PAGMS. That the AGMS and AAGMS are bisimilar can also be proved analogously to proving that the AGMS and PAGMS are bisimilar. Regarding the initial states, the main observation is as follows: every initial AGMS state $\mathcal{I}_0^{\text{AGMS}} = (\mathcal{D}, [\,], (\sigma, s_0), s_0)$ is isomorphic to the initial AAGMS state $\mathcal{I}_0^{\text{AAGMS}} = ((\mathcal{T}_\mathcal{D}, s_0), [\,], \Pi_\sigma)$, and viceversa, where $(\mathcal{T}_\mathcal{D}, s_0)$ is the DFA of the FOND domain and Π_σ is the representation of σ as a transducer. Hence, we have:

Theorem 23. *The* AGMS *and the* AAGMS *are bisimilar.*

Analogously, we can prove that the AAGMS and PAGMS are also bisimilar.

Theorem 24. *The* AAGMS *and the* PAGMS *are bisimilar.*

Finally, we obtain tight complexity bounds for the AGMS and PAGMS problems by Theorems 21, 23 and 24.

Theorem 25. *The* AGMS/PAGMS *problem is* EXPTIME*-complete wrt* $\mathcal{D}$ *and* 2EXPTIME*-complete wrt added/dropped goals* φ_i*.*

6 Empirical Analysis

We implemented both the PAGMS and AAGMS specifications in a tool called ISABEL[4]. ISABEL 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. ISABEL represents state space and transition function of symbolic DFAs using Binary Decision Diagrams (Bryant 1992), with CUDD-3.0.0 (Somenzi 1998) as the underlying BDD library. The FOND domain is specified in PDDL (Haslum et al. 2019); the symbolic DFA of the domain is obtained using the PDDL-to-DFA transformation shown in (De Giacomo, Di Stasio, and Parretti 2025).

We now present a preliminary empirical analysis for the PAGMS and AAGMS. The goal of the empirical analysis is to assess whether the AAGMS has better performance than the PAGMS in practice, thus confirming the results in our theoretical findings, see Theorems 9 and 21. To do so, we compare the performance of the ISABEL implementations for the AAGMS and PAGMS in some scalable benchmarks.

Experimental Methodology. We considered TIREWORLD benchmarks for our empirical analysis inspired by those used in FOND planning (Muise, McIlraith, and Beck 2012). An *instance* of TIREWORLD 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 AAGMS and PAGMS before the timeout, as well as the time required to add each goal. To reproduce the online 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 20.04, 3.6 GHz CPU, and 8 GBs memory.

In TIREWORLD benchmarks, 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). Instances in TIREWORLD benchmarks are parametrized by the number of locations ℓ. In each instance, we generate goals φ_n, with $n \leq 30$, as follows: the agent must traverse locations from 0 to $\ell - 1$ in sequence, for a total of $2n + 1$ visits, written in LTL$_f$ $\varphi_n \doteq$

$$(\Diamond(at_0 \wedge \bigcirc(\Diamond at_1 \wedge \cdots \wedge \bigcirc(\Diamond at_{\ell-1} \wedge \bigcirc(\Diamond at_0 \wedge \cdots \wedge \bigcirc(\Diamond at_{2n+1 \bmod \ell}) \cdots)))))$$

Where n is the goal parameter. We consider TIREWORLD instances with varying numbers of locations ℓ such that $5 \leq \ell \leq 10$, denoted TIREWORLD-ℓ; in each instance, we consider goals with $n \leq 30$, for a total of 180 goals.

Instance	Added Goals (Low)		Avg. Runtime (s)	
	AAGMS	PAGMS	AAGMS	PAGMS
TIREWORLD-5	**27**/30	20/30	**12.56**	130.82
TIREWORLD-6	**21**/30	19/30	**38.22**	182.96
TIREWORLD-7	**17**/30	**17**/30	**52.38**	190.10
TIREWORLD-8	**15**/30	**15**/30	**23.60**	125.40
TIREWORLD-9	**13**/30	**13**/30	**19.65**	56.52
TIREWORLD-10	**12**/30	**12**/30	**32.04**	39.22
Total	**105**/180	96/180		

Instance	Added Goals (High)		Avg. Runtime (s)	
	AAGMS	PAGMS	AAGMS	PAGMS
TIREWORLD-5	**27**/30	15/30	**1.85**	228.35
TIREWORLD-6	**21**/30	14/30	**3.76**	202.80
TIREWORLD-7	**17**/30	13/30	**5.23**	208.34
TIREWORLD-8	**15**/30	12/30	**5.73**	158.93
TIREWORLD-9	**13**/30	12/30	**19.50**	288.33
TIREWORLD-10	**12**/30	11/30	**17.72**	168.12
Total	**105**/180	77/180		

Table 1: Comparison of the number of goals added at lowest (top) and highest (bottom) priority index and average time to add a goal achieved by the AAGMS and PAGMS. Best-performing implementation in bold.

Empirical Results. Table 1 compares the number of goals added by the AAGMS and PAGMS at the lowest and highest priority indexes 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. It can be seen that in both cases the AAGMS performs better than the PAGMS, gaining up to one or two orders of magnitude on average. In fact, the performance gap between the two becomes smaller as the size of TIREWORLD instance grows. 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 PAGMS and AAGMS, until *they both run out of memory during the construction of the* DFA *game arena for computing the new strategy*. Overall, the empirical analysis shows the superiority of the AAGMS to the PAGMS, thus confirming the results in our theoretical findings. In fact, by Theorem 24, the AAGMS can be seen as a more efficient implementation of the PAGMS that avoids recomputing the DFAs of the progressed goals by caching them.

7 Conclusion

The update operations supported by our AGMS framework can be viewed as analogous to that for belief revision, but

[4]https://github.com/GianmarcoDIAG/ISabel/tree/fond

focusing on goals. Note that unlike in the belief revision case where one can only specify postulates/constraints on the results, our goal adoption/dropping operations produce a unique resulting set of prioritized goals given our assumptions of full observability and total ordering on goal priority. (Icard, Pacuit, and Shoham 2010) proposes postulates for joint revision of belief and goals. But their goals are restricted to doing a given action at a given time. (Baral and Zhao 2008) proposes a framework for goal revision; but they do not deal with how goals change after actions. (van Benthem and Liu 2007) develops a dynamic epistemic logic where agents' preferences can be updated following suggestions or commands. (Khan and Lespérance 2010; Khan 2018) propose a general account of knowledge and goal change based on the situation calculus. It handles incomplete knowledge and knowledge-producing actions, as well as temporally extended desires and goals. But it does not deal with nondeterministic effects and only requires that the agent's goals be consistent with each other and with what is known, so it does not ensure that the agent can achieve them no matter how the environment behaves.

Observe that we do not make any assumption about how the agent selects its goals and establishes their priorities (which we take as given). Work on goal reasoning (Aha 2018) addresses goal selection and refinement as well as related issues, such as plan execution, monitoring, and strategy repair. Note that goal management with goals totally ordered by priority is related to planning with hard constraints (Bonassi et al. 2021; Bonassi, Gerevini, and Scala 2022) and soft preferences (Bienvenu, Fritz, and McIlraith 2006; Baier and McIlraith 2008; Baier, Bacchus, and McIlraith 2009): the constraints and preferences can be viewed as the goals that the agent should adopt at the highest and lowest priority levels, respectively.

In the synthesis literature, reasoning about dynamically changing goals has been studied in *live synthesis* (Finkbeiner, Klein, and Metzger 2022) and *synthesis with duties and rights* (Zhu and De Giacomo 2022). Live synthesis deals with goals being updated (vs. new goals being added); when a goal is updated, the agent must update its strategy to satisfy the new goal and only parts of the original goal, the so-called *pending obligations*. Synthesis with duties and rights deals with synthesizing strategies that satisfy the "duty" goal, but can also be adapted during execution to satisfy the "right" goal (which is known at the beginning vs. received during execution) if the agent chooses to do so.

In this paper, we aim to provide a foundational framework for agents that reason about dynamically changing goals based on reactive synthesis. Extensions can be built by leveraging previous work in the field. For instance, suppose that the assumption of full observability does not hold: we can adapt the AGMS specification to handle this based on *synthesis with partial observability* (De Giacomo and Vardi 2016; Tabajara and Vardi 2020). In future work, it would also be interesting to perform an experimental comparison of our approaches with forward-search methods used in synthesis and planning (Camacho and McIlraith 2019; De Giacomo et al. 2022; Bonassi et al. 2023; Duret-Lutz et al. 2025), which may be faster when the state space is large.

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.

AI Declaration

The authors have not employed any Generative AI tools.

References

Aha, D. W. 2018. Goal reasoning: Foundations, emerging applications, and prospects. *AI Mag.*

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. *Artif. Intell.*

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

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. *AI Mag.*

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.*

Baral, C., and Zhao, J. 2008. Non-monotonic temporal logics that facilitate elaboration tolerant revision of goals. In *AAAI*.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. In *KR*.

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.; 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.; and Scala, E. 2022. Planning with qualitative action-trajectory constraints in PDDL. In *IJCAI*.

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*.

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 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*.

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

De Giacomo, G., and Vardi, M. Y. 2016. LTL_f and LDL_f synthesis under partial observability. In *IJCAI*.

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.; Di Stasio, A.; and Parretti, G. 2025. PDDL to DFA: A symbolic transformation for effective reasoning. In *TIME*.

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

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*.

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

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*.

Icard, T.; Pacuit, E.; and Shoham, Y. 2010. Joint revision of beliefs and intention. In *KR*.

Khan, S. M., and Lespérance, Y. 2010. A logical framework for prioritized goal change. In *AAMAS*.

Khan, S. M. 2018. *Rational Agents: Prioritized Goals, Goal Dynamics, and Agent Programming Languages with Declarative Goals*. Ph.D. Dissertation, York University, Toronto, Canada.

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

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*.

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. LTLf synthesis under partial observability: From theory to practice. In *GandALF*.

van Benthem, J., and Liu, F. 2007. Dynamic logic of preference upgrade. *J. Appl. Non Class. Logics*.

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

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*.