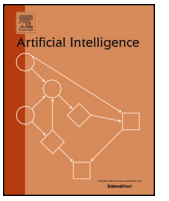
Artificial Intelligence 348 (2025) 104409

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Planning for temporally extended goals in pure-past linear temporal logic

Luigi Bonassi [a,b,*], Giuseppe De Giacomo [b,c], Marco Favorito [d,1], Francesco Fuggitti [d,**,1], Alfonso Emilio Gerevini [a], Enrico Scala [a]

[a] University of Brescia, Brescia, Italy
[b] University of Oxford, Oxford, United Kingdom
[c] Sapienza University, Rome, Italy
[d] Bank of Italy, Rome, Italy

ABSTRACT

We study planning for temporally extended goals expressed in Pure-Past Linear Temporal Logic (PPLTL) in the context of deterministic (i.e., classical) and fully observable nondeterministic (FOND) domains. PPLTL is the variant of Linear-time Temporal Logic on finite traces (LTL$_f$) that refers to the past rather than the future. Although PPLTL is as expressive as LTL$_f$, we show that it is computationally much more effective for planning. In particular, we show that checking the validity of a plan for a PPLTL formula is Markovian. This is achieved by introducing a linear number of additional propositional variables that capture the validity of the entire formula in a modular fashion. The solution encoding introduces only a linear number of new fluents proportional to the size of the PPLTL goal and does not require any additional spurious action. We implement our solution technique in a system called Plan4Past, which can be used alongside state-of-the-art classical and FOND planners. Our empirical analysis demonstrates the practical effectiveness of Plan4Past in both classical and FOND problems, showing that the resulting planner performs overall better than other planning approaches for LTL$_f$ goals.

1. Introduction

In AI Planning, a temporally extended goal is a (possibly complex) property that the state trace induced by a plan must satisfy. In this scenario, a powerful formalism to express temporally extended goals is Linear-time Temporal Logic (LTL), which has been advocated as an excellent tool to express properties of processes in Formal Methods [10]. Given that tasks in AI planning are inherently of finite nature, meaning that their execution stops after a finite number of steps, a finite-trace variant of LTL, namely LTL$_f$, has often been employed [7,13,37]. An alternative to LTL$_f$ is Pure-Past Linear Temporal Logic [60], or PPLTL. PPLTL looks at the trace backward, instead of forward as in LTL$_f$, by expressing properties on traces using only past operators. PPLTL and LTL$_f$ have the same expressive power, but translating a formula from one to the other (and vice versa) can be prohibitive since the best-known algorithms are 3EXPTIME [33].

Planning for an LTL$_f$ goal requires synthesizing a sequence of states that satisfies the LTL$_f$ goal formula. Similarly, planning for a PPLTL goal requires reaching a certain state satisfying the PPLTL goal, that is, the state-trace produced to reach such a state satisfies the goal formula. In this paper, we study planning for PPLTL temporally extended goals in the context of both *Deterministic* (classical) and *Fully Observable Non-Deterministic* (FOND) domains. The standard approach for planning for PPLTL goals consists of computing the

* Corresponding author at: University of Oxford, Oxford, United Kingdom.
** Corresponding author.
E-mail addresses: luigi.bonassi@unibs.it (L. Bonassi), giuseppe.degiacomo@cs.ox.ac.uk (G. De Giacomo), marco.favorito@gmail.com (M. Favorito), francesco.fuggitti@gmail.com (F. Fuggitti), alfonso.gerevini@unibs.it (A.E. Gerevini), enrico.scala@unibs.it (E. Scala).
[1] Views and opinions expressed are of the author's own and are not representative of the Bank of Italy's official position.

https://doi.org/10.1016/j.artint.2025.104409
Received 5 November 2024; Received in revised form 27 August 2025; Accepted 27 August 2025

Available online 8 September 2025

cross-product between the deterministic automaton (DFA) for the planning domain and the DFA for the PPLTL goal formula, and then checking non-emptiness on the resulting automaton returning a plan if any [35,33]. Exploiting a property of reverse languages [29], the DFA corresponding to a PPLTL formula can be computed directly in *single* exponential time in the size of the formula. Therefore, classical planning for PPLTL goals is PSPACE-complete and FOND planning for PPLTL goals is EXPTIME-complete [33].

In this paper, we propose an approach for planning for PPLTL goals that bypasses the standard construction of the automaton of the goal formula. Our technique exploits the well-known fixpoint characterization [46,62,39] of temporal logic formulas, which recursively splits a PPLTL formula into a propositional formula on the *current* instant and a temporal formula to be checked at the *previous* instant. By monitoring the truth of specific subformulas of the PPLTL goal at the previous instant using a few additional variables, we can leverage the fixpoint characterization to evaluate the PPLTL goal based solely on the current state, without reasoning over the entire traces. We employ this technique to seamlessly encode classical planning for PPLTL goals into classical planning for *reachability* (i.e., final-state) goals. Notably, unlike existing polynomial encodings for LTL_f goals [74], our technique does not require any additional action, thereby preserving the size of solution plans. Interestingly, this approach can also be effortlessly extended to encode FOND planning for PPLTL goals into FOND planning for reachability goals.

We study the proposed encoding for PPLTL goals from both the theoretical and the practical perspectives. Theoretically, we can encode PPLTL goals into both classical and FOND planning linearly in the size of the goal formula, without introducing any spurious actions. In the context of classical planning, the state-of-the-art encodings for LTL_f goals are either exponential [13,14] in the size of the goal or significantly increase the length of solution plans [74]. In FOND planning, the only known encoding for LTL_f goals depends on explicit automata construction and is worst-case exponential in the size of the goal formula [27,26]. Practically, we have implemented our approach in a tool called Plan4Past, which can be used along with state-of-the-art classical and FOND planners. Moreover, our experimental analysis shows the effectiveness of Plan4Past by comparing it against state-of-the-art approaches for LTL_f goals over a set of benchmarks featuring both classical and FOND planning domains.

1.1. Contributions

This paper presents an approach to planning for temporally extended goals in PPLTL. We significantly extend and detail our previous works [20,21] by providing (*i*) a much more detailed presentation of the encoding technique, (*ii*) extended examples, (*iii*) full-spelled proofs, *(iv)* a presentation of our approach from an automata-theoretic perspective, *(v)* an extended experimental analysis with a deeper discussion and new empirical results, and *(vi)* an in-depth related work analysis comparing our approach with existing LTL_f encodings for classical and FOND planning.

1.2. Outline

The paper is organized as follows. Section 2 reviews the syntax and semantics of PPLTL, and provides some examples of compelling PPLTL formulas. Section 3 contains the theoretical foundations for handling PPLTL formulas; in particular, this section shows that any PPLTL formula can be evaluated state-by-state as a Markovian property by only adding a few propositional variables. Section 4 formalizes the problem of classical planning for PPLTL goals and shows how to exploit the theoretical results of Section 3 to encode classical planning for PPLTL goals into classical planning for reachability goals; the proposed encoding employs *axioms* and *derived predicates* [55,73] to elegantly and compactly capture the evaluation of the PPLTL goal formula in each generated planning state. Section 5 introduces FOND planning for PPLTL goals and shows that the encoding presented in Section 4 can be used as-is to encode PPLTL goals into FOND domains. Section 6 presents an alternative approach that explicitly encodes the relevant parts of the PPLTL goal formula into the domain to avoid introducing derived predicates, which only a few planners fully support. Section 7 reports our empirical analysis. Finally, Section 8 analyzes related work.

2. Pure-past linear temporal logic

Pure-Past Linear Temporal Logic (PPLTL), recently surveyed in [33], is the variant of LTL_f that refers about the past instead of the future. Here, we summarize its main characteristics and give some examples.

Given a set $\mathcal{P}$ of propositions, PPLTL is defined as:

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

where $p \in \mathcal{P}$, Y is the *yesterday* operator and S is the *since* operator. Intuitively, $\mathsf{Y}(\varphi)$ specifies that the formula φ must hold in the previous step, while $\varphi_1 \,\mathsf{S}\, \varphi_2$ is satisfied when φ_1 held since φ_2 became satisfied.

PPLTL formulas are interpreted on *finite-length nonempty* traces, also called *histories*, $\tau = s_0, \ldots, s_n$ with $n \geq 0$, where s_i at instant i is a propositional interpretation over the alphabet $2^{\mathcal{P}}$. We denote the length of τ by $\mathsf{length}(\tau) = n{+}1$ and the last element of τ by $\mathsf{last}(\tau) = s_n$. We define the satisfaction relation $\tau, i \vDash \varphi$, stating that φ holds at instant i, with $0 \leq i < \mathsf{length}(\tau)$, as follows:

- $\tau, i \vDash p$ iff $p \in s_i$ (for $p \in \mathcal{P}$);
- $\tau, i \vDash \neg\varphi$ iff $\tau, i \nvDash \varphi$;
- $\tau, i \vDash \varphi_1 \wedge \varphi_2$ iff $\tau, i \vDash \varphi_1$ and $\tau, i \vDash \varphi_2$;
- $\tau, i \vDash \mathsf{Y}\varphi$ iff $i \geq 1$ and $\tau, i-1 \vDash \varphi$;
- $\tau, i \vDash \varphi_1 \,\mathsf{S}\, \varphi_2$ iff there exists k, with $0 \leq k \leq i$ such that $\tau, k \vDash \varphi_2$ and for all j, with $k < j \leq i$, we have that $\tau, j \vDash \varphi_1$.

Table 1
PDDL3 operators and their equivalent PPLTL and LTL_f formulas. Superscripts abbreviate nested temporal operators. θ is a propositional formula on planning fluents; X, WX, F, U and G are the next, weak next, eventually, until and always operators of LTL_f; $\phi_1 \,\mathsf{R}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{U}\, \neg\phi_2)$, and end is the end of the trace in LTL_f [37]. * tags an operator with the corrected LTL_f translation.

PDDL3 Operator	PPLTL Formula	LTL_f Formula
at-end θ	θ	$\mathsf{F}(\theta \wedge \text{end})$
always θ	$\mathsf{H}\theta$	$\mathsf{G}\theta$
sometime θ	$\mathsf{O}\theta$	$\mathsf{F}\theta$
sometime-aft. $\theta_1\theta_2$	$(\neg\theta_1 \,\mathsf{S}\, \theta_2) \vee \mathsf{H}(\neg\theta_1)$	$\mathsf{G}(\theta_1 \rightarrow \mathsf{F}\theta_2)$
sometime-bef. $\theta_1\theta_2$	$\mathsf{H}(\theta_1 \rightarrow \mathsf{Y}(\mathsf{O}(\theta_2))$	$\theta_2 \,\mathsf{R}\, \neg\theta_1$
at-most-once θ	$\mathsf{H}(\theta \rightarrow (\theta \,\mathsf{S}\, (\mathsf{H}(\neg\theta) \vee \text{start})))$	$\mathsf{G}(\theta \rightarrow (\theta \,\mathsf{U}\, (\mathsf{G}(\neg\theta) \vee \text{end})))$
hold-during $n_1\ n_2\ \theta$	$\bigvee_{0\le i\le n_1}(\theta \wedge \mathsf{Y}^i(\text{start})) \vee$ $\bigwedge_{n_1<i\le n_2} \mathsf{H}(\theta \vee \mathsf{WY}^i(\mathsf{Y}\mathit{true}))$	$\bigvee_{0\le i\le n_1} \mathsf{X}^i(\theta \wedge \text{end}) \vee$ $\bigwedge_{n_1<i\le n_2} \mathsf{WX}^i(\theta)$
* hold-after $n\ \theta$	$\bigvee_{0\le i\le n}(\theta \wedge \mathsf{Y}^i(\text{start})) \vee$ $\mathsf{O}(\theta \wedge \mathsf{Y}^{n+1}(\mathsf{O}(\text{start})))$	$\bigvee_{0\le i\le n} \mathsf{X}^i(\theta \wedge \text{end}) \vee$ $\mathsf{X}^{n+1}(\mathsf{F}(\theta))$

In addition, we define the following common abbreviations: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, the *once* operator $\mathsf{O}\varphi \equiv \mathit{true}\,\mathsf{S}\,\varphi$, the *historically* operator $\mathsf{H}\varphi \equiv \neg\mathsf{O}\neg\varphi$, the *weak-yesterday* operator $\mathsf{WY}\phi \equiv \neg\mathsf{Y}\neg\phi$, and the propositional Boolean constants $\mathit{true} \equiv p \vee \neg p$, for any proposition p, and $\mathit{false} \equiv \neg\mathit{true}$. The semantics of these derived temporal PPLTL operators are the following:

- $\mathsf{O}(\varphi)$ requires that φ held at some point in the past. Formally, $\tau, i \vDash \mathsf{O}(\varphi)$ iff there exists k with $0 \le k \le i$, such that $\tau, k \vDash \varphi$;
- $\mathsf{H}(\varphi)$ is satisfied when φ always held in the past. Formally, $\tau, i \vDash \mathsf{H}(\varphi)$ iff for every k with $0 \le k \le i$, $\tau, k \vDash \varphi$;
- $\mathsf{WY}(\varphi)$ specifies that the formula φ must hold at the previous step, but only when the current step is not the first. Formally, $\tau, i \vDash \mathsf{WY}\varphi$ iff $i = 0$ or $\tau, i-1 \vDash \varphi$.

Also, formula $\text{start} \equiv \neg\mathsf{Y}(\mathit{true})$ expresses that the trace has started. A PPLTL formula φ is *true* in τ, denoted $\tau \vDash \varphi$, if $\tau, \text{length}(\tau)-1 \vDash \varphi$. We denote by $\text{sub}(\varphi)$ the set of all subformulas of φ obtained from the abstract syntax tree of φ [37]. For instance, if $\varphi = a \wedge \neg\mathsf{Y}(b \vee (c \vee d))$, where a, b, c, d are atomic, then $\text{sub}(\varphi) = \{a, b, c, d, (c \vee d), b \vee (c \vee d), \mathsf{Y}(b \vee (c \vee d)), \neg\mathsf{Y}(b \vee (c \vee d)), a \wedge \neg\mathsf{Y}(b \vee (c \vee d))\}$. Moreover, we define the size $|\varphi|$ of a PPLTL formula φ as the number of symbols of φ, while $|\text{sub}(\varphi)|$ denotes the number of subformulas of φ. It is important to note that $|\text{sub}(\varphi)|$ is $O(|\varphi|)$ in the worst case.

Several temporal properties can be compactly expressed in PPLTL.

Example 1. The goal "We are now at location l_1 and have passed through location l_2" in PPLTL is $l_1 \wedge \mathsf{O}(l_2)$. Another interesting property is "Every time I took the bus, I bought a new ticket beforehand", which translates as $\mathsf{H}(\mathit{Bus} \implies \mathsf{Y}(\neg\mathit{Bus}\,\mathsf{S}\,\mathit{Ticket}))$ [33].

In addition, some common PPLTL patterns have been employed in the context of MDP rewards in [5] or as norms in multi-agent systems [40,59,2]. Further examples appear in [33] and in [43].

Table 1 provides the translation to equivalent PPLTL formulas of PDDL3 patterns [49], while Table 2 reports the equivalent PPLTL formulation of DECLARE templates (Table 2), the *de-facto* standard modeling language for Business Processes [1]. Notably, a systematic translation between LTL_f and PPLTL (and vice versa) does exist, but it is impractical [33]. Hence, we formally proved the correctness of translations for both tables by checking equivalence in terms of languages, i.e., by translating them to DFA and solving graph isomorphism.

3. Handling pure-past linear temporal logic formulas

In AI planning, the objective is to achieve a certain condition in a final state. However, in this paper, we tackle the problem of achieving a PPLTL temporal formula, which is a *Non-Markovian* property. Normally, evaluating a PPLTL formula requires considering the whole history (i.e., the state trace) generated by the planning process, but searching for a trace is quite demanding. Instead, in this section, we propose an approach to evaluate a PPLTL formula state-by-state, without considering traces at all. Given a trace $\tau = s_0, s_1, \ldots, s_n$ and a PPLTL formula φ, our technique works as follows. We record the truth of a specific set of subformulas of φ at every instant of τ with a new set of propositional variables denoted with Σ_φ. Then, we properly combine these new atoms with the original set of atoms $\mathcal{P}$ to determine the truth of φ state-by-state.

To determine which subformulas we need to consider, we exploit the observation that we can put a PPLTL formula in a form where its evaluation depends only on the current state and the evaluation of some PPLTL subformulas at the previous instant. In particular, similarly to LTL and LTL_f, PPLTL formulas can be decomposed into present and past components, given the fixpoint characterization of the since operator[2]:

[2] We remind the reader that the other PPLTL operators such as O and H can be expressed in terms of the S operator.

Table 2
DECLARE templates, their equivalent PPLTL and LTL_f formulas. a, b are atomic propositions.

DECLARE Template	Equiv. PPLTL Formula	Equiv. LTL_f Formula
init(a)	$\mathsf{O}(a \wedge \neg\mathsf{Y}(\mathit{true}))$	a
existence(a)	$\mathsf{O}(a)$	$\mathsf{F}(a)$
absence(a)	$\neg\mathsf{O}(a)$	$\neg\mathsf{F}(a)$
absence2(a)	$\mathsf{H}(a \rightarrow \mathsf{WYH}(\neg a))$	$\neg(\mathsf{F}a \wedge \mathsf{XF}(a))$
choice(a, b)	$\mathsf{O}(a) \vee \mathsf{O}(b)$	$\mathsf{F}(a) \vee \mathsf{F}(b)$
exclusive-choice(a, b)	$(\mathsf{O}(a) \vee \mathsf{O}(b)) \wedge \neg(\mathsf{O}(a) \wedge \mathsf{O}(b))$	$(\mathsf{F}(a) \vee \mathsf{F}(b)) \wedge \neg(\mathsf{F}(a) \wedge \mathsf{F}(b))$
co-existence(a, b)	$\mathsf{H}(\neg a) \leftrightarrow \mathsf{H}(\neg b)$	$\mathsf{F}(a) \leftrightarrow \mathsf{F}(b)$
responded-existence(a, b)	$\mathsf{O}(a) \rightarrow \mathsf{O}(b)$	$\mathsf{F}(a) \rightarrow \mathsf{F}(b)$
response(a, b)	$(\neg a\,\mathsf{S}\,b) \vee \mathsf{H}(\neg a)$	$\mathsf{G}(a \rightarrow \mathsf{F}(b))$
precedence(a, b)	$\mathsf{H}(b \rightarrow \mathsf{O}(a))$	$(\neg b\,\mathsf{U}\,a) \vee \mathsf{G}(\neg b)$
succession(a, b)	response(a, b) $\wedge$ precedence(a, b)	
chain-response(a, b)	$\mathsf{H}(\mathsf{Y}(a) \rightarrow b) \wedge \neg a$	$\mathsf{G}(a \rightarrow \mathsf{X}(b))$
chain-precedence(a, b)	$\mathsf{H}(b \rightarrow \mathsf{Y}(a))$	$\mathsf{G}(\mathsf{X}(b) \rightarrow a) \wedge \neg b$
chain-succession(a, b)	$(\mathsf{H}(\mathsf{Y}(a) \rightarrow b) \wedge \neg a) \wedge \mathsf{H}(\mathsf{Y}(\neg a) \rightarrow \neg b)$	$\mathsf{G}(a \leftrightarrow \mathsf{X}(b))$
not-co-existence(a, b)	$\mathsf{O}(a) \rightarrow \neg\mathsf{O}(b)$	$\mathsf{F}(a) \rightarrow \neg\mathsf{F}(b)$
not-succession(a, b)	$\mathsf{H}(b \rightarrow \neg\mathsf{O}(a))$	$\mathsf{G}(a \rightarrow \neg\mathsf{F}(b))$
not-chain-succession(a, b)	$\mathsf{H}(b \rightarrow \neg\mathsf{Y}(a))$	$\mathsf{G}(a \rightarrow \neg\mathsf{X}(b))$

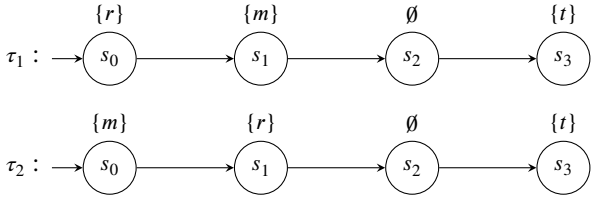
$$\phi_1\,\mathsf{S}\,\phi_2 \;\equiv\; \phi_2 \vee (\phi_1 \wedge \mathsf{Y}(\phi_1\,\mathsf{S}\,\phi_2)).$$

Exploiting this equivalence, the formula decomposition can be computed by recursively applying the following transformation function $\mathsf{pnf}(\cdot)$:

- $\mathsf{pnf}(a) = a$;
- $\mathsf{pnf}(\mathsf{Y}\phi) = \mathsf{Y}\phi$;
- $\mathsf{pnf}(\phi_1\,\mathsf{S}\,\phi_2) = \mathsf{pnf}(\phi_2) \vee (\mathsf{pnf}(\phi_1) \wedge \mathsf{Y}(\phi_1\,\mathsf{S}\,\phi_2))$;
- $\mathsf{pnf}(\phi_1 \wedge \phi_2) = \mathsf{pnf}(\phi_1) \wedge \mathsf{pnf}(\phi_2)$;
- $\mathsf{pnf}(\neg\phi) = \neg\mathsf{pnf}(\phi)$.

A formula resulting from applying $\mathsf{pnf}(\cdot)$ is in Previous Normal Form (PNF). Note that formulas in PNF have proper temporal subformulas (i.e., subformulas whose main construct is a temporal operator) appearing only in the scope of the Y operator.

Example 2. As a running example, we use the formula $\varphi = t \wedge \neg r\,\mathsf{S}\,m$, which models a situation in which we have an agent that must finish the task t without using the resource r since activating the machine m. For example, consider the two traces τ_1 and τ_2 defined as follows:



As usual, we compactly represent a propositional interpretation as the set of atoms that are true in it. We have that $\tau \vDash \varphi$, while $\tau' \nvDash \varphi$. The PNF of φ is:

$$\begin{aligned}\mathsf{pnf}(t \wedge \neg r\,\mathsf{S}\,m) &= \mathsf{pnf}(t) \wedge \mathsf{pnf}(\neg r\,\mathsf{S}\,m)\\ &= t \wedge (\mathsf{pnf}(m) \vee (\mathsf{pnf}(\neg r) \wedge \mathsf{Y}(\neg r\,\mathsf{S}\,m)))\\ &= t \wedge (m \vee (\neg r \wedge \mathsf{Y}(\neg r\,\mathsf{S}\,m))).\end{aligned}$$

Note that all (proper) temporal subformulas of φ are enclosed within the Y operator.

Also, observe that the formulas of the form $\mathsf{Y}\phi$ in $\mathsf{pnf}(\varphi)$ are such that $\phi \in \mathsf{sub}(\varphi)$. It is easy to see that the following holds.

Proposition 1. *Every PPLTL formula φ can be converted to its PNF form $\mathsf{pnf}(\varphi)$ in linear-time in $|\mathsf{sub}(\varphi)|$. Moreover, $\mathsf{pnf}(\varphi)$ is equivalent to φ.*

Proof. Immediate from the definition of $\mathsf{pnf}(\cdot)$ and the semantics of PPLTL formulas. □

Notably, the PNF transformation allows us to determine the truth value of a PPLTL formula by knowing the truth of some atomic propositions at the current instant and the truth of some key subformulas at the previous instant. In particular, the key subformulas that we need to monitor are those appearing within the Y-scope in the PNF. Therefore, we keep track of the truth of these specific subformulas using a set Σ_φ of new atomic propositions of the form "$\mathsf{Y}\phi$". These propositions are:

1. "$\mathsf{Y}\phi$" for each subformula of φ of the form $\mathsf{Y}\phi$;
2. "$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$" for each subformula of φ of the form $\phi_1 \,\mathsf{S}\, \phi_2$.

Example 3. For $\varphi = t \wedge \neg r \,\mathsf{S}\, m$ we have $\Sigma_\varphi = \{\text{"}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{"}\}$. Therefore, to evaluate φ, we only need to keep track of the subformula $\mathsf{Y}(\neg r \,\mathsf{S}\, m)$ using the propositional variable "$\mathsf{Y}(\neg r \,\mathsf{S}\, m)$".

To evaluate the truth value of propositions in Σ_φ, we introduce the interpretation:

$$\sigma : \Sigma_\varphi \to \{\top, \bot\}.$$

Moreover, we denote with σ_i an interpretation over Σ_φ at an instant i. Intuitively, we can use the interpretation s_i of the original propositional symbols and the current interpretation σ_i of the "$\mathsf{Y}(\phi)$" atoms to determine the truth of φ by evaluating the (propositional) formula obtained by swapping every $\mathsf{Y}(\phi)$ with "$\mathsf{Y}(\phi)$" in $\mathsf{pnf}(\varphi)$. The next definition formalizes how to do so à la dynamic programming by introducing a predicate denoted with val.

Definition 1. Let s_i be a propositional interpretation over $\mathcal{P}$, σ_i a propositional interpretation over Σ_φ, and ϕ a PPLTL subformula in $\mathsf{sub}(\varphi)$, we define the predicate $\mathsf{val}(\phi, \sigma_i, s_i)$, recursively as follows:

- $\mathsf{val}(a, \sigma_i, s_i)$ iff $s_i \vDash a$;
- $\mathsf{val}(\mathsf{Y}\phi', \sigma_i, s_i)$ iff $\sigma_i \vDash \text{"}\mathsf{Y}\phi'\text{"}$;
- $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \sigma_i, s_i)$ iff $\mathsf{val}(\phi_2, \sigma_i, s_i) \vee (\mathsf{val}(\phi_1, \sigma_i, s_i) \wedge \sigma_i \vDash \text{"}\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)\text{"})$;
- $\mathsf{val}(\phi_1 \wedge \phi_2, \sigma_i, s_i)$ iff $\mathsf{val}(\phi_1, \sigma_i, s_i) \wedge \mathsf{val}(\phi_2, \sigma_i, s_i)$;
- $\mathsf{val}(\neg\phi', \sigma_i, s_i)$ iff $\neg\mathsf{val}(\phi', \sigma_i, s_i)$.

Note that the '*iff*' in Definition 1 states when the predicate $\mathsf{val}(\phi, \sigma_i, s_i)$ is true or false, for a formula ϕ, depending on σ_i and s_i. Moreover, $\mathsf{val}(\phi, \sigma_i, s_i)$ basically follows the $\mathsf{pnf}(\cdot)$ transformation rules, where the $\mathsf{Y}(\phi)$ subformulas are swapped with their propositional "$\mathsf{Y}(\phi)$" counterparts.

Example 4. We have that $\mathsf{val}(t \wedge \neg r \,\mathsf{S}\, m, \sigma_i, s_i)$ is defined as:

$$\begin{aligned}
&\mathsf{val}(t \wedge \neg r \,\mathsf{S}\, m, \sigma_i, s_i) \text{ iff } \mathsf{val}(t, \sigma_i, s_i) \wedge \mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_i, s_i)\\
&\mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_i, s_i) \text{ iff } \mathsf{val}(m, \sigma_i, s_i) \vee (\mathsf{val}(\neg r, \sigma_i, s_i) \wedge \sigma_i \vDash \text{"}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{"})\\
&\mathsf{val}(\neg r, \sigma_i, s_i) \text{ iff } \neg\mathsf{val}(r, \sigma_i, s_i)\\
&\mathsf{val}(r, \sigma_i, s_i) \text{ iff } s_i \vDash r\\
&\mathsf{val}(m, \sigma_i, s_i) \text{ iff } s_i \vDash m\\
&\mathsf{val}(t, \sigma_i, s_i) \text{ iff } s_i \vDash t.
\end{aligned}$$

By analyzing the above rules, it is easy to see that $\mathsf{val}(t \wedge \neg r \,\mathsf{S}\, m, \sigma_i, s_i)$ holds iff $(\sigma_i, s_i) \vDash t \wedge (m \vee (\neg r \wedge \text{"}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{"}))$. Observe that the formula $t \wedge (m \vee (\neg r \wedge \text{"}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{"}))$ can be obtained by swapping $\mathsf{Y}(\neg r \,\mathsf{S}\, m)$ with "$\mathsf{Y}(\neg r \,\mathsf{S}\, m)$" in $\mathsf{pnf}(\varphi)$.

It is important to note that we can conveniently use $\mathsf{val}(\phi, \sigma_i, s_i)$ to determine the truth of any subformula $\phi \in \mathsf{sub}(\varphi)$. Clearly, evaluating $\mathsf{val}(\phi, \sigma_i, s_i)$ requires the interpretation σ_i. Luckily, we can exploit the semantics of PPLTL to recursively determine σ_i at every step i of a trace. In particular, we exploit the two following observations:

- Every formula of the form $\mathsf{Y}\phi$ is false at the beginning of the trace (i.e., when $i = 0$), meaning that all "$\mathsf{Y}\phi$" propositions must also be false. In other words, σ_0 is known a priori;
- Every formula of the form $\mathsf{Y}\phi$ is true at step i iff ϕ is true at step $i-1$. Therefore, "$\mathsf{Y}\phi$" must hold in σ_i iff ϕ was true at $i-1$. Since $\mathsf{val}(\phi, \sigma_{i-1}, s_{i-1})$ captures the truth of ϕ at $i-1$, we have that "$\mathsf{Y}\phi$" must hold in σ_i iff $\mathsf{val}(\phi, \sigma_{i-1}, s_{i-1})$ holds.

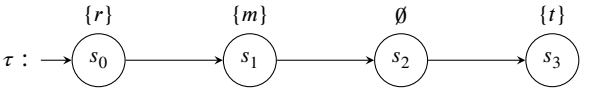
Following these observations, we have that σ_i can be recursively computed as follows:

$$\sigma_0(\text{"}\mathsf{Y}\phi\text{"}) \doteq \bot \text{ for each "}\mathsf{Y}\phi\text{"} \in \Sigma_\varphi \tag{1}$$

$$\sigma_i(\text{"}\mathsf{Y}\phi\text{"}) \doteq \mathsf{val}(\phi, \sigma_{i-1}, s_{i-1}) \text{ for each "}\mathsf{Y}\phi\text{"} \in \Sigma_\varphi. \tag{2}$$

Given a trace $\tau = s_0, s_1, \ldots, s_n$, we denote by $\tau^{[\varphi]} = \sigma_0, \sigma_1, \ldots, \sigma_n$ the corresponding trace of interpretations over Σ_φ.

Example 5. Consider the formula $\varphi = t \wedge (\neg r \,\mathsf{S}\, m)$ and the trace:



We can compute the interpretation σ_i of the variables $\Sigma_\varphi = \{\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}\}$ at every instant i by using the following assignments:

1. $\sigma_0(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \bot$;
2. $\sigma_1(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_0, s_0)$;
3. $\sigma_2(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_1, s_1)$;
4. $\sigma_3(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_2, s_2)$.

Recall that $\mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_i, s_i)$ iff $(\sigma_i, s_i) \vDash t \wedge (m \vee (\neg r \wedge \text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}))$. Hence, we have that:

1. $\mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_0, s_0)$ is false, therefore $\sigma_1(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \bot$;
2. $\mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_1, s_1)$ is true, therefore $\sigma_2(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \top$;
3. $\mathsf{val}(\neg r \,\mathsf{S}\, m, \sigma_2, s_2)$ is true, therefore $\sigma_3(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \top$.

Now, we formally prove that for every $\phi \in \mathsf{sub}(\varphi)$, $\mathsf{val}(\phi, \sigma_i, s_i)$ holds iff so does ϕ at instant i of a trace.

Theorem 1. *Let φ be a PPLTL formula over $\mathcal{P}$, $\phi \in \mathsf{sub}(\varphi)$ a subformula of φ, τ a trace over $\mathcal{P}$, and $\tau^{[\varphi]}$ the corresponding trace over Σ_φ. Then,*

$$\tau \vDash \phi \;\; \textit{iff} \;\; \mathsf{val}(\phi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau)).$$

Proof. We prove the thesis by double induction on the length of the trace τ and on the structure of the formula ϕ.

- Base case: $\tau = s_0$. We show that $s_0 \vDash \phi$ iff $\mathsf{val}(\phi, \sigma_0, s_0)$ by structural induction on the formula ϕ.
  - $\phi = p$. By definition of $\mathsf{val}(\cdot)$, $\mathsf{val}(p, \sigma_0, s_0)$ iff $s_0 \vDash p$.
  - $\phi = \mathsf{Y}\phi'$. By definition of σ_0, $\sigma_0(\text{“}\mathsf{Y}\phi'\text{”}) = \bot$, and by the semantics, $s_0 \nvDash \mathsf{Y}\phi'$. Therefore, the thesis holds.
  - $\phi = \phi_1 \,\mathsf{S}\, \phi_2$. $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \sigma_i, s_i)$ iff $\mathsf{val}(\phi_2, \sigma_i, s_i) \vee (\mathsf{val}(\phi_1, \sigma_i, s_i) \wedge \sigma_i \vDash \text{“}\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)\text{”})$. By definition of σ_0, $\sigma_0(\text{“}\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2\text{”})) = \bot$, hence the formula above simplifies to $\mathsf{val}(\phi_2, \sigma_i, s_i)$. On the other hand, by the semantics, $s_0 \vDash \phi_1 \,\mathsf{S}\, \phi_2$ iff $s_0 \vDash \phi_2$. Hence, by induction the thesis holds.
  - $\phi = \phi_1 \wedge \phi_2$ or $\phi = \neg\phi'$. The thesis holds by structural induction.
- Inductive step: Let $\tau = \tau_{n-1} \cdot s_n$. By inductive hypothesis, the thesis holds for the trace τ_{n-1} of length $n-1$:

  $$\tau_{n-1} \vDash \phi \;\text{ iff }\; \mathsf{val}(\phi, \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$$

  Now, we prove that the thesis holds also for $\tau_{n-1} \cdot s_n$:

  $$\tau_{n-1} \cdot s_n \vDash \phi \;\text{ iff }\; \mathsf{val}(\phi, \mathsf{last}((\tau_{n-1} \cdot s_n)^{[\varphi]}), \mathsf{last}(\tau_{n-1} \cdot s_n)).$$

  To prove the claim, we now proceed by structural induction on the formula, knowing that $\mathsf{last}((\tau_{n-1} \cdot s_n)^{[\varphi]}) = \sigma_n$ and $\mathsf{last}(\tau_{n-1} \cdot s_n) = s_n$:
  - $\phi = p$. We have that $\tau_{n-1} \cdot s_n \vDash p$ iff $s_n \vDash p$. For the $\mathsf{val}(\cdot)$ predicate we have that $s_n \vDash p$ iff $\mathsf{val}(p, \sigma_n, s_n)$. Therefore, the thesis holds.
  - $\phi = \mathsf{Y}\phi'$. We have that $\mathsf{last}(\tau_{n-1} \cdot s_n) \vDash \mathsf{Y}\phi'$ iff $\mathsf{last}(\tau_{n-1}) \vDash \phi'$. By inductive hypothesis, $\tau_{n-1} \vDash \phi'$ iff $\mathsf{val}(\phi', \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$. For the $\mathsf{val}(\cdot)$ predicate $\mathsf{val}(\mathsf{Y}\phi', \sigma_n, s_n)$ iff $\sigma_n \vDash \text{“}\mathsf{Y}\phi'\text{”}$, which in turn is defined as $\mathsf{val}(\phi', \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$. Hence the thesis holds.
  - $\phi = \phi_1 \,\mathsf{S}\, \phi_2$. In this case it suffices to remember that $s_n \vDash \phi_1 \,\mathsf{S}\, \phi_2$ iff $s_n \vDash \phi_2 \vee (\phi_1 \wedge \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2))$. On the other hand, $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \sigma_n, s_n)$ iff $\mathsf{val}(\phi_2, \sigma_n, s_n) \vee (\mathsf{val}(\phi_1, \sigma_n, s_n) \wedge \sigma_n \vDash \text{“}\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)\text{”})$. By structural induction we have that $s_n \vDash \phi_1$ iff $\mathsf{val}(\phi_1, \sigma_n, s_n)$, and $s_n \vDash \phi_2$ iff $\mathsf{val}(\phi_2, \sigma_n, s_n)$. Moreover $s_n \vDash \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$ iff $\mathsf{last}(\tau_{n-1}) \vDash \phi_1 \,\mathsf{S}\, \phi_2$, and $\sigma_n \vDash \text{“}\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)\text{”}$ iff $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$. Finally, $\mathsf{last}(\tau_{n-1}) \vDash \phi_1 \,\mathsf{S}\, \phi_2$ iff $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$ holds by induction on the length of the trace.
  - $\phi = \phi_1 \wedge \phi_2$ or $\phi = \neg\phi'$. The thesis holds by structural induction. □

Theorem 1 gives us the basis of our technique. Specifically, it guarantees that by keeping a suitably updated interpretation σ of Σ_φ variables, we can evaluate our PPLTL goal at any instant without reasoning over traces.

3.1. Automata-theoretic perspective

We now show how the theoretical approach presented in this section can be used to construct a *symbolic* DFA whose size is linear in $|\varphi|$ and that accepts a trace τ iff $\tau \vDash \varphi$. We start by defining a symbolic DFA.

Definition 2. A symbolic DFA is a tuple $\mathcal{A} = \langle \mathcal{P}, Q, q_I, \delta, F \rangle$, where:

- $\mathcal{P}$, known as the alphabet, is a set of boolean propositions;
- Q is a set of boolean propositions representing the set of possible automaton's states;
- q_I is an interpretation over Q representing the initial state;
- $\delta : 2^Q \times 2^{\mathcal{P}} \to 2^Q$ is a transition function mapping an interpretation s over $\mathcal{P}$ and an interpretation q over Q into a successor state $q' = \delta(q, s)$ such that q' is an interpretation of Q;
- $F : 2^Q \to \{\top, \bot\}$ is a function mapping a state q into $\top$ or $\bot$ representing the acceptance condition.

Essentially, a symbolic DFA can be seen as a generalization of an Alternating Automaton [29] with a deterministic transition function. Moreover, a symbolic DFA $\mathcal{A}$ can have up to $2^{|Q|}$ different states, and a run of $\mathcal{A}$ is a sequence of interpretations over Q.

Definition 3. Let $\mathcal{A} = \langle \mathcal{P}, Q, q_I, \delta, F \rangle$ be a symbolic automaton, and let $\tau = s_0, \dots, s_n$ be a sequence of interpretations over the alphabet $\mathcal{P}$. A run of $\mathcal{A}$ on trace τ is a sequence of states $\tau_Q = q_0, q_1, \dots, q_{n+1}$, such that $q_0 = q_I$, and $q_i = \delta(q_{i-1}, s_{i-1})$ for every $i = 1, \dots, n+1$. $\mathcal{A}$ accepts τ iff the run on τ is accepting. The run on τ is accepting iff $F(q_{n+1}) = \top$.

Instead, a standard (explicit) DFA $\mathcal{A}$ is defined as a symbolic DFA, with the exception that each of its states is a single element of Q.

We now show how to build a symbolic DFA for a PPLTL formula φ, denoted by $\mathcal{A}_\varphi$. Intuitively, the states q of $\mathcal{A}_\varphi$ are symbols in Σ_φ, and the initial state is q_0 as defined in Equation (1). Then, the transition function of $\mathcal{A}_\varphi$ takes as input a state q and an interpretation s over the alphabet $2^{\mathcal{P}}$ and returns a successor state q', which, analogously to Equation (2), is defined according to $\mathsf{val}(\cdot)$. In addition, the automaton states contain an additional atomic proposition, called final, which keeps track of whether φ has been satisfied, and is used as the accepting condition of the automaton.

Definition 4. Let φ be a PPLTL formula defined over a set of propositions $\mathcal{P}$. The symbolic DFA associated with φ is $\mathcal{A}_\varphi = \langle \mathcal{P}, Q, q_I, \delta, F \rangle$, where:

- $Q = \Sigma_\varphi \cup \{\mathsf{final}\}$;
- q_I is defined as $q_I(\text{“Y}\phi\text{”}) \doteq \bot$ for each $\text{“Y}\phi\text{”} \in \Sigma_\varphi$ and $q_I(\mathsf{final}) \doteq \bot$;
- δ is a function defined as $\delta(q, s) = q'$, where:

$$q'(\text{“Y}\phi\text{”}) \doteq \mathsf{val}(\phi, q, s) \text{ for each } \text{“Y}\phi\text{”} \in \Sigma_\varphi$$

$$q'(\mathsf{final}) \doteq \mathsf{val}(\varphi, q, s).$$

- F is a function defined as $F(q) = \top$ iff $q(\mathsf{final}) = \top$.

It is easy to see that, for any trace τ and for any PPLTL formula φ, $\tau \vDash \varphi$ iff $\mathcal{A}_\varphi$ accepts τ.

Theorem 2. *Let φ be a PPLTL formula over $\mathcal{P}$, and τ a trace of length n over $\mathcal{P}$. Then, $\tau \vDash \varphi$ iff $\mathcal{A}_\varphi$ accepts τ.*

Proof. We show that the claim holds by proving that the run of $\mathcal{A}_\varphi$ on τ is a sequence of states $\tau_Q = q_0, q_1, \dots, q_n, q_{n+1}$ such that, for every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$ and for every $i = 1, \dots, n$, $q_0(\text{“Y}\phi\text{”}) = \sigma_0(\text{“Y}\phi\text{”})$ and $q_i(\text{“Y}\phi\text{”}) = \sigma_i(\text{“Y}\phi\text{”})$ where σ_0 satisfies Equation (1) and σ_i satisfies Equation (2). By induction on the length n of the trace.

- Base case: $n = 0$. By Definition 4, $q_0(\text{“Y}\phi\text{”}) = \sigma_0(\text{“Y}\phi\text{”})$ for every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$ and the thesis holds;
- Inductive step: by inductive hypothesis, $q_{n-1}(\text{“Y}\phi\text{”}) = \sigma_{n-1}(\text{“Y}\phi\text{”})$ for every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$. By Equation (2), every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$ holds in σ_n iff $\mathsf{val}(\phi, s_{n-1}, \sigma_{n-1})$. By Definition 4, every $\text{“Y}\phi\text{”} \in Q$ also holds in q_n iff $\mathsf{val}(\phi, s_{n-1}, \sigma_{n-1})$. Therefore, by induction hypothesis, we have $q_n(\text{“Y}\phi\text{”}) = \sigma_n(\text{“Y}\phi\text{”})$ for every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$.

By Theorem 1, we have $\tau \vDash \varphi$ iff $\mathsf{val}(\varphi, \sigma_n, s_n)$. Hence, as $q_{n+1}(\mathsf{final})$ holds iff $\mathsf{val}(\varphi, q_n, s_n)$, and $q_n(\text{“Y}\phi\text{”}) = \sigma_n(\text{“Y}\phi\text{”})$ for every $\text{“Y}\phi\text{”} \in \Sigma_\varphi$, we have $F(q_{n+1}) = \top$ iff $\mathsf{val}(\varphi, \sigma_n, s_n)$ and therefore $\tau \vDash \varphi$ iff $\mathcal{A}_\varphi$ accepts φ. □

Example 6 provides an example of a symbolic DFA obtained following Definition 4 along with a graphical representation of the associated explicit DFA.

Example 6. Consider the formula $\varphi = t \wedge (\neg r \,\mathsf{S}\, m)$. The symbolic DFA $\mathcal{A}_\varphi$ is $\mathcal{A}_\varphi = \langle \mathcal{P}, Q, q_I, \delta, F \rangle$, where:

- $\mathcal{P} = \{t, r, m\}$;
- $Q = \{\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}, \mathsf{final}\}$;
- q_I is defined as $q_I(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \bot$ and $q_I(\mathsf{final}) \doteq \bot$;
- $\delta(q, s) = q'$ where:
  - $q'(\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}) \doteq \mathsf{val}(\neg r \,\mathsf{S}\, m, q, s)$;
  - $q'(\mathsf{final}) \doteq \mathsf{val}(t \wedge (\neg r \,\mathsf{S}\, m), q, s)$.

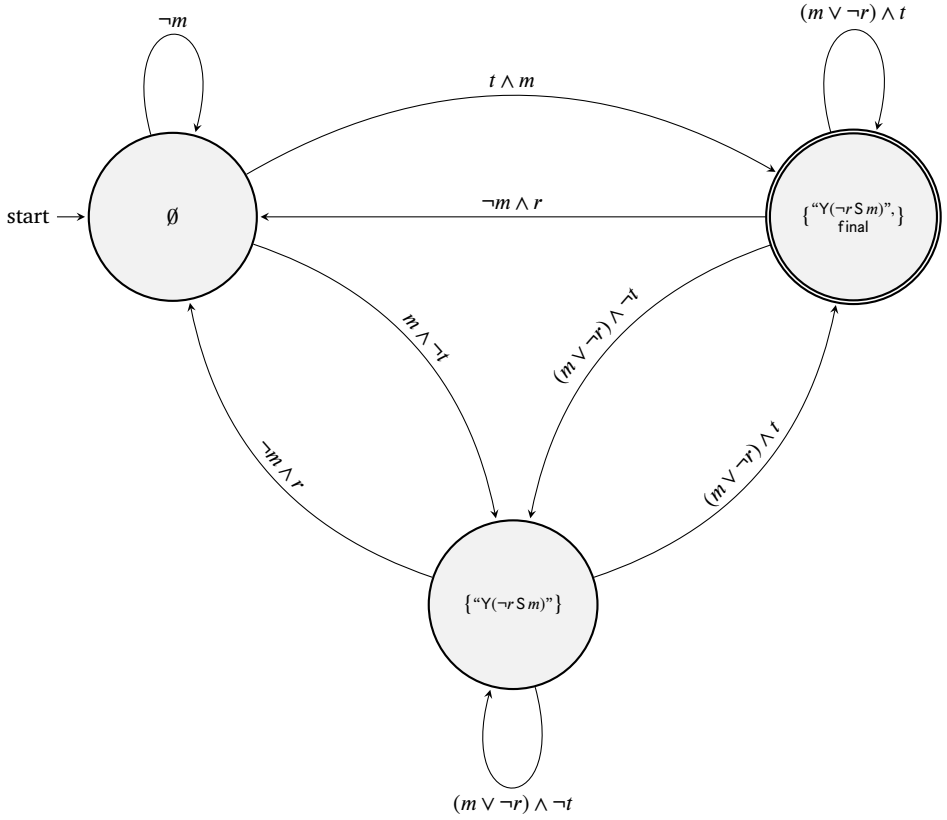
  Recall from Example 5 that:

$$\mathsf{val}(\neg r \,\mathsf{S}\, m, q, s) \text{ iff } (q, s) \vDash m \vee (\neg r \wedge \text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”})$$

$$\mathsf{val}(t \wedge (\neg r \,\mathsf{S}\, m), q, s) \text{ iff } (q, s) \vDash t \wedge (m \vee (\neg r \wedge \text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”})).$$

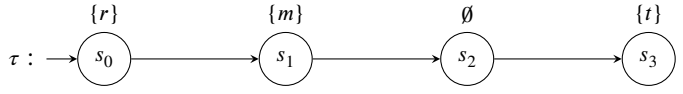
- $F(q) = \top$ iff $q(\mathsf{final}) = \top$.

We can build the explicit DFA $\mathcal{A}_\varphi$ for $t \wedge (\neg r \,\mathsf{S}\, m)$ by enumerating all possible states (that is, interpretations over Σ_φ) and all possible transitions from every state and for each propositional interpretation over $\mathcal{P}$. Clearly, in general, the explicit DFA can be exponentially larger than the symbolic DFA. The possible states are $\emptyset$, $\{\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}\}$, and $\{\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}, \mathsf{final}\}$. This is because, according to the definition of $\mathcal{A}_\varphi$, the state where only final holds cannot be reached. The resulting DFA is as follows.

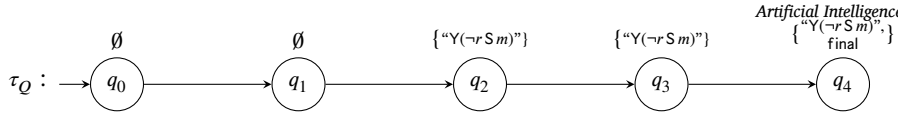


Note that we write each transition from q to q' as a propositional formula representing the set of interpretations over $\mathcal{P}$ that enable the transition from q to q'. The accepting state, that is, $\{\text{“}\mathsf{Y}(\neg r \,\mathsf{S}\, m)\text{”}, \mathsf{final}\}$, is marked with a double circle.

Consider the input trace τ of Example 5:



The run of the automaton $\mathcal{A}_\varphi$ on τ is:



The run is accepting, as $F(q_4)$ holds true since $q_4(\texttt{final}) = \top$.

Before concluding this section we want to make a remark that is relevant for the following. In Definition 4, we use the atomic proposition final to keep track of the accepting state of the symbolic automaton. Although this provides a standard acceptance condition, it is important to note that we can also check the acceptance of the automaton without the additional final proposition. For any trace $\tau = s_0, \dots, s_n$ over the alphabet $\mathcal{P}$, we can simply check whether the second-last state $q_n = \delta(q_{n-1}, s_{n-1})$ and s_n satisfy $\mathsf{val}(\varphi, \sigma_n, s_n)$. It is easy to see that, when $\mathsf{val}(\varphi, \sigma_n, s_n)$ holds, then by Definition 4 the DFA $\mathcal{A}_\varphi$ will always transition to an accepting state. This approach checks the acceptance condition on the *last transition*, while the automaton in Definition 4 accepts based on the last state. This distinction is analogous to the difference between *Mealy* and *Moore* machines [57]. In the next sections, we will implicitly use this accepting condition on the transitions to conveniently handle PPLTL goals in planning.

4. Classical planning for pure-past linear temporal logic goals

Automated Planning represents the model-based approach to producing autonomous behavior where the agent behavior is automatically derived from a model of the world state, actions, sensors, and goals [50,48]. Several configurations of planning have been studied over the years. In this section, we study the problem of deterministic (also known as *classical*) planning for temporally extended goals expressed in PPLTL. Specifically, by leveraging the theoretical results shown in Theorem 1, we demonstrate that PPLTL goals can be compiled away with minimal additional complexity. This can be done by introducing a set of new fluents that is at most linear in the size of the PPLTL goal and without adding any spurious action to the model.

This section is structured as follows; we first provide the preliminary notions of classical planning, that is, planning with deterministic actions and a reachability goal. We then extend the classical planning model with the notion of PPLTL goals. Finally, we show how to encode classical planning for PPLTL goals into classical planning, we formally prove the correctness of the approach, and we show that the resulting encoded problem is polynomial in the size of the original domain and the size of the PPLTL goal formula.

4.1. Classical planning

A planning domain model is a tuple $D = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff} \rangle$ where $\mathcal{F}$ is a set of fluents, $\mathcal{F}_{der}$ is a set of derived predicates, $\mathcal{X}$ is a set of axioms, A is a set of action labels, pre and eff are two functions that define the preconditions and effects of each action $a \in A$.[3]

A planning state s is a subset of $\mathcal{F}$, and a positive literal f holds true in s if $f \in s$; otherwise, f is false in s. Axioms have the form $d \leftarrow \psi$ where $d \in \mathcal{F}_{der}$ and ψ is a propositional formula over $\mathcal{F} \cup \mathcal{F}_{der}$. An axiom $d \leftarrow \psi$ specifies that d is derived to be true from a state s if and only if we can prove that $s \models \psi$, possibly using other axioms from $\mathcal{X}$. We assume that the set of axioms $\mathcal{X}$ is *stratified* [55] – this guarantees that given a state s and a derived predicate d, it is possible to uniquely determine whether d holds true in s. Thus, it is always possible to determine whether a formula ψ over $\mathcal{F} \cup \mathcal{F}_{der}$ is satisfied by a state s. Both functions pre and eff take an action label $a \in A$ as input and return a propositional formula over $\mathcal{F} \cup \mathcal{F}_{der}$ and a set of conditional effects, respectively. A conditional effect is a pair $c \rhd e$, where c is a propositional formula over $\mathcal{F} \cup \mathcal{F}_{der}$ and $e \subseteq \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$ is a set of literals from $\mathcal{F}$. With e^- and e^+, we indicate the partition of e that features only negative and positive literals, respectively.

An action a can be applied in a state s if $pre(a)$ holds true in s, formally, if $s \models pre(a)$. A conditional effect $c \rhd e$ is triggered in a state s if c is true in s. Applying a in s yields a successor state s' where $\forall f \in F$, f holds true in s' if and only if either:

1. f was true in s and no conditional effect $c \rhd e$ triggered in s deletes it ($\neg f \in e^-$).
2. there is a conditional effect $c \rhd e$ triggered in s that adds it ($f \in e^+$).

More formally, the new state s' is defined as follows:

$$s' = (s \setminus \bigcup_{\substack{c \rhd e \in \mathsf{eff}(a) \\ \text{with } s \models c}} e^-) \cup \bigcup_{\substack{c \rhd e \in \mathsf{eff}(a) \\ \text{with } s \models c}} e^+.$$

Note that, in case of conflicting effects, we define the successor state s' using the delete-before-adding semantics [70]. That is, s' is obtained by deleting all literals in e^- before adding all positive literals in e^+. We denote with $tr(a, s)$ the successor state resulting from applying a in s.

A classical planning problem combines a domain with an initial state and a goal, and consists of looking for a sequence of actions that transforms the initial state into a desired goal state. Formally, a planning problem is a tuple $\Gamma = \langle D, s_0, G \rangle$, where D is the domain model, s_0 is the initial state, i.e., an initial assignment to fluents in $\mathcal{F}$, and G is a formula over $\mathcal{F}$ called the reachability goal.

[3] Derived predicates and axioms are a standard extension of the original classical STRIPS-like planning [55].

Table 3
Components of the compiled classical planning problem $\Gamma' = \langle \langle \mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, eff' \rangle, s'_0, G' \rangle$ for a given planning problem $\Gamma = \langle \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, eff \rangle, s_0, G \rangle$.

Components	Encoding
Fluents $\mathcal{F}'$	$\mathcal{F}' := \mathcal{F} \cup \Sigma_\varphi$
Derived Predicates $\mathcal{F}'_{der}$	$\mathcal{F}'_{der} := \mathcal{F}_{der} \cup \{\mathsf{val}_\phi \mid \phi \in \mathsf{sub}(\varphi)\}$
Axioms $\mathcal{X}'$	$\mathcal{X}' := \mathcal{X} \cup \{x_\phi \mid \phi \in \mathsf{sub}(\varphi)\}$ where x_ϕ is $\begin{cases} \mathsf{val}_p \leftarrow p & (\phi = p) \\ \mathsf{val}_{\mathsf{Y}\phi'} \leftarrow \text{“}\mathsf{Y}\phi'\text{”} & (\phi = \mathsf{Y}\phi') \\ \mathsf{val}_{\phi_1 \mathsf{S} \phi_2} \leftarrow (\mathsf{val}_{\phi_2} \vee (\mathsf{val}_{\phi_1} \wedge \text{“}\mathsf{Y}(\phi_1 \mathsf{S} \phi_2)\text{”})) & (\phi = \phi_1 \mathsf{S} \phi_2) \\ \mathsf{val}_{\phi_1 \wedge \phi_2} \leftarrow (\mathsf{val}_{\phi_1} \wedge \mathsf{val}_{\phi_2}) & (\phi = \phi_1 \wedge \phi_2) \\ \mathsf{val}_{\neg\phi'} \leftarrow \neg \mathsf{val}_{\phi'} & (\phi = \neg\phi') \end{cases}$
Effects eff'	$eff'(a) := eff(a) \cup \{\mathsf{val}_\phi \rhd \{\text{“}\mathsf{Y}\phi\text{”}\}, \neg\mathsf{val}_\phi \rhd \{\neg\text{“}\mathsf{Y}\phi\text{”}\} \mid \text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi\}$
Initial State s'_0	$s'_0 := \sigma_0 \cup s_0$
Goal G'	$G' := \mathsf{val}_\varphi$

A solution to planning problem Γ is a sequence of actions $a \in A$ called *plan* $\pi = a_0, \ldots, a_{n-1}$ such that, when executed, induces a finite *state-trace* $\tau = s_0, \ldots, s_n$, where $s_i \vDash pre(a_i)$ and $s_{i+1} = tr(s_i, a_i)$ for $0 \leq i \leq n-1$, and $s_n \vDash G$. This definition of a classical planning problem can be naturally extended to the case of temporally extended goals expressed in PPLTL.

Definition 5. A classical planning problem with PPLTL goals is a tuple $\Gamma = \langle D, s_0, \varphi \rangle$, where $D = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, eff \rangle$ is a domain model, s_0 is the initial state, and φ is a PPLTL formula over $\mathcal{F}$.

In the case of temporally extended goals, the objective becomes to synthesize a sequence of states that satisfies the goal formula. Formally, a plan $\pi = a_0, \ldots, a_{n-1}$ is a solution for the planning problem $\Gamma = \langle D, s_0, \varphi \rangle$ if π induces a sequence of states $\tau = s_0, \ldots, s_n$ such that $\tau \vDash \varphi$. To solve Γ for PPLTL goals, we can build the deterministic automaton (DFA) for the domain and the nondeterministic automaton (NFA) for the goal formula,[4] compute their product, and check non-emptiness on the resulting automaton returning a plan if any [37,35]. The major drawback of this approach is that we need to compute the entire DFA (which can be exponentially larger than the goal formula) upfront. Instead, our approach completely bypasses the explicit DFA construction.

4.2. Encoding PPLTL goals in classical planning

This section presents a novel encoding for planning for PPLTL goals that exploits the technique presented in Section 3 to evaluate a PPLTL formula state-by-state. Intuitively, the encoding works as follows. Given a goal formula φ, we introduce a set of fresh fluents Σ_φ to monitor the truth of some specific subformulas of φ. Then, using the rules in Definition 1, we properly combine a state σ made of these new fluents with a state s from $\mathcal{F}$ to determine the truth of φ at every step of a generated trace. Moreover, we extend the model of the actions with new effects to iteratively compute σ as new states are generated. For the remainder of the paper, for the sake of clarity, we use s and σ to denote the partition of a planning state featuring the original domain fluents and the new fluents $\text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi$, respectively. Moreover, we use (σ, s) to denote the state $\sigma \cup s$.

Given a classical planning problem $\Gamma = \langle D, s_0, \varphi \rangle$, where $D = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, eff \rangle$ is a domain model, s_0 is the initial state, and φ a PPLTL goal, the encoded classical planning problem is $\Gamma' = \langle D', s'_0, G' \rangle$, where $D' = \langle \mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, eff' \rangle$ is the encoded domain model, s'_0 is the new initial state and G' is the new reachability goal. The full encoding is reported in Table 3.

In this encoding, the new axioms determine which subformula ϕ of the goal φ is true in a planning state s. In particular, the new classical planning problem includes an axiom $\mathsf{val}_\phi \leftarrow \psi$ for every subformula $\phi \in \mathsf{sub}(\varphi)$. Given a sequence of states $(\sigma_0, s_0), \ldots, (\sigma_n, s_n)$, these axioms mimic the rules in Definition 1 and are intended to be such that the current state $(\sigma_i, s_i) \vDash \mathsf{val}_\phi$ iff $\mathsf{val}(\phi, \sigma_i, s_i)$. Axioms not only elegantly model the mathematics behind Theorem 1 (that is, $\mathsf{val}(\phi, \sigma_i, s_i)$), but also simplify the action schema and goal descriptions.

From Table 3, it is easy to see that no new action is added to the encoded problem, and that the precondition function *pre* remains unchanged. In fact, every problem's action $a \in A$ is only modified on its effects $eff(a)$ by adding a way to update the assignments of propositions in Σ_φ. Specifically, σ_0 is encoded in the initial state, and for every subsequent step, we model the assignment $\sigma_i(\text{“}\mathsf{Y}\phi\text{”}) \doteq \mathsf{val}(\phi, \sigma_{i-1}, s_{i-1})$ for each $\text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi$ using two conditional effects of the form:

$$\mathsf{val}_\phi \rhd \ \text{“}\mathsf{Y}\phi\text{”}$$
$$\neg\mathsf{val}_\phi \rhd \neg\text{“}\mathsf{Y}\phi\text{”}.$$

[4] In fact, for PPLTL, the automaton is directly a DFA [33].

These effects specify that the fluent "Yϕ" is set to true (false, resp.) in the state s_i iff val_ϕ is true (false, resp.) in s_{i-1}. These additional effects are exactly the same for every action in A. Moreover, since σ_i maintains values of "Yϕ" in Σ_φ, they are *independent* from the effect of the action on the original fluents, which, instead, is maintained in the propositional interpretation s_i. This means that we can compute the next value of σ without knowing which action has been executed or which effect such action has had on the original fluents. Observe that the auxiliary part $\mathsf{eff}_{\mathsf{val}}$ in $\mathit{eff}'(a)$ updates the subformula values in Σ_φ without affecting any fluent $f \in \mathcal{F}$ of the original problem. This is crucial for the encoding's correctness.

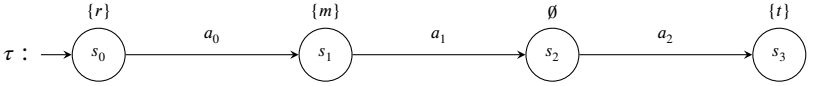
Example 7 *(Compilation Example).* We describe in detail the compilation of the formula $\varphi = t \wedge (\neg r \,\mathsf{S}\, m)$. Given a planning problem $\Gamma = \langle\langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff}\rangle, s_0, \varphi\rangle$, the compiled classical planning problem $\Gamma' = \langle\langle \mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, \mathit{eff}'\rangle, s_0, G\rangle$ is defined as follows:

- ($\mathcal{F}'$). The new set of fluents is the union of the original set of fluents $\mathcal{F}$ with $\Sigma_\varphi = \{\text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}\}$. Therefore, $\mathcal{F}' = \mathcal{F} \cup \{\text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}\}$;
- ($\mathcal{F}'_{der}$). The new set of derived predicates is obtained by adding one fluent of the form val_ϕ for each $\phi \in \mathsf{sub}(\varphi)$. Since $\mathsf{sub}(\varphi) = \{r, m, t, \neg r, \neg r \,\mathsf{S}\, m, t \wedge (\neg r \,\mathsf{S}\, m)\}$, we have $\mathcal{F}'_{der} = \mathcal{F}_{der} \cup \{\mathsf{val}_r, \mathsf{val}_m, \mathsf{val}_t, \mathsf{val}_{\neg r}, \mathsf{val}_{\neg r \,\mathsf{S}\, m}, \mathsf{val}_{t \wedge (\neg r \,\mathsf{S}\, m)}\}$;
- ($\mathcal{X}'$). Following the rules illustrated in Table 3, we have that $\mathcal{X}'$ is $\mathcal{X}$ plus the following new axioms:
  - $\mathsf{val}_{t \wedge (\neg r \,\mathsf{S}\, m)} \leftarrow (\mathsf{val}_t \wedge \mathsf{val}_{\neg r \,\mathsf{S}\, m})$;
  - $\mathsf{val}_{\neg r \,\mathsf{S}\, m} \leftarrow (\mathsf{val}_m \vee (\mathsf{val}_{\neg r} \wedge \text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}))$;
  - $\mathsf{val}_{\neg r} \leftarrow \neg \mathsf{val}_r$;
  - $\mathsf{val}_r \leftarrow r$;
  - $\mathsf{val}_m \leftarrow m$;
  - $\mathsf{val}_t \leftarrow t$.
- (eff'). We extend the effects of each action $act \in A$ as follows:

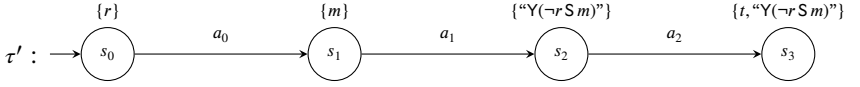
$$\mathit{eff}'(act) = \mathit{eff}(act) \cup \{\mathsf{val}_{\neg r \,\mathsf{S}\, m} \rhd \text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}, \neg \mathsf{val}_{\neg r \,\mathsf{S}\, m} \rhd \neg \text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}\};$$

- (s'_0). The new initial state is $s'_0 = s_0 \cup \sigma_0$ where $\sigma_0 = \emptyset$, meaning that "Y$(\neg r \,\mathsf{S}\, m)$" is false in s'_0;
- (G'). The new reachability goal is $G' = \mathsf{val}_{t \wedge (\neg r \,\mathsf{S}\, m)}$.

Consider the plan $\pi = \langle a_0, a_1, a_2\rangle$ for Γ that induces the following state trajectory.



It is easy to see that $\tau \vDash t \wedge (\neg r \,\mathsf{S}\, m)$. By analyzing the structure of the compiled problem Γ', and in particular the new effect function eff', we can see that, for the encoded problem Γ', π induces the state trajectory:



In particular, the conditional effect $\mathsf{val}_{\neg r \,\mathsf{S}\, m} \rhd \text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}$ of actions a_1 makes "Y$(\neg r \,\mathsf{S}\, m)$" true in state s_2 because m is true in s_1. The same conditional effect of the next action a_2 maintains "Y$(\neg r \,\mathsf{S}\, m)$" true in s_3 because $\neg r \wedge \text{"Y}(\neg r \,\mathsf{S}\, m)\text{"}$ holds in s_2. Since $t, \neg r$, and "Y$(\neg r \,\mathsf{S}\, m)$" hold the last state s_3, we have that $\mathsf{val}_{t \wedge (\neg r \,\mathsf{S}\, m)}$ (the goal of Γ') also holds in the last state.

It is easy to see that our encoding is polynomially related to the original problem.

Theorem 3. *The size of the encoded planning problem Γ' is polynomial in the size of the original problem Γ. In particular, the additional fluents introduced are linear in $|\mathsf{sub}(\varphi)|$.*

Proof. Immediate, by analyzing the construction. □

Next, we turn to correctness. Let $\Gamma = \langle \mathcal{D}, s_0, \varphi\rangle$ be a planning problem, where $\mathcal{D}$ is a domain, s_0 is the initial state, and φ is a PPLTL goal formula, and let $\Gamma' = \langle \mathcal{D}', s'_0, G'\rangle$ be the corresponding compiled planning problem as previously defined. Any trace $\tau' = s'_0, \ldots, s'_n$ on $\mathcal{D}'$ can be seen as $\tau' = zip(\tau^{[\varphi]}, \tau)$, with $\tau^{[\varphi]} = \sigma_0, \ldots, \sigma_n \in (2^{\Sigma_\varphi})^+$ and $\tau = s_0, \ldots, s_n \in (2^{\mathcal{F}})^+$, where each element of τ' is of the form $s'_i = (\sigma_i, s_i)$ for all $i \geq 0$. Here, we use the $zip(\cdot, \cdot)$ function to represent the aggregation of the two traces $\tau^{[\varphi]}$ and τ. Given a trace $\tau' = s'_0, \ldots, s'_n$ on the encoded planning domain $\mathcal{D}'$, there exists a single trace $\tau' \,|_{\mathcal{F}} = \tau = s_0, \ldots, s_n$ on the original planning domain $\mathcal{D}$. Conversely, given a trace $\tau = s_0, \ldots, s_n$ on the original planning domain $\mathcal{D}$, there exists a unique corresponding trace $\tau^{[\varphi]}$, and hence a single $\tau' = zip(\tau^{[\varphi]}, \tau)$ on the encoded domain $\mathcal{D}'$. Finally, we observe that every executable action sequence $a_0, \ldots, a_{n-1}$ in the planning problem Γ with PPLTL goal φ is also executable in the encoded planning problem Γ' (and vice versa) since the encoding does not have auxiliary actions and preconditions do not change.

Theorem 4 *(Correctness). Let Γ be a planning problem with a PPLTL goal φ, and Γ' be the corresponding encoded planning problem with a reachability goal. Then, every action sequence $\pi = a_0, \dots, a_{n-1}$ is a plan for Γ iff $\pi = a_0, \dots, a_{n-1}$ is a plan for Γ'.*

Proof. Every executable action sequence $a_0, \dots, a_{n-1}$ in the planning problem Γ with PPLTL goal φ is also executable in the encoded planning problem Γ' (and vice versa) since, by definition, the encoding does not have auxiliary actions, actions preconditions do not change, and additional conditional effects on the original actions are deterministic.

The action sequence π is a plan if its induced state trace τ is such that $\tau \vDash \varphi$. By Theorem 1, we have that $\tau \vDash \varphi$ iff $\mathsf{val}(\varphi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau))$. However, by construction of the encoding for Γ', we have that $\mathsf{val}(\varphi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau))$ holds iff val_φ holds in the last state of the induced state trace for Γ', i.e., in $\tau' = zip(\tau^{[\varphi]}, \tau)$. In other words, $\mathsf{val}(\varphi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau))$ iff $\mathsf{last}(\tau') \vDash \mathsf{val}_\varphi$. Hence, the thesis holds. □

A direct consequence of Theorem 4 is that every sound and complete planner returns a plan π for the encoded planning problem Γ' if a plan π for the original planning problem Γ with PPLTL goal exists. If no solution exists for Γ', then no solution exists for Γ.

We conclude by discussing how the approach presented in this section can be interpreted with an automata-theoretic perspective, cf. [35]. Essentially, our encoding synchronizes the *symbolic* representation of the planning problem Γ and the *symbolic* DFA $\mathcal{A}_\varphi$ (Definition 4) of the PPLTL goal formula φ by computing their cross-product. This process is very similar to the schema for handling PPLTL goals outlined in Section 4.1, except that we do not have to compute the explicit DFAs for the problem and PPLTL goal. In our case, each planning state (σ, s) of the resulting problem Γ' keeps the current state q of the automaton $\mathcal{A}_\varphi$ in σ. Moreover, as discussed in Section 3.1, we do not use the auxiliary proposition $\mathtt{final}$. Instead, we check whether the automaton is accepting by evaluating the condition $\mathsf{val}(\varphi, q_n, s_n)$ with the derived predicate val_φ. Finally, it is worth considering whether information about dead-end states in the symbolic DFA can be leveraged to extend the preconditions of actions. Although a similar optimization has been applied to PPLTL formulas with a specific structure [25], extending this approach to arbitrary PPLTL goals would require, in general, enumerating all states of the symbolic DFA, thus undermining the advantages of our polynomial encoding.

5. FOND planning for pure-past linear temporal logic goals

In the previous section, we provided a compact, sound, and complete encoding of classical planning for PPLTL goals into classical planning for reachability goals. Most notably, by inspecting the encoding, we observe that the new effects added by the compilation are designed to update the key fluents representing the subformulas of the PPLTL goal by only considering the current state. This observation suggests that the same encoding, with minor modifications, can also be employed by FOND domain models, that is, in planning domains where action effects are non-deterministic, but we operate under full observability. Indeed, in this section, we formally prove that we can use the schema presented in Section 4 to encode FOND planning for PPLTL goals into FOND planning with reachability goals.

This section is organized as follows. We provide the necessary background notions of FOND planning for reachability goals and then introduce FOND planning for temporally extended goals in PPLTL. Finally, we formally prove that we can polynomially encode a FOND problem for PPLTL goals into an *equivalent* FOND problem for reachability goals using the encoding presented in Section 4.

5.1. FOND planning

A *Fully Observable Non-Deterministic* (FOND) domain model can be formalized as a tuple $\mathcal{D} = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, eff \rangle$ where $\mathcal{F}$, $\mathcal{F}_{der}$, $\mathcal{X}$, A, and pre are defined as in the classical planning case, while eff is a function that takes an action label $a \in A$ as input and returns a set $\{\mathtt{eff}_1, \dots, \mathtt{eff}_n\}$ of effects, where each effect $\mathtt{eff}_i \in eff(a)$ is a set of conditional effects. Analogously to classical planning, an action a can be applied in a state s if $s \vDash pre(a)$. Differently, applying a in s yields a successor state s' determined by one of the sets of conditional effects $\mathtt{eff}_i \in eff(a)$ *nondeterministically* drawn from $eff(a)$. The new state s' is then obtained according to the set of conditional effects $\mathtt{eff}_i \in eff(a)$ as defined in Section 4.1 for classical planning. In the context of FOND planning, $tr(s, a)$ denotes the set of possible successor states $\{s'_1, \dots, s'_n\}$ obtained by executing a in s. We also assume that if $s \nvDash pre(a)$ then $tr(s, a) = \emptyset$.

A FOND planning problem is a tuple $\Gamma = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a FOND domain model, $s_0 \subseteq \mathcal{F}$ is the initial state, and G is a formula over $\mathcal{F} \cup \mathcal{F}_{der}$ representing the reachability goal. In FOND planning, solutions to Γ are *strategies* (aka *policies*). A strategy is defined as a partial function $\pi : (2^{\mathcal{F}})^+ \to A$ mapping a sequence of *non-goal* states into an applicable action. A strategy π for Γ, starting from the initial state s_0, induces a set of (possibly infinite) state trajectories (or executions) Λ of the form $\tau = s_0, s_1, \dots$, where s_0 is the initial state, $s_{i+1} \in tr(s_i, a_i)$, and $a_i = \pi(s_0, \dots, s_i)$ for $i \geq 0$. Moreover, a state trajectory $\tau = s_0, \dots, s_n$ induced by π is a *finite* trace if $\pi(\tau)$ is undefined, that is, no further action is prescribed by π.

As usual, we consider two kinds of solutions to FOND planning problems: *strong* solutions and *strong-cyclic* solutions [32]. A strategy π is a *strong solution* to Γ if every generated execution is a finite trace τ such that the last state s_n of τ entails G, while π is *strong-cyclic* if every generated execution that is a *stochastic-fair trace* [3] is also a finite trace such that $s_n \vDash G$. Intuitively, stochastic fairness assumes that every outcome of an action can occur with a non-zero probability. So, assuming stochastic fairness, a strategy π is a strong-cyclic solution if every trace induced by π terminates satisfying the goal with probability 1. When a strategy π is a solution (either strong or strong-cyclic, depending on the kind of solution we are interested in), we say that π is *winning*.

As done in the classical planning setting, we consider the case where the goal to achieve is a temporally extended goal expressed in PPLTL.

Definition 6. A FOND planning problem for temporally extended goals is a tuple $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$, where $\mathcal{D} = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff} \rangle$ is a FOND domain model, s_0 is the initial state, and φ is a PPLTL formula over $\mathcal{F}$.

In the case of a temporally extended goal φ, a strategy $\pi : (2^{\mathcal{F}})^+ \to A$ is a solution to Γ if every state trace induced by π is finite and satisfies φ. When the strategy needs finite memory, then it can be represented as a finite-state transducer [35].

5.2. Encoding PPLTL goals in FOND planning

We now show that the schema presented in Section 4 can be extended to encode FOND planning for PPLTL goals into FOND planning for reachability goals.

Definition 7 *(FOND encoding).* Let $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$ be a FOND planning problem, where $\mathcal{D} = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff} \rangle$ is a non-deterministic domain, s_0 the initial state and φ a PPLTL goal. The encoded FOND planning problem is $\Gamma' = \langle \mathcal{D}', s_0', G' \rangle$, with $\mathcal{D}' = \langle \mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, \mathit{eff}' \rangle$ where $\mathcal{F}'$, $\mathcal{F}'_{der}$, $\mathcal{X}'$, s_0' and G' are defined as in Table 3, while for every $a \in A$, $\mathit{eff}'(a)$ is defined as:

$$\begin{aligned} &\mathit{eff}'(a) := \{\mathtt{eff}_i \cup \mathtt{eff}_{\mathsf{val}} \mid \mathtt{eff}_i \in \mathit{eff}(a)\}, \text{ where} \\ &\mathtt{eff}_{\mathsf{val}} \quad = \{\mathsf{val}_\phi \rhd \{\text{“}Y\phi\text{”}\}, \neg\mathsf{val}_\phi \rhd \{\neg\text{“}Y\phi\text{”}\} \mid \text{“}Y\phi\text{”} \in \Sigma_\varphi\}. \end{aligned}$$

Definition 7 extends the encoding schema in Table 3 to FOND problems by simply adding the set $\mathtt{eff}_{\mathsf{val}}$ of new conditional effects to every outcome of actions.

It is easy to see that Theorem 3 also holds in the case of a FOND problem Γ. That is, the size of Γ' is polynomial in the size of Γ, and the additional fluents are linear in the size of the PPLTL goal φ.

We now turn to correctness, that is, we show that the encoding is sound and complete. Unlike in classical planning, where it is sufficient to show that there is a one-to-one correspondence between plans for the original and encoded problem, the correctness proof for FOND problems requires proving that it is always possible to construct a solution strategy to Γ given the solution strategy to Γ', and vice versa. We do so using the correspondence between the traces of the encoded domain $\mathcal{D}'$ and the original domain $\mathcal{D}$. Note that in the encoded domain $\mathcal{D}'$ it is sufficient to consider policies mapping a state to an action, similar to classical planning. However, for consistency, we keep the definition of policies in the FOND setting as functions mapping traces to actions.

Given a trace $\tau = s_0, \ldots, s_n \in (2^{\mathcal{F}})^+$ on $\mathcal{D}$ and trace $\tau^{[\varphi]} = \sigma_0, \ldots, \sigma_n \in (2^{\Sigma_\varphi})^+$, we again use '$zip(\tau, \tau^{[\varphi]})$' to indicate the trace $\tau' = (s_0, \sigma_0), \ldots, (s_n, \sigma_n)$ on $\mathcal{D}'$ obtained by aggregating τ with $\tau^{[\varphi]}$. Moreover, given a trace $\tau' = s_0', \ldots, s_n'$ on $\mathcal{D}'$, we use $\tau' \mid_{\mathcal{F}}$ to denote the trace $\tau' \mid_{\mathcal{F}} = \tau = s_0, \ldots, s_n$ on the original planning domain $\mathcal{D}$.

Recall that every trace $\tau' = s_0', \ldots, s_n'$ on $\mathcal{D}'$ can be uniquely mapped to the trace $\tau' \mid_{\mathcal{F}} = \tau = s_0, \ldots, s_n$ on $\mathcal{D}$, and vice versa. Therefore, for every strategy $\pi : (2^{\mathcal{F}})^+ \to A$ for the FOND planning problem Γ with PPLTL goal φ, we can build the strategy $\pi' : (2^{\mathcal{F}'})^+ \to A$ for Γ' as follows:

$$\begin{aligned} &\pi'(\tau') = a && \text{iff} \quad \pi(\tau' \mid_{\mathcal{F}}) = a \\ &\pi'(\tau') \text{ is undefined} && \text{iff} \quad \pi(\tau' \mid_{\mathcal{F}}) \text{ is undefined.} \end{aligned}$$

Lemma 1. *For every $\pi : (2^{\mathcal{F}})^+ \to A$ that is a (strong or strong-cyclic) winning strategy for the FOND planning problem Γ with PPLTL goal φ, the $\pi' : (2^{\mathcal{F}'})^+ \to A$, defined as above, is a (strong or strong-cyclic, resp.) winning strategy for the encoded planning problem Γ'.*

Proof. Strategy π is winning if every generated execution τ (that is stochastic-fair, for strong-cyclic solutions) is finite, i.e., $\pi(\tau)$ is undefined, and such that $\tau \vDash \varphi$. Correspondingly, the strategy π' induces the finite generated execution $\tau' = zip(\tau^{[\varphi]}, \tau)$. Then, $\mathsf{val}(\varphi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau))$ holds by Theorem 1, so we have that $\mathsf{last}(\tau') \vDash \mathsf{val}_\varphi$.

On the other hand, if a generated execution τ' is finite, i.e., such that $\pi'(\tau')$ is undefined, then π induces a corresponding finite generated execution $\tau = \tau' \mid_{\mathcal{F}}$. Being π winning, it must be the case that $\tau \vDash \varphi$. Hence, by Theorem 1, $\mathsf{last}(\tau') \vDash \mathsf{val}_\varphi$. Thus, if π is winning for Γ, then π' is winning for Γ'. □

Now we consider the converse. Again, the correspondence between traces of $\mathcal{D}$ and $\mathcal{D}'$ imply that for every strategy $\pi' : (2^{\mathcal{F}'})^+ \to A$ for the encoded planning problem Γ', we can build the strategy $\pi : (2^{\mathcal{F}})^+ \to A$ for the original problem Γ with PPLTL goal φ as follows (where $\tau' = zip(\tau^{[\varphi]}, \tau)$):

$$\begin{aligned} &\pi(\tau) = a && \text{iff} \quad \pi'(\tau') = a \\ &\pi(\tau) \text{ is undefined} && \text{iff} \quad \pi'(\tau') \text{ is undefined.} \end{aligned}$$

Lemma 2. *For every $\pi' : (2^{\mathcal{F}'})^+ \to A$ that is a (strong or strong-cyclic) winning strategy for the encoded planning problem Γ', the $\pi : (2^{\mathcal{F}})^+ \to A$, defined as above, is a (strong or strong-cyclic, resp.) winning strategy for the FOND planning problem Γ with PPLTL goal φ.*

Proof. Strategy π' is winning if every generated execution τ' (that is stochastic fair, for strong-cyclic solutions) is finite, i.e., such that $\pi'(\tau')$ is undefined, and such that $\mathsf{last}(\tau') \vDash \mathsf{val}_\varphi$. Correspondingly, the strategy π induces the finite execution $\tau = \tau' \mid_{\mathcal{F}}$. Then, by Theorem 1, considering that $\mathsf{val}(\varphi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau))$ holds, we have that $\tau \vDash \varphi$.

On the other hand, if a generated execution τ is finite, i.e., such that $\pi(\tau)$ is undefined then π' induces a corresponding finite generated execution $\tau' = zip(\tau^{[\varphi]}, \tau)$. Being π' winning, we have that $\mathsf{last}(\tau') \vDash \mathsf{val}_\varphi$. Hence, by Theorem 1, $\tau \vDash \varphi$. Thus, if π' is winning for Γ', then π is winning for Γ. □

Theorem 5 *(Correctness). Let Γ be a FOND planning problem with a PPLTL goal φ, and Γ' be the corresponding encoded FOND planning problem with reachability goal G'. Then, Γ has a (strong or strong-cyclic) winning strategy iff Γ' has a (strong or strong-cyclic, resp.) winning strategy.*

Proof. Direct consequences of Lemma 1 and Lemma 2. □

Also in this case, as discussed in Section 4.2, the result of our encoding can be seen as the cross-product between the symbolic DFA of the FOND problem Γ [35] and the *symbolic* DFA $\mathcal{A}_\varphi$ of the PPLTL goal formula φ.

6. Encoding variant: elimination of axioms

We describe a variant of the encoding presented in the previous sections that does not add derived predicates. We present this encoding in the context of FOND problems only, as any classical planning domain can be seen as a FOND domain where $|\mathit{eff}(a)| = 1$ for every action a of the domain.

Like the previous encoding, this variant introduces a fresh fluent of the form "$\mathsf{Y}\phi$" for each temporal component of φ. However, instead of using the $\mathsf{val}(\cdot)$ predicates, it explicitly represents the PNF of a PPLTL formula as a propositional formula. We, therefore, introduce the $\mathsf{ppnf}(\cdot)$ transformation.

Definition 8. Let φ be a PPLTL formula. $\mathsf{ppnf}(\varphi)$ is a propositional formula obtained by substituting every $\mathsf{Y}(\phi)$ with "$\mathsf{Y}(\phi)$" in $\mathsf{pnf}(\phi)$.

For example, if $\varphi = a\,\mathsf{S}\,b$ then $\mathsf{ppnf}(\varphi) = b \vee (a \wedge \text{“}\mathsf{Y}(a\,\mathsf{S}\,b)\text{”})$. For any formula φ, the $\mathsf{ppnf}(\varphi)$ captures the truth of $\mathsf{pnf}(\varphi)$ without using temporal operators, provided that every "$\mathsf{Y}\phi$" $\in \Sigma_\varphi$ reflects the truth of $\mathsf{Y}\phi$. Most importantly, $\mathsf{ppnf}(\varphi)$ is linear in the size of φ.

Lemma 3. *Let φ be a PPLTL formula. Then $|\mathsf{ppnf}(\varphi)|$ is linear in $|\varphi|$.*

Proof. The formula $\mathsf{ppnf}(\varphi)$ can be computed by recursively exploring the structure of φ using the following rules:

- $\mathsf{ppnf}(p) = p$;
- $\mathsf{ppnf}(\mathsf{Y}\phi) =$ "$\mathsf{Y}\phi$";
- $\mathsf{ppnf}(\phi_1\,\mathsf{S}\,\phi_2) = \mathsf{ppnf}(\phi_2) \vee (\mathsf{ppnf}(\phi_1) \wedge \text{“}\mathsf{Y}(\phi_1\,\mathsf{S}\,\phi_2)\text{”})$;
- $\mathsf{ppnf}(\phi_1 \wedge \phi_2) = \mathsf{ppnf}(\phi_1) \wedge \mathsf{ppnf}(\phi_2)$;
- $\mathsf{ppnf}(\neg\phi) = \neg\mathsf{ppnf}(\phi)$.

As we can see, the ppnf computation makes φ grow by a constant factor only when evaluating the S operator. Furthermore, for any element of the formula, $\mathsf{ppnf}(\cdot)$ is applied at most once. This implies that the size of $\mathsf{ppnf}(\varphi)$ is linear in the size of φ. □

Definition 9 *(Axiom-free encoding).* Let $\Gamma = \langle D, s_0, \varphi\rangle$ be either a FOND or a classical planning problem, where $D = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, \mathit{pre}, \mathit{eff}\rangle$ is a domain model, s_0 the initial state, and φ a PPLTL goal. The encoded planning problem is $\Gamma'' = \langle D'', s_0', G'\rangle$ with $D'' = \langle \mathcal{F}', \mathcal{F}_{der}, \mathcal{X}, A, \mathit{pre}, \mathit{eff}''\rangle$, where $\mathcal{F}'$, s_0', G' are defined as in Table 3, while, for every $a \in A$, $\mathit{eff}''(a)$ is defined as follows:

$$\mathit{eff}''(a) := \{\mathsf{eff}_i \cup \mathsf{eff}_{\mathsf{ppnf}} \mid \mathsf{eff}_i \in \mathit{eff}(a)\}, \text{ where}$$
$$\mathsf{eff}_{\mathsf{ppnf}} = \{\mathsf{ppnf}(\phi) \rhd \{\text{“}\mathsf{Y}\phi\text{”}\}, \neg\mathsf{ppnf}(\phi) \rhd \{\neg\text{“}\mathsf{Y}\phi\text{”}\} \mid \text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi\}.$$

Intuitively, this encoding is defined exactly as the encodings reported in Table 3 and Definition 7, where the only difference is that val_ϕ is changed with the $\mathsf{ppnf}(\phi)$ in the condition of conditional effects. The correctness of this approach comes from the fact that every state (s_i, σ_i) satisfies val_ϕ iff (s_i, σ_i) satisfies $\mathsf{ppnf}(\phi)$.

Theorem 6 *(Correctness of axiom-free encoding). Let Γ be a FOND planning problem and let Γ'' be the problem encoded following Definition 9. Then Γ has a winning strategy π iff so does Γ''.*

Proof. Let Γ' be the problem obtained by Γ following the encoding of Definition 7. By Theorem 5 we know that Γ' has a winning strategy iff so does Γ. So in order to prove our thesis, we can show that a strategy π is winning for Γ' iff so does for Γ''. By definition, Γ'' and Γ' share the same fluents, initial state, actions, and precondition function. Therefore, if an action a is applicable in a state s for Γ', so is for Γ'', and vice versa. In addition, it is easy to see that, by the definition of ppnf and the derived predicates of Γ', for any state s, $\forall\phi \in \mathsf{sub}(\varphi) \cdot s \vDash \mathsf{ppnf}(\phi)$ iff $s \vDash \mathsf{val}_\phi$. This has two implications:

1. A conditional effect $\mathsf{val}_\phi \rhd$ "Y(ϕ)" ($\neg\mathsf{val}_\phi \rhd \neg$"Y($\phi$)", resp.) is triggered by a state s iff so does $\mathsf{ppnf}(\phi) \rhd$ "Y(ϕ)" ($\neg\mathsf{ppnf}(\phi) \rhd \neg$"Y($\phi$)", resp.). By analyzing the construction, this implies that, for every state s and for every action a applicable in s, the effects $\mathtt{eff}'(a)$ and $\mathtt{eff}''(a)$ will induce the same set of possible successor states (recall that the effects added by the two encodings only affect "Y(ϕ)" fluents, and do not modify any fluent of the original problem). Therefore, since Γ'' and Γ' have the same initial state, the strategy π will induce the same set of traces for both Γ' and Γ''.
2. For any state s, $s \vDash G'$ iff $s \vDash G''$.

The above points imply that a strategy π is winning for Γ' iff is winning for Γ''. This plus Theorem 5 imply that Γ admits a winning strategy π iff so does Γ''. □

Although this second encoding removes additional axioms that the first encoding generates, it remains polynomially related to the original problem.

Theorem 7. *Let Γ be either a FOND or a classical planing. The size of Γ'' obtained following the encoding of Definition 9 is polynomial in the size of Γ. In particular, it is linear in the size of the domain specification and quadratic in the size of the goal.*

Proof. By analyzing the construction, we can observe that $|\mathcal{F}''| = |\mathcal{F}| + |\{\text{"Y}\phi\text{"} \mid \text{"Y}\phi\text{"} \in \Sigma_\varphi\}|$ and $|\Sigma_\varphi| \leq |\mathsf{sub}(\varphi)|$. Furthermore, $|s_0''| = |s_0|$ and the goal G'' is linear in the size of φ by Lemma 3. We proceed by analyzing eff''. The encoding adds the set $\mathtt{eff}_{\mathsf{ppnf}}$ to the effects of actions $a \in A$. For a conditional effect $c \rhd e \in \mathtt{eff}_{\mathsf{ppnf}}$, we have that $|e| = 1$, while c is a formula of size $|\mathsf{ppnf}(\phi)|$ where $\phi \in \mathsf{sub}(\varphi)$. In the worst case, since $|\phi| \leq |\varphi|$ (as ϕ is a subformula of φ) we have that $|\phi| = |\varphi|$ and therefore, by Lemma 3, $|\mathsf{ppnf}(\phi)|$ is $O(|\varphi|)$. This implies that the size of $\mathit{eff}(a)$ grows as $O(|\varphi|^2)$ since $\mathtt{eff}_{\mathsf{ppnf}}$ contains at most two effects for each subformula of φ (that is, $|\mathtt{eff}_{\mathsf{ppnf}}|$ is $O(|\varphi|)$), and each element of $\mathtt{eff}_{\mathsf{ppnf}}$ can be as large as $|\varphi|$ itself. □

We conclude this section by remarking that if the original domain $\mathcal{D}$ does not include derived predicates, that is, $\mathcal{D} = \langle \mathcal{F}, \emptyset, \emptyset, A, \mathit{pre}, \mathit{eff} \rangle$, then also the encoded domain $\mathcal{D}''$ will not include derived predicates, enabling the use of any state-of-the-art planner not supporting this feature.

7. Experimental analysis

The goal of our experimental evaluation is to understand the effectiveness of the proposed encoding over large and non-trivial PPLTL goals in comparison to the state-of-the-art systems supporting semantically equivalent LTL_f goals. To do so, we implemented the approach of the previous sections in a tool called Plan4Past (P4P),[5] which can be run with the intensive conditional effects compilation ($\text{P4P}_{\mathsf{adl}}$ for short) or with derived predicates ($\text{P4P}_{\mathcal{X}}$ for short). Preliminary experiments revealed how current planners struggle to preprocess most of the problems compiled by $\text{P4P}_{\mathsf{adl}}$. This behavior is caused by the complexity of conditional effects introduced by the compilation. Since all actions have the same set $\mathtt{eff}_{\mathsf{ppnf}}$ of effects, we can overcome this issue by aggregating these effects into a dummy action `check`. Then, we force the encoding to execute one occurrence of `check` before any domain actions. Such a modification does not undermine our theoretical results and instead proves to be much more convenient with the current planners. Thus, we will hereinafter refer to $\text{P4P}_{\mathsf{adl}}$ as the compilation with this modification.

Comparing P4P with compilations for LTL_f goals requires a set of semantically equivalent LTL_f and PPLTL formulas. Although LTL_f and PPLTL have the same expressiveness, no tool to translate one into the other exists yet. Therefore, in our experiments, this translation was performed manually. Then, for each LTL_f and PPLTL goal, we formally and automatically proved their semantic equivalence by verifying that the two formulations yield the same minimal DFA (modulo state renaming). In addition, to have a fair comparison, we considered PPLTL and LTL_f formulas that are *polynomially related* in terms of size.

For temporally extended goals in PPLTL, P4P supports both FOND and classical planning problems. On the other hand, the state-of-the-art approaches for handling LTL_f goals in classical planning are practically and methodologically different from those designed for FOND domains. For this reason, we address these two settings separately. Specifically, our analysis first compares P4P with the state-of-the-art compilations for LTL_f goals in classical domains, and then we focus this comparison over FOND domains. For both settings, we briefly describe the considered LTL_f approaches, the planners we employed, benchmark domains, and temporally extended goals we used for the comparison.

7.1. Deterministic setting

To our knowledge, in the deterministic setting, the state-of-the-art approaches for handling LTL_f goals are the encodings by Baier and McIlraith [13] (LTLExp) and [74] (LTLPoly). In particular, [13] builds an NFA for the LTL_f formula and computes the Cartesian product with the planning domain (cf. [35]), incurring in a worst-case exponential increase in the number of states of the NFA. On the other hand, Torres and Baier [74] constructs a symbolic NFA for the goal formula and encodes it in the planning problem. In principle, this approach is similar to Plan4Past's encoding: the (factorized) state of the symbolic NFA tracks the progression of the

[5] Source code and benchmarks are publicly available at https://github.com/whitemech/Plan4Past.

LTL_f formula and evolves as actions are executed. Similar to our encoding, Torres and Baier (2015)'s [74] construction remains polynomial in the size of the goal. However, a crucial difference lies in how the state is updated. Specifically, evolving the state of the symbolic NFA during planning requires introducing additional dummy (synchronization) actions into the plan, which significantly increases the length of solutions and makes the overall search harder for the planner. Dealing with spurious actions has already been studied in [64] theoretically and practically from a heuristic perspective in, e.g., [52]. Instead, Plan4Past exploits the semantics of PPLTL to track and incrementally update the satisfaction of the goal subformulas without introducing any spurious action. Therefore, compared to LTLExp and LTLPoly, the theoretical advantage of P4P is clear: under an automata-theoretic view, P4P exploits the fact that goals are expressed in PPLTL to implicitly and incrementally build a DFA (vs. an NFA) for the temporal formula while doing planning, keeping optimality with respect to computational complexity and preserving the plan length.

To determine whether this theoretical advantage manifests itself in actual planning performance, we tested $\text{P4P}_{\mathcal{X}}$, P4P_{adl}, LTLExp and LTLPoly over a set of benchmarks and analyzed the number of problems solved (Coverage), the time spent to find a solution (compilation plus search time), the number of expanded nodes, and the plan length. As a classical planner, we considered LAMA [68], a planner built on top of FastDownward [54], and $\text{FF}_{\mathcal{X}}$ [73]. LAMA is a satisficing planner based on a sophisticated search mechanism that runs (in the first iteration) Lazy Greedy Best-First Search driven by the h_{ff} [56] and the landmarks counting heuristics. LAMA yields solution plans of decreasing plan cost incrementally; for our analysis, we take the first generated plan that corresponds to running LAMA with the "lama-first" alias. $\text{FF}_{\mathcal{X}}$ is yet another satisficing planner based on enforced hill climbing and is the one originally used with LTLPoly and LTLExp. Both systems handle the compiled problems, but in the rest of this section, we will focus on LAMA as it was the system with the highest overall coverage for all compilations.

7.1.1. Benchmark domains

Our benchmark suite includes domains BLOCKSWORLD (abbreviated with BLOCKS), ELEVATOR, ROVERS, and OPENSTACKS. These domains were introduced in past International Planning Competitions, and all but ELEVATOR have also been used by Torres and Baier [74]. For BLOCKS, ROVERS, and OPENSTACKS, we have a set of instances with the same temporally extended goals defined by Torres and Baier [74] (hereinafter referred to as TB15), and a second set of instances with temporally extended goals defined by us (hereinafter referred to as BF23). For ELEVATOR, we only have BF23.

TB15 were originally specified in LTL_f and are based on predefined families of formulas specified in [74]. For P4P, we manually translated such patterns into PPLTL, as reported in the Appendix (Table 10). We did so for all but one type of temporally extended goals used in [74], namely those of type '$h : \alpha \,\mathsf{U}\, \beta$' where α or β have n nested U operators, for which we did not find an easy translation into PPLTL.[6]

BF23 were designed in PPLTL directly and, analogously to what was done for TB15, we got an equivalent formulation in LTL_f. Differently from TB15, BF23 are specific for each domain; they were designed to stress all compilations and understand the planner's scalability over non-trivial and large instances. Indeed, all instances with TB15 proved trivial for Plan4Past. For TB15, we have 15 instances for BLOCKS, 7 for ROVERS, and 10 for OPENSTACKS. Their definition is provided by Torres and Baier [74]. We briefly describe the PPLTL goals employed in BF23 below, while we provide the LTL_f formulation of BF23 in the Appendix (Table 9).

BLOCKS In this domain, we want to arrange blocks in a particular configuration. An arm can pick up and move blocks on top of other blocks or the table. BF23 were formulated to study the reach of all compilations with complex temporally extended goals. Here, BF23 specifies a goal $\varphi = \varphi_1 \wedge \varphi_2$ that intertwines two formulas, both requiring the existence of a particular sequence of states in the trace of the plan. In particular, φ_1 consists of the pattern $\mathsf{O}(p_1 \wedge \mathsf{Y}(\mathsf{O}(p_2 \wedge \mathsf{Y}(\mathsf{O}(p_3 \wedge \mathsf{Y}(\mathsf{O}(\ldots \wedge \mathsf{Y}(\mathsf{O}(p_k)))))))))$, with $p_1, p_2, p_3, \ldots, p_k$ propositions, which requires the existence of a state that satisfies p_1, preceded by a state that satisfies p_2, and so on. On the other hand, φ_2 consists of the slightly different pattern $\mathsf{O}(p_1 \wedge \mathsf{Y}(\psi)) \wedge \mathsf{O}(p_2 \wedge \mathsf{Y}(\psi)) \wedge \mathsf{O}(p_3 \wedge \mathsf{Y}(\psi)) \wedge \ldots \wedge \mathsf{O}(p_k \wedge \mathsf{Y}(\psi))$ where $p_1, p_2, p_3, \ldots, p_k$ are propositions and ψ is a PPLTL formula defined following the pattern of φ_1. Intuitively, this formula specifies that the sub-goals p_1, p_2, p_3, …, p_k must be achieved, with no particular order between them, after achieving the PPLTL formula ψ. In BF23, these patterns are designed to grow incrementally with the number of blocks to obtain challenging temporally extended goals. Specifically, consider a problem with n blocks, and let $on_{i,j}$ be the predicate modeling block i being on block j. For an even number of blocks, φ_1 and φ_2 are formulated in PPLTL as follows:

$$\varphi_1 = \mathsf{O}(on_{1,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,3} \wedge \mathsf{Y}(\mathsf{O}(\ldots \wedge \mathsf{Y}(\mathsf{O}(on_{n-1,n})))))))$$

$$\varphi_2 = \bigwedge_{j \in \{6,8,\ldots,n\}} \mathsf{O}(on_{j,j-1} \wedge \mathsf{Y}(\mathsf{O}(on_{4,3} \wedge \mathsf{Y}(\mathsf{O}(on_{3,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,1})))))))).$$

The formulation for an odd number of blocks is analogous. We generate a temporally extended goal for each instance of the domain, starting from that with 10 up to 30 blocks.

OPENSTACKS In this domain, the objective is to ship a set of orders, each containing a specific set of products that must be completed. Here, BF23 requires that a valid plan completes all products following a specific production order. In PPLTL, we can easily specify the property "p_2 can become true only if p_1 was true in the past" with the formula $\mathsf{H}(p_2 \rightarrow \mathsf{Y}(\mathsf{O}(p_1)))$. Therefore, PPLTL formula

$$\mathsf{H}(made_{p_3} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_2}))) \wedge \mathsf{H}(made_{p_2} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_1})))$$

[6] Similarly, it is not easy to translate to LTL_f the pattern "h': $\alpha \,\mathsf{S}\, \beta$, where α or β has n nested S operators".

Table 4
Coverage, average runtime, average expanded nodes, and average plan length achieved by $\text{P4P}_{\mathcal{X}}$ and P4P_{adl}. For P4P_{adl}, we report in parentheses the average plan length considering the `check` actions added by the compilation. Averages are only among instances solved by both systems. Column I is the number of instances in a domain. Bold fonts are used for the best performers.

	Domain		Coverage		Runtime		Expanded Nodes		Plan length	
		I	$\text{P4P}_{\mathcal{X}}$	P4P_{adl}	$\text{P4P}_{\mathcal{X}}$	P4P_{adl}	$\text{P4P}_{\mathcal{X}}$	P4P_{adl}	$\text{P4P}_{\mathcal{X}}$	P4P_{adl}
TB15	ROVERS	7	**7**	**7**	**1.45**	5.25	**15.57**	26.71	**6.43**	13.86
TB15	BLOCKS	15	**15**	**15**	**1.40**	5.87	**29.80**	38.47	**10.53**	21.80
TB15	OPENSTACKS	10	**10**	8	**5.52**	60.08	**51.00**	58.00	**22.00**	45.50
BF23	ROVERS	40	**33**	6	**2.06**	74.11	**131.33**	4369.67	**13.50**	27.00
BF23	BLOCKS	21	**21**	20	**15.92**	155.87	**5998.95**	14061.60	**285.20**	543.00
BF23	OPENSTACKS	30	**7**	5	**11.88**	75.81	**99.80**	2025.60	**24.00**	49.00
BF23	ELEVATOR	29	**29**	9	**1.63**	101.30	**1358.44**	741117.00	**27.33**	48.78
	Total	152	122	70						

encodes that p_1 is made strictly before p_2, which in turn must be made before p_3. In addition, we require every order to be shipped with $\mathsf{O}(shipped_{order})$.

ROVERS The goal in this domain is to gather and communicate data about soil, rock, and images to the Earth using a set of rovers. BF23 enforces a total order over the communications of the data with PPLTL formula

$$\mathsf{O}(data_{soil} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{rock}))) \wedge \mathsf{O}(data_{rock} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{img}))) \wedge \mathsf{O}(data_{img}).$$

This goal uses the pattern $\mathsf{O}(p \wedge \mathsf{WY}(\mathsf{H}(\neg q)))$ to specify that p eventually becomes true while q has always been false in the past. In addition, when the rover reaches the lander, that rover must re-calibrate all cameras. For example, if the lander is at waypoint w_l and the rover r has 2 cameras, c_1 and c_2, we have PPLTL formula

$$((\neg at_{r,w_l} \,\mathsf{S}\, calibrated_{c_1}) \wedge (\neg at_{r,w_l} \,\mathsf{S}\, calibrated_{c_2})) \vee \mathsf{H}(\neg at_{r,w_l}).$$

In this case, we exploit the pattern $(\neg q \,\mathsf{S}\, p) \vee \mathsf{H}(\neg q)$ to specify that either q never becomes true or q remains false since p became true in the past.

ELEVATOR This domain models the problem of scheduling passengers in the use of an elevator. In BF23, half of the passengers have priority over the other half of regular passengers. We enforce that a priority passenger must be served before every regular one in PPLTL with

$$\mathsf{O}(served_{p_2} \wedge served_{p_3}) \wedge \mathsf{O}(served_{p_0} \wedge served_{p_1} \wedge \mathsf{WY}(\mathsf{H}(\neg served_{p_2} \wedge \neg served_{p_3}))).$$

Also, we model that no passenger may share the elevator with another passenger, and do so with formula

$$\mathsf{H}(boarded_{p_0} \rightarrow (\neg boarded_{p_1} \wedge \neg boarded_{p_2})).$$

7.1.2. Comparison between $\text{P4P}_{\mathcal{X}}$ and P4P_{adl}

Our first analysis compares $\text{P4P}_{\mathcal{X}}$ and P4P_{adl}, our two variants of P4P. Table 4 compares them in terms of coverage, runtime, number of nodes expanded during the search, and the length of the resulting plans. Overall, we observe a clear dominance of $\text{P4P}_{\mathcal{X}}$ over P4P_{adl}; the former encoding solves more instances, spending less time and expanding fewer nodes on average. This is expected, as LAMA natively handles axioms, and $\text{P4P}_{\mathcal{X}}$ introduces only conditional effects with atomic conditions. On the other hand, P4P_{adl} uses complex formulas in the definition of conditional effects, and this makes LAMA fail during the preprocessing of many instances. Moreover, the spurious `check` action introduces some noise in the planning process, as reflected by a higher number of expanded nodes. Regarding the length of solutions, if we consider the actual plan (obtained by removing the `check` actions), we observe that the two systems perform similarly.

7.1.3. Comparison between $\text{P4P}_{\mathcal{X}}$ and LTL_f encodings

Given that $\text{P4P}_{\mathcal{X}}$ dominates P4P_{adl}, we report the comparison between $\text{P4P}_{\mathcal{X}}$ and the two compilations for LTL_f goals. Table 5 reports the number of instances solved and compiled by all systems across all domains, while Table 6 shows a pairwise comparison in terms of average runtime, average compilation time, average number of expanded nodes, and average plan length. In terms of coverage, $\text{P4P}_{\mathcal{X}}$ performs equally to or better than LTLPoly and LTLExp in most cases.

Regarding the TB15 instances, $\text{P4P}_{\mathcal{X}}$ achieves the same coverage as LTLPoly (the best LTL_f-based compilation) but it is much faster in terms of average runtime: $\text{P4P}_{\mathcal{X}}$ is roughly one order of magnitude faster than LTLPoly; this seems to be justified by a great reduction in the number of expanded nodes, up to two orders of magnitude in OPENSTACKS and BLOCKS, which is somehow expected. Indeed, for each planning action taken, LTLPoly interleaves quite a complex automaton synchronization phase, from the

Table 5
Number of solved instances and number of compiled instances by P4P$_{\mathcal{X}}$, LTLExp, and LTLPoly across all domains.

Domain			Coverage			Number of compiled instances		
		I	P4P$_{\mathcal{X}}$	LTLExp	LTLPoly	P4P$_{\mathcal{X}}$	LTLExp	LTLPoly
TB15	ROVERS	7	**7**	6	**7**	**7**	**7**	**7**
	BLOCKS	15	**15**	8	**15**	**15**	14	**15**
	OPENSTACKS	10	**10**	6	**10**	**10**	**10**	**10**
BF23	ROVERS	40	**33**	22	6	**40**	**40**	**40**
	BLOCKS	21	**21**	1	1	**21**	7	**21**
	OPENSTACKS	30	7	**8**	5	**30**	22	22
	ELEVATOR	29	**29**	**29**	4	**29**	**29**	**29**
	Total		**122**	80	48	**152**	129	144

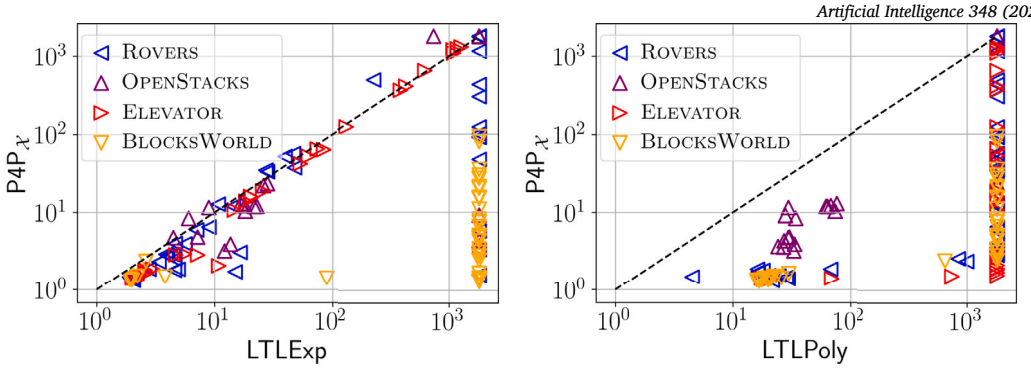
Table 6
Comparison of P4P$_{\mathcal{X}}$ vs. LTLExp and P4P$_{\mathcal{X}}$ vs. LTLPoly, over the intersection of instances solved by each pair of systems, in terms of average runtime, average expanded nodes, average plan length, and average compilation time. For LTLPoly, we report in parentheses the average plan length considering the spurious actions added by the compilation. Bold fonts are used for the best performers.

Domain		P4P$_{\mathcal{X}}$	LTLExp	P4P$_{\mathcal{X}}$	LTLPoly	
TB15	ROVERS	**1.43**	1.98	**1.45**	20.97	Runtime
	BLOCKS	**1.41**	13.13	**1.40**	20.12	
	OPENSTACKS	**6.11**	8.75	**5.74**	29.62	
BF23	ROVERS	35.36	**24.24**	**2.02**	325.32	
	BLOCKS	**2.36**	2.63	**2.36**	650.15	
	OPENSTACKS	**14.98**	21.66	**11.88**	68.86	
	ELEVATOR	231.83	**228.09**	**1.42**	212.55	
TB15	ROVERS	**12.67**	**12.67**	**15.57**	725.00	Expanded nodes
	BLOCKS	22.12	**21.75**	**29.80**	1042.93	
	OPENSTACKS	**52.67**	52.83	**52.20**	4390.40	
BF23	ROVERS	**7665.36**	7826.32	**124.33**	6438549.00	
	BLOCKS	**950.00**	**950.00**	**950.00**	8407916.00	
	OPENSTACKS	1905.00	**845.14**	**99.80**	1207764.80	
	ELEVATOR	**1712076.48**	1712090.79	**84.00**	4468320.75	
TB15	ROVERS	**5.33**	**5.33**	**6.43**	6.71 (95.00)	Plan length
	BLOCKS	7.50	**7.25**	10.53	**10.33** (193.40)	
	OPENSTACKS	**22.00**	**22.00**	**22.20**	**22.20** (380.80)	
BF23	ROVERS	43.68	**43.50**	13.67	**13.33** (451.33)	
	BLOCKS	**118.00**	**118.00**	118.00	**104.00** (6402.00)	
	OPENSTACKS	**31.43**	**31.43**	**24.00**	**24.00** (841.00)	
	ELEVATOR	**75.48**	**75.48**	16.00	**14.00** (448.50)	
TB15	ROVERS	**0.41**	1.01	**0.41**	1.67	Compilation time
	BLOCKS	**0.40**	8.43	**0.41**	4.22	
	OPENSTACKS	**0.46**	2.20	**0.46**	2.55	
BF23	ROVERS	**0.63**	5.62	**0.63**	17.51	
	BLOCKS	**0.46**	46.05	**0.47**	6.23	
	OPENSTACKS	**2.97**	31.98	**2.97**	166.57	
	ELEVATOR	**0.50**	3.47	**0.50**	46.96	

initial state to the goal. On average, in TB15 instances, 94.3% of actions in plans obtained with LTLPoly come from the automaton's synchronization phases.

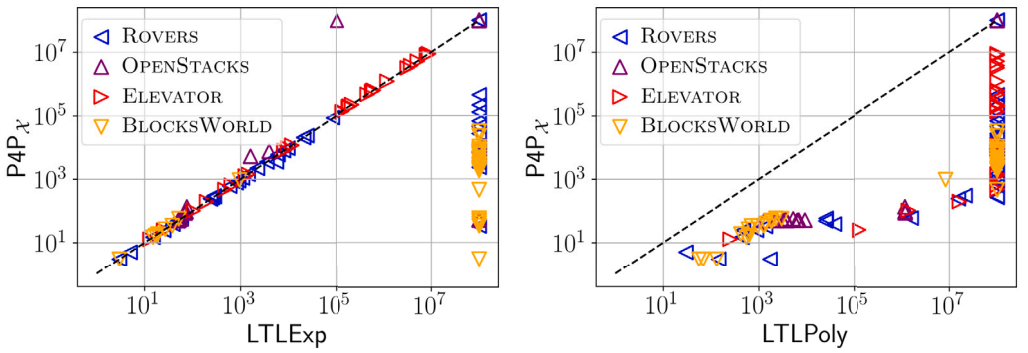
The situation is different if we look at the BF23 instances. Here, the best performing LTL_f compilation is LTLExp, which is superior to LTLPoly over all instances. The BF23 instances are of increasing sizes and have been constructed to be computationally more challenging. For example, in BLOCKS, the hardest instance in TB15 requires a 22-action plan, while the BF23 instances require up to 652 actions. This increase in plan length is particularly challenging for LTLPoly, as LAMA has to deal with many synchronization phases and has to do much more search to find a solution. This behavior is reflected in the average number of nodes expanded during the search: on average, LTLPoly expands up to five orders of magnitude more nodes than P4P$_{\mathcal{X}}$.

If we compare P4P$_{\mathcal{X}}$ and LTLExp, we observe that P4P$_{\mathcal{X}}$ is again the system that generally performs better. The only exception is for one instance of OPENSTACKS. LTLExp solves this instance in roughly 739s while P4P$_{\mathcal{X}}$ times out. By looking at the average number of expanded nodes, LAMA's search turned out to be slightly less informed with P4P$_{\mathcal{X}}$ in this domain, which leads to timing



(a) LTLExp vs P4P$_\mathcal{X}$ (runtime)

(b) LTLPoly vs P4P$_\mathcal{X}$ (runtime)



(c) LTLExp vs P4P$_\mathcal{X}$ (EN)

(d) LTLPoly vs P4P$_\mathcal{X}$ (EN)

Fig. 1. Pairwise comparison between P4P$_\mathcal{X}$ and LTLExp (left plots) and between P4P$_\mathcal{X}$ and LTLPoly (right plots) in terms of runtime (above).

out in that particular instance. For BLOCKS, P4P$_\mathcal{X}$ is instead much more effective than LTLExp, which manages to compile only 7 instances. This is because LTLExp uses automata-decomposition techniques to avoid the exponential blow-up for those formulas that can be decomposed. Intuitively, given a LTL$_f$ goal $\varphi = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$, LTLExp first computes a single (smaller) automaton for each formula ϕ_i of φ, and then forces the planner to meet the accepting condition of each sub-automaton. Although this technique works well in many cases, some formulas cannot be easily decomposed, and this is the case for the goals in BLOCKS. For example, the goal formula $\varphi_2 = \mathsf{F}(on_{2,1} \wedge \mathsf{X}(\mathsf{F}(on_{3,2} \wedge \mathsf{X}(\mathsf{F}(on_{4,3} \wedge \bigwedge_{j\in\{6,8,\ldots,n\}} \mathsf{X}(\mathsf{F}(on_{j,j-1}))))))))$ cannot be easily decomposed into smaller sub-automatons, making LTLExp fail during compilation. The same behavior also manifests in some of the TB15 instances, as pointed out by Torres and Baier [74]. The compilation stage is not an issue for P4P$_\mathcal{X}$, as the encoding time remains stable and consistent (Table 6) across all domains, and the compilation terminates successfully for every instance (Table 9). Instead, for some instances, compilation time seems to be an issue for the LTL$_f$ approaches. On average, both LTLExp and LTLPoly are up to two orders of magnitude slower than P4P$_\mathcal{X}$ during compilation.

Fig. 1 reports on a pairwise comparison P4P$_\mathcal{X}$ vs LTLExp and P4P$_\mathcal{X}$ vs LTLPoly over the number of expanded nodes and runtime, instance by instance. P4P$_\mathcal{X}$ is generally faster than LTLExp, except for 15 instances. Looking at our raw data, we observe that, for most of the instances, LTLExp spends much more time than P4P$_\mathcal{X}$ in compilation and slightly more time in evaluating a node of the search. Fig. 1 confirms that P4P$_\mathcal{X}$ allows for a lower runtime and leads to fewer expanded nodes than LTLPoly in every solved instance. Moreover, our data shows that P4P$_\mathcal{X}$ expands nodes more slowly than LTLPoly, and therefore the runtime advantage of P4P$_\mathcal{X}$ is related to the fact that P4P$_\mathcal{X}$ leads LAMA to do much less search than LTLPoly. We advocate this behavior to a heavier use of axioms, whose evaluation is needed at every node expansion in LAMA.

Regarding plan quality, we observed that all compilations yield *solution plans* to the original problems of similar length, making their overall performance the same in these terms.

7.2. FOND setting

In the FOND setting, to our knowledge, the state-of-the-art approach for handling LTL$_f$ goals is the compilation by Camacho et al. [27] called ltlfond2fond (ltlf2f). Similarly to what was done by Baier and McIlraith [13], ltlf2f explicitly computes an automaton representing the LTL$_f$ temporal goal. Also, in this case, the advantage of P4P seems clear: ltlf2f is exponential, while the compilation performed by Plan4Past is polynomial in the size of the PPLTL goals. Yet, from a practical standpoint, the impact of this advantage

in FOND planning is unclear. To this end, we tested $\mathsf{P4P}_{\mathcal{X}}$, $\mathsf{P4P}_{\mathsf{adl}}$, and ltlf2fond on a set of benchmarks and measured the number of problems solved (Coverage), the time taken to get a solution (compilation plus search), and the size of the policies (number of state-action pairs).

To evaluate the three encodings, we used two state-of-the-art FOND planners: PRP [63] and Paladinus [42]. Paladinus fully supports axioms, but not disjunctive conditional effects. Conversely, PRP supports conditional effects with disjunctive conditions but does not support axioms. Hence, since $\mathsf{P4P}_{\mathsf{adl}}$ requires disjunctive conditional effects and no axioms, while $\mathsf{P4P}_{\mathcal{X}}$ requires conjunctive conditional effects and axioms, we use $\mathsf{P4P}_{\mathsf{adl}}$ with PRP ($\mathsf{P4P}^{\mathsf{PRP}}_{\mathsf{adl}}$ for short), and Paladinus with $\mathsf{P4P}_{\mathcal{X}}$ ($\mathsf{P4P}^{\mathsf{Pal}}_{\mathcal{X}}$ for short). Moreover, the problems encoded by ltlf2fond are supported by FOND planners with conditional effects, but our findings show that ltlf2fond performs significantly better with PRP. Hence, we only present the results of ltlf2fond with PRP.

7.2.1. Benchmark domains

Our benchmark suite features the FOND domains ROVERS, BLOCKS, and COFFEE used in [27]. We tested all compilations on the same instances by Camacho et al. [27] (C17 for short). The temporally extended goals included in the C17 instances were originally specified in LTL_f, and therefore, for comparison reasons, we manually translated them to PPLTL. We observed that all C17 instances were trivially solved by the three systems. Therefore, to study the scalability of the compilations, we generated a new set of instances of increasing dimensions for each domain (BF23 for short). The C17 instances are publicly available, while our newly generated instances are described below, along with other information about the domain and the intuition behind some translations from LTL_f to PPLTL. We report a comprehensive translation between the two formalisms in the Appendix (Table 11).

BLOCKS In the FOND version of the BLOCKS domain, a block may slip from the arm, falling on the table. Starting from the 24 C17 instances, we generated bigger problems considering four types of formula employed by Camacho et al. [27]:

$$\mathsf{F}(a)\,\mathsf{U}\,(\mathsf{F}(b_1)\wedge\mathsf{F}(b_2)\wedge\ldots\wedge\mathsf{F}(b_n)) \tag{1}$$

$$(\mathsf{F}(b_1)\vee\mathsf{F}(b_2)\vee\ldots\vee\mathsf{F}(b_n))\,\mathsf{U}\,\mathsf{F}(a) \tag{2}$$

$$\mathsf{F}(\mathsf{F}(b_1)\wedge\mathsf{F}(b_2)\wedge\ldots\wedge\mathsf{F}(b_n)\wedge a) \tag{3}$$

$$(a\,\mathsf{U}\,\mathsf{F}(b_1))\wedge(a\,\mathsf{U}\,\mathsf{F}(b_2))\wedge\ldots\wedge(a\,\mathsf{U}\,\mathsf{F}(b_n)) \tag{4}$$

where atoms a and b_i represent whether a block is on the table or on top of another block. Formulas (1) and (4) require each atom b_i to eventually be true in some state, while (2) expresses that a eventually becomes true. In all three formulas, the "Until" (U) does not add any semantic meaning (i.e., requirements to be satisfied by the trace) to the formula. For this case, we can obtain the equivalent PPLTL formulation by simply swapping the future temporal operators with the past ones. For instance, in PPLTL, we write formula (4) as $(a\,\mathsf{S}\,\mathsf{O}(b_1))\wedge(a\,\mathsf{S}\,\mathsf{O}(b_2))\wedge\ldots\wedge(a\,\mathsf{S}\,\mathsf{O}(b_n))$. Formula (3) expresses that "there exists a state where a is true, and each atom b_i is true in the same state or future states". In this case, the translation in PPLTL is not straightforward: we used the semantically equivalent formula $\mathsf{O}(b_1\wedge\mathsf{O}(a))\wedge\mathsf{O}(b_2\wedge\mathsf{O}(a))\wedge\ldots\wedge\mathsf{O}(b_n\wedge\mathsf{O}(a))$. Each temporal goal is used over 10 instances obtained by adding more blocks to the table, resulting in 40 new instances.

To challenge the different compilations, for this domain, we also designed a new type of formula, i.e., $seq_{i,j}$ where i is the number of blocks, and j is the number of towers to build in a specific order. For example, formula $seq_{3,2}$ requires building two towers made of three blocks. In the end, such blocks must be on the table (t). In PPLTL this is expressed as:

$$\mathsf{O}(on_{a,t}\wedge on_{b,t}\wedge on_{c,t}\wedge\mathsf{YO}(on_{c,b}\wedge on_{b,a}\wedge\mathsf{YO}(on_{a,c}\wedge on_{c,b}))).$$

We generated a new instance with a goal $seq_{i,j}$ for $i=3,4$ and with $j=1,\ldots,i!$, for a total of 30 "seq" problems. Therefore, the total number of instances for BLOCKS sums up to 94 instances, 24 original instances from [27], 40 instances that we generated by scaling the temporal goals from [27], and 30 "seq" instances.

ROVERS Similarly to the deterministic version of this domain, the goal is to gather data about soil, rocks, and images of a planet by planning the activities of one or more rovers. In the FOND variant of ROVERS, some actions may cause a loss of information or errors during sampling operations. For example, gathering soil data may lead to acquiring rock data instead, or communicating an image to the lander may result in the rover losing the photograph. We used the 9 instances in C17 and generated other larger instances with 4 types of formulas:

$$\mathsf{F}(g_1)\wedge\mathsf{F}(g_2)\wedge\ldots\wedge\mathsf{F}(g_n) \tag{1}$$

$$\mathsf{F}(\mathsf{F}(g_1)\wedge\mathsf{F}(g_2)\wedge\ldots\wedge\mathsf{F}(g_n)) \tag{2}$$

$$\mathsf{F}(\mathsf{F}(g_1)\vee\mathsf{F}(g_2)\vee\ldots\vee\mathsf{F}(g_n)) \tag{3}$$

$$\mathsf{G}(\mathsf{F}(\mathsf{G}(g_1)\vee\mathsf{G}(g_2)\vee\ldots\vee\mathsf{G}(g_n))) \tag{4}$$

where every g_i is an atom representing the completion of a task, i.e., the communication of data about soil, rock, or image. The first two formulas require that each task is accomplished in some state of the trace induced by the execution of the policy. Formula (3) requires that at least one task is eventually completed, while the last formula requires that all tasks are satisfied in the last state. The translation for formulas (1), (2), and (3) is straightforward; we can simply swap future operators with their past counterparts. For

Table 7
Coverage achieved by all systems across all domains. Bold fonts are for the best performers.

	Domain	I	$\text{P4P}^{\text{Pal}}_{\mathcal{X}}$	$\text{P4P}^{\text{PRP}}_{\text{adl}}$	ltlf2f
BF23	BLOCKS-1	10	**10**	**10**	1
	BLOCKS-2	10	**10**	**10**	**10**
	BLOCKS-3	10	**10**	6	2
	BLOCKS-4	10	**10**	6	4
	BLOCKS-*seq*	30	8	**30**	20
	COFFEE-1	15	5	**12**	4
	COFFEE-2	15	5	**11**	2
	COFFEE-3	15	0	**2**	1
	COFFEE-4	15	**15**	**15**	**15**
	ROVERS-1	40	3	**24**	10
	ROVERS-2	40	4	**23**	7
	ROVERS-3	40	19	37	**40**
	ROVERS-4	40	16	**23**	7
C17	BLOCKS	24	**24**	**24**	**24**
	COFFEE	10	**10**	**10**	**10**
	ROVERS	9	**9**	**9**	**9**
	Total		158	**252**	166

instance, we write formula (2) in PPLTL as $\mathsf{O}(\mathsf{O}(g_1) \wedge \mathsf{O}(g_2) \wedge \ldots \wedge \mathsf{O}(g_n))$. The last formula requires one of the goals to be achieved in the last state; this could be represented in PPLTL without using any temporal operator. However, for the sake of a fair comparison, we translated such LTL_f formula into a PPLTL formula with the same number of nested temporal operators, i.e.,

$$\mathsf{H}(\mathsf{H}(\mathsf{H}(g_1) \vee \mathsf{H}(g_2) \vee \ldots \vee \mathsf{H}(g_n))) \vee (g_1 \vee g_2 \vee \ldots \vee g_n).$$

These constraints are used over 40 propositional problems of ROVERS from the 5th IPC, giving us 160 instances, which, added to the 9 C17 instances, gives a total of 169 instances.

COFFEE A robot has to prepare coffee in a kitchen and deliver it to different offices. The robot can move between adjacent offices, and delivering the coffee could non-deterministically result in the mug being accidentally spilled, forcing the robot to prepare another coffee. We considered the 10 problems from C17 and generated further larger instances with the following types of temporally extended goals:

$$\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n}) \tag{1}$$

$$\mathsf{F}(\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n})) \tag{2}$$

$$\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n}) \wedge \mathsf{G}(R_{o_i} \Rightarrow \mathsf{X}(\neg R_{o_j})) \wedge \ldots \tag{3}$$

$$\mathsf{X}(\mathsf{X}(\ldots \mathsf{X}(R_{kitchen}))). \tag{4}$$

The first two formulas are satisfied by delivering the coffee to all offices. Formula (3) is as formula (1) but with the requirement that if the robot is in the office o_i, then in the next state, it cannot be in office o_j. This in PPLTL can be easily expressed with $\mathsf{H}(\mathsf{Y}(R_{o_i}) \Rightarrow \neg(R_{o_j})) \wedge \neg R_{o_i}$. The last temporally extended goal requires the robot to be in the kitchen in the k+1-th state, where k is equal to the number of nested X operators. In PPLTL, this can be captured by $\mathsf{O}(R_{kitchen} \wedge \mathsf{Y}(\mathsf{Y}(\ldots \mathsf{Y}(start))))$, where the number of Y is equal to k. We generated a new set of instances, 15 for each type of formula, increasing the number of offices. Formulas (1), (2), and (3) scale with the number of offices, while the formula of type (4) scales with k ranging from 7 to 21. In total, we have 70 instances; 60 generated by us and 10 from C17.

7.2.2. Results

Table 7 reports the overall coverage for all systems, while Table 8 compares $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ with ltlf2f and $\text{P4P}^{\text{PRP}}_{\text{adl}}$ with ltlf2f in terms of average runtime, and average size of solution policies. The BF23 instances are indicated with the pattern "domain-formula" (e.g., BLOCKS-1 refers to blocksworld with goals of type (1)). $\text{P4P}^{\text{PRP}}_{\text{adl}}$ achieves the highest coverage, followed by ltlf2f and $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$. The C17 instances are small and, as we can see from Table 8, trivially solved by all systems in less than 11 seconds on average. Conversely, the BF23 instances are larger and designed to challenge the compilations. Indeed, in BF23, we observe that $\text{P4P}^{\text{PRP}}_{\text{adl}}$ solves 81.7% more instances than $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ and 69.9% more instances than ltlf2f, yet the three systems compared show complementary behaviors. For example, ltlf2f outperforms both P4P compilation in ROVERS-3. Such results reflect the weaknesses of each system. P4P_{adl} and $\text{P4P}_{\mathcal{X}}$ build a factorized version of the DFA either introducing possibly complex conditional effects or using axioms, while ltlf2f uses an explicit representation of the DFA. Comparing the factorized version of the DFA to an explicit representation of the DFA is key to understanding the different behaviors. For example, all C17 instances feature a compact DFA due to the limited size of goal formulas.

Table 8
Comparison between $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ and ltlf2f and between $\text{P4P}^{\text{PRP}}_{\text{adl}}$ and ltlf2f in terms of average runtime and the average size of solution policies. The policy size is reported without counting spurious actions added by compilations. Bold fonts are for the best performers.

	Domain	Runtime				Policy size			
		$\text{P4P}^{\text{Pal}}_{\mathcal{X}}$	ltlf2f	$\text{P4P}^{\text{PRP}}_{\text{adl}}$	ltlf2f	$\text{P4P}^{\text{Pal}}_{\mathcal{X}}$	ltlf2f	$\text{P4P}^{\text{PRP}}_{\text{adl}}$	ltlf2f
BF23	BLOCKS-1	**4.53**	15.39	**4.71**	15.39	**3.00**	**3.00**	4.00	**3.00**
	BLOCKS-2	19.77	**6.33**	24.13	**6.33**	**1.00**	2.00	**2.00**	**2.00**
	BLOCKS-3	**3.35**	12.22	**7.57**	12.22	**2.00**	3.00	5.00	**3.00**
	BLOCKS-4	**6.57**	32.85	36.04	**32.85**	**3.00**	**3.00**	**3.00**	**3.00**
	BLOCKS-*seq*	54.94	**4.14**	**6.16**	12.72	14.00	**13.75**	42.60	**38.40**
	COFFEE-1	**75.91**	126.17	**7.01**	126.17	**50.50**	56.00	**56.00**	**56.00**
	COFFEE-2	**4.46**	19.27	**5.52**	19.27	**43.50**	48.00	**48.00**	**48.00**
	COFFEE-4	**2.62**	2.97	21.30	**2.97**	**14.47**	15.00	**15.00**	**15.00**
	ROVERS-1	28.93	**5.23**	**11.61**	50.43	21.33	**16.67**	**28.40**	30.50
	ROVERS-2	389.45	**3.79**	**7.93**	9.94	25.00	**15.75**	**21.00**	24.29
	ROVERS-3	108.75	**3.63**	161.81	**26.74**	5.89	**5.32**	6.76	**5.84**
	ROVERS-4	8.87	**4.01**	19.33	**4.01**	**5.29**	**5.29**	**5.29**	**5.29**
C17	BLOCKS	**2.31**	2.39	5.37	**2.39**	**5.21**	6.75	7.71	**6.75**
	COFFEE	10.86	**2.75**	4.72	**2.75**	**12.33**	13.44	14.22	**13.44**
	ROVERS	8.54	**2.53**	6.28	**2.53**	12.56	**9.11**	**9.11**	**9.11**

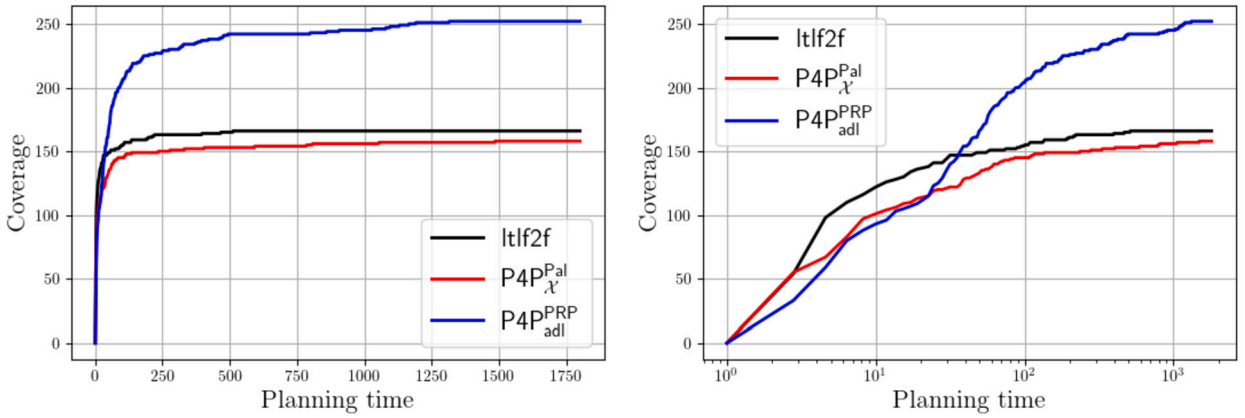


Fig. 2. Survival plot in linear (lhs) and logarithmic (rhs) scale.

Other examples are BLOCKS-2 and ROVERS-3, where the LTL_f goal can be represented by a 2-state minimal DFA. Indeed, in all of these domains ltlf2f performs remarkably well, solving all instances within seconds. On the contrary, both versions of P4P exploit the syntactic structure of the input goal formula. When the DFA size increases (in the worst case becoming exponentially larger with respect to the syntactic size of the formula), P4P outperforms ltlf2f. This is the case, for example, of formulas from BLOCKS-3, BLOCKS-4, ROVERS-1, ROVERS-2, COFFEE-1 and COFFEE-2.

Looking at other domains, we observe that in ROVERS-4, $\text{P4P}^{\text{PRP}}_{\text{adl}}$ and $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ perform well, while in many instances ltlf2f crashes during the automaton computation. COFFEE-4 is easily solved by all systems, while COFFEE-3 proved to be challenging for every compilation. Lastly, in BLOCKS-seq $\text{P4P}^{\text{PRP}}_{\text{adl}}$ and ltlf2f both perform better than $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$.

Table 8 compares $\text{P4P}_{\mathcal{X}}$ with ltlf2f and $\text{P4P}^{\text{PRP}}_{\text{adl}}$ with ltlf2f in terms of average runtime and average policy size. On average, both $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ and $\text{P4P}^{\text{PRP}}_{\text{adl}}$ are faster than ltlf2f in 7 domains. To shed some light on this aspect, Fig. 2 displays coverage over time for all systems. ltlf2f dominates the other compilations at the start (visible in the logarithmic scale plot on the right). Instead, $\text{P4P}^{\text{PRP}}_{\text{adl}}$ solves more instances than ltlf2f after 36.4 seconds. In an instance-by-instance comparison (Fig. 3, left), we observe that ltlf2f solves many problems (112) before $\text{P4P}^{\text{PRP}}_{\text{adl}}$ does. $\text{P4P}^{\text{PRP}}_{\text{adl}}$ introduces complex formulas in conditional effects and goals, and we observed that the preprocessing of PRP often exceeds by orders of magnitude the compilation time. Overcoming this issue without introducing axioms is an open question for future work. The pairwise comparison of ltlf2f with $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ in terms of runtime (Fig. 3, right), shows that ltlf2f is faster than $\text{P4P}^{\text{Pal}}_{\mathcal{X}}$ in most instances, and this behavior can be attributed to Paladinus being slower than PRP for those instances. Finally, all systems computed policies of comparable dimensions.

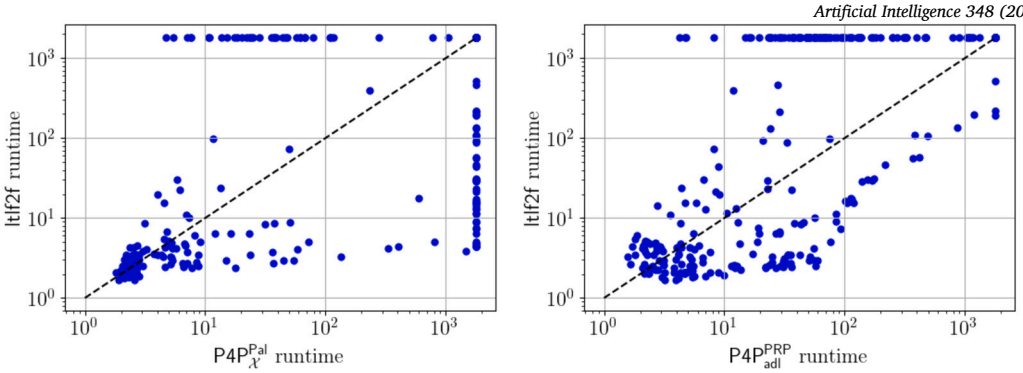


Fig. 3. Run-Time comparison of ltlf2f vs P4P$^{\text{PRP}}_{\text{adl}}$ and P4P$^{\text{Pal}}_{\chi'}$.

8. Related work

Planning for temporally extended goals has a long tradition in AI Planning, including pioneering work in the late '90s [5,7,6,9], work on planning via Model Checking [31,36,51], and work on declarative and procedural constraints [13,12,19]. Also, recent works focus on handling the set of temporally extended goals defined by PDDL3 [49], interpreting them as preferences [38,58,18,28,67], state-trajectory constraints [22,24], or over action sequences [23].

PPLTL has been used in the past to express non-Markovian rewards in Markov decision processes (MDPs). In this context, Bacchus et al. [5] proposed an encoding that has exponential complexity in the input size, while Bacchus et al. [6] proposed an encoding that uses a linear amount of propositional variables; such variables are meant to map a non-Markovian MDP into a standard MDP. As in our case, Bacchus et al. (1997)'s [6] construction uses the fixpoint equivalence of the *since* operator to identify the set of PPLTL subformulas to be monitored, and introduces one propositional "yesterday" variable for each of them. These variables are used to evaluate the truth of the PPLTL specification and are updated only considering the current and the previous step. We extended Bacchus et al. (1997)'s [6] work specifically along two dimensions. Firstly, we provide full spelled proofs for the theoretical construction in Section 3. Secondly, we specialize the formulation in the context of classical and FOND planning. This operationalisation is not direct and there are many ways to do so. In our work, we propose two encodings that exploit new specific features of modern planning formalisms, namely conditional effects and PDDL axioms. Then, differently from Bacchus et al. [6] whose work remains at a more conceptual level of abstraction, we provide empirical evidence on the practical benefits of our approaches. Specifically, we conduct an experimental analysis over a variety of deterministic and non-deterministic domains, studying the upside and the downside of the two encoding variants among each other, and against existing state-of-the-art approaches for handling temporally extended goals.

Beyond these works, past temporal logic has also been advocated for non-Markovian models in reasoning about actions [45], for preferred explanations in the context of dynamical diagnosis [71], for normative properties in multi-agent systems [40,59,2], for synthesis [30], and in the context of plan/goal recognition [41].

The most closely related line of research concerns handling LTL$_f$ temporally extended goals via encoding to classical or FOND planning. In the deterministic setting, handling LTL$_f$ goals is PSPACE-complete, as is the case for PPLTL goals. This is because it is sufficient to compute the cross-product between the DFA for the planning domain and the non-deterministic automaton (NFA) for the goal formula, and then check non-emptiness on the resulting automaton returning a plan if any [13,14,37]. The works by Baier and McIlraith [13,14] adopt this approach for encoding LTL$_f$ goals into classical planning domains. Since the NFA representing a LTL$_f$ formula can be exponential in the size of the formula, the overall approach can yield a planning problem with an exponential amount of fluents. Our approach, instead, uses a linear number of fluents in the size of the PPLTL formula. Similarly to our approach, Baier and McIlraith [13] also uses derived predicates and axioms to simplify the action schema. On the other hand, Baier and McIlraith's schema supports first-order LTL$_f$ formulas while our encoding only works with PPLTL formulas over propositional atoms.

In the context of FOND domains, where actions have nondeterministic outcomes, the standard approach for handling LTL$_f$ goals consists of computing the cross product between the DFA representing the LTL$_f$ formula and the DFA representing the planning domain [35]. In this case, LTL$_f$ goals must be (implicitly or explicitly) translated into a DFA, which is double exponential in the worst case.[7] This makes FOND planning for LTL$_f$ goals 2EXPTIME-complete [35], while FOND planning for standard reachability is EXPTIME-complete [69]. Practically, this approach has been presented in [27] and [26]. Hence, the only existing encoding for LTL$_f$ goals in FOND is worst-case exponential in the size of the goal formula. Instead, our approach remains polynomial in the size of the PPLTL goal.

Given our encoding, it is important to consider under which conditions PPLTL is preferable to LTL$_f$, or vice versa, for expressing temporally extended goals. As shown in [33], every PPLTL formula can, in principle, be translated into an equivalent LTL$_f$ formula, and vice versa. However, this operation is worst-case triply exponential in the size of the input formula, and to our knowledge, no existing tool implements this transformation. Furthermore, recent work indicates that even translating a fragment of LTL$_f$ into PPLTL

[7] Note that the same operation for PPLTL formulas can be performed in single exponential time.

can incur in a worst-case exponential blowup [4]. It is therefore reasonable to conclude that if a goal can be easily expressed in, for example, LTL_f but not in PPLTL (or vice versa), then we should adopt that formulation and employ the corresponding planning tools. More interesting is the case where a temporally extended goal can be easily and compactly expressed in both PPLTL and LTL_f. In this situation, we argue that the PPLTL formulation is preferable, as it enables the use of our polynomial and plan-size preserving encoding. Nonetheless, we believe that the PPLTL and LTL_f encodings effectively complement each other, since our experiments did not reveal a clear dominance of one approach over the other.

Related are the works that exploit the fixpoint characterization of LTL. In the area of AI planning, this characterization, referred to as "formula progression", has been used as an alternative to automata construction in the TLPlan planner [7,8] for handling temporally extended goals. This is perhaps one of the most influential works in this area. A similar approach has been adopted for handling control knowledge expressed in LTL [9]. Formula progression has also been used to develop LTL-aware heuristics in the context of planning with probabilistic LTL constraints [17,61], and planning with temporally extended goals and preferences expressed using other variants of LTL [11,19]. The notion of progression has also been employed in the context of MDPs with rewards in LTL and PPLTL [5,6,72]. The fixpoint characterization has also been used in the MetateM approach [16] for programming in temporal logic and in the context of LTL_f synthesis [34].

Other works focus on classical planning with LTL goals interpreted over infinite traces [65,66]. These approaches resort to nondeterministic automata construction and the notion of *lasso* plans to deal with the temporal specifications. PPLTL expresses properties over finite traces only, and understanding how to extend our encoding to infinite executions is a direction for future work.

Our findings related to an efficient handling and evaluation of PPLTL in [20] have led to new ongoing research. For instance, our work on automated planning for PPLTL goals inspired [47] to develop an effective technique to solve the problem of reactive synthesis for DECLARE patterns via symbolic automata. Moreover, our encoding has been used for converting natural language instructions to LTL formulas [44]. Finally, our encoding technique could definitely inspire new approaches in the area of synthesis for temporal logic formulas [53,15].

9. Conclusions

We have studied the problem of planning for PPLTL goals in both deterministic and FOND domains. Specifically, we have shown that planning for PPLTL goals can be encoded into planning for reachability goals with minimal overhead and without additional spurious actions. Handling PPLTL goals is remarkably simple and elegant, given the direct mapping between the theoretical formulation and the compilation schema without sacrificing efficiency. Moreover, we practically show that Plan4Past can solve a wider range of problems compared to the state-of-the-art encodings for LTL_f goals in classical and FOND domains. The general theoretical and practical advantages of PPLTL observed so far suggest that PPLTL may definitely become a promising candidate to be the mainstream language to express temporal goals in planning. Future work concerns developing planners that can *natively* handle PPLTL goals, exploring PPLTL-aware heuristics, and handling more expressive formalisms such as Pure-Past Linear Dynamic Logic [33].

CRediT authorship contribution statement

Luigi Bonassi: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Giuseppe De Giacomo:** Writing – review & editing, Visualization, Supervision, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Marco Favorito:** Writing – review & editing, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Francesco Fuggitti:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alfonso Emilio Gerevini:** Writing – review & editing, Visualization, Supervision, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Enrico Scala:** Writing – review & editing, Visualization, Supervision, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially supported by the projects ERC Advanced Grant WhiteMech (No. 834228), MUR PRIN project RIPER (No. 20203FFYLK), and projects SERICS (PE00000014), cascade funding call AI-RESCUE, and FAIR (PE0000013), cascade funding call ResilientPlans, under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Appendix A

This appendix provides all temporal goals, formulated in PPLTL and LTL_f, used in our experimental analysis.

Table 9
Formulas employed in deterministic benchmarks.

	Formula	Logic
BLOCKS	$\mathsf{O}(on_{1,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,3} \wedge \mathsf{Y}(\mathsf{O}(\ldots \wedge \mathsf{Y}(\mathsf{O}(on_{n-1,n})))))))$	PPLTL
	$\mathsf{F}(on_{n-1,n} \wedge \mathsf{X}(\mathsf{F}(on_{n-2,n-1} \wedge \mathsf{X}(\mathsf{F}(\ldots \wedge \mathsf{X}(\mathsf{F}(on_{1,2})))))))$	LTL_f
	$\bigwedge_{j\in\{6,8,\ldots,n\}} \mathsf{O}(on_{j,j-1} \wedge \mathsf{Y}(\mathsf{O}(on_{4,3} \wedge \mathsf{Y}(\mathsf{O}(on_{3,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,1})))))))$	PPLTL
	$\mathsf{F}(on_{2,1} \wedge \mathsf{X}(\mathsf{F}(on_{3,2} \wedge \mathsf{X}(\mathsf{F}(on_{4,3} \wedge \bigwedge_{j\in\{6,8,\ldots,n\}} \mathsf{X}(\mathsf{F}(on_{j,j-1})))))))$	LTL_f
OPENSTACKS	$\mathsf{H}(made_{p_3} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_2}))) \wedge \mathsf{H}(made_{p_2} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_1})))$	PPLTL
	$(made_{p_2})\,\mathsf{R}\,(\neg made_{p_3}) \wedge (made_{p_1})\,\mathsf{R}\,(\neg made_{p_2})$	LTL_f
	$\mathsf{O}(shipped_{order})$	PPLTL
	$\mathsf{F}(shipped_{order})$	LTL_f
ROVERS	$\mathsf{O}(data_{soil} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{rock}))) \wedge \mathsf{O}(data_{rock} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{image}))) \wedge \mathsf{O}(data_{image})$	PPLTL
	$(\neg data_{rock})\,\mathsf{U}\,(data_{soil}) \wedge (\neg data_{image})\,\mathsf{U}\,(data_{rock}) \wedge \mathsf{F}(data_{image})$	LTL_f
	$((\neg at_{r,w_l}\,\mathsf{S}\,calibrated_{c_1}) \wedge (\neg at_{r,w_l}\,\mathsf{S}\,calibrated_{c_2})) \vee \mathsf{H}(\neg at_{r,w_l})$	PPLTL
	$\mathsf{G}(at_{r,w_l} \rightarrow (\mathsf{F}(calibrated_{c_1}) \wedge \mathsf{F}(calibrated_{c_2})))$	LTL_f
ELEVATOR	$\mathsf{O}(served_{p_2} \wedge served_{p_3}) \wedge \mathsf{O}(served_{p_0} \wedge served_{p_1} \wedge \mathsf{WY}(\mathsf{H}(\neg served_{p_2} \wedge \neg served_{p_3})))$	PPLTL
	$\mathsf{F}(served_{p_2} \wedge served_{p_3}) \wedge (\neg served_{p_2} \wedge \neg served_{p_3})\,\mathsf{U}\,(served_{p_0} \wedge served_{p_1})$	LTL_f
	$\mathsf{H}(boarded_{p_0} \rightarrow (\neg boarded_{p_1} \wedge \neg boarded_{p_2}))$	PPLTL
	$\mathsf{G}(boarded_{p_0} \rightarrow (\neg boarded_{p_1} \wedge \neg boarded_{p_2}))$	LTL_f

Table 10
TB15 Classes of formulas.

Formula class	PPLTL	LTL_f
a	$\alpha\,\mathsf{S}\,\bigwedge_{i=1}^{n} \mathsf{O}(\beta_i)$	$\alpha\,\mathsf{U}\,\bigwedge_{i=1}^{n} \mathsf{F}(\beta_i)$
b	$(\bigvee_{i=1}^{n} \mathsf{O}(p_i))\,\mathsf{S}\,\mathsf{O}(q)$	$(\bigvee_{i=1}^{n} \mathsf{F}(p_i))\,\mathsf{U}\,\mathsf{F}(q)$
c	$\bigwedge_{i=1}^{n} \mathsf{O}(\beta_i \wedge \mathsf{O}(\alpha))$	$\mathsf{F}(\alpha \wedge \bigwedge_{i=1}^{n} \mathsf{F}(\beta_i))$
d	$\bigwedge_{i=1}^{n} \mathsf{O}(\beta_i \wedge \mathsf{WY}(\mathsf{H}(\alpha)))$	$\bigwedge_{i=1}^{n} (\alpha\,\mathsf{U}\,\beta_i)$
e	$\mathsf{O}(\bigwedge_{i=1}^{n} \mathsf{O}(\beta_i))$	$\mathsf{F}(\bigwedge_{i=1}^{n} \mathsf{F}(\beta_i))$
f	$\bigwedge_{i=1}^{n} \mathsf{O}(p_i)$	$\bigwedge_{i=1}^{n} \mathsf{F}(p_i)$
g	$\bigwedge_{i=1}^{3} \mathsf{O}(p_i) \wedge \mathsf{O}(r_i \wedge \mathsf{WY}(\mathsf{H}(q_i)))$	$\bigwedge_{i=1}^{3} \mathsf{F}(p_i) \wedge \bigwedge_{i=1}^{n-3} (q_i\,\mathsf{U}\,r_i)$
i	$\mathsf{O}(\bigvee_{i=1}^{n} \mathsf{O}(\beta_i))$	$\mathsf{F}(\bigvee_{i=1}^{n} \mathsf{F}(\beta_i))$
j	$\bigvee_{i=1}^{n} \beta_i \wedge (\bigvee_{i=1}^{n} \mathsf{O}(\beta_i))$	$\mathsf{F}(\bigvee_{i=1}^{n} \mathsf{G}(\beta_i))$

Table 11
Formulas employed in non-deterministic benchmarks.

	Formula	Logic
BLOCKS (1,2,3,4, seq)	$\mathsf{O}(a)\,\mathsf{S}\,(\mathsf{O}(b_1) \wedge \mathsf{O}(b_2) \wedge \ldots \wedge \mathsf{O}(b_n))$	PPLTL
	$\mathsf{F}(a)\,\mathsf{U}\,(\mathsf{F}(b_1) \wedge \mathsf{F}(b_2) \wedge \ldots \wedge \mathsf{F}(b_n))$	LTL_f
	$(\mathsf{O}(b_1) \vee \mathsf{O}(b_2) \vee \ldots \vee \mathsf{O}(b_n))\,\mathsf{U}\,\mathsf{O}(a)$	PPLTL
	$(\mathsf{F}(b_1) \vee \mathsf{F}(b_2) \vee \ldots \vee \mathsf{F}(b_n))\,\mathsf{U}\,\mathsf{F}(a)$	LTL_f
	$\mathsf{O}(b_1 \wedge \mathsf{O}(a)) \wedge \mathsf{O}(b_2 \wedge \mathsf{O}(a)) \wedge \ldots \wedge \mathsf{O}(b_n \wedge \mathsf{O}(a))$	PPLTL
	$\mathsf{F}(\mathsf{F}(b_1) \wedge \mathsf{F}(b_2) \wedge \ldots \wedge \mathsf{F}(b_n) \wedge a)$	LTL_f
	$(a\,\mathsf{S}\,\mathsf{O}(b_1)) \wedge (a\,\mathsf{S}\,\mathsf{O}(b_2)) \wedge \ldots \wedge (a\,\mathsf{S}\,\mathsf{O}(b_n))$	PPLTL
	$(a\,\mathsf{S}\,\mathsf{O}(b_1)) \wedge (a\,\mathsf{S}\,\mathsf{O}(b_2)) \wedge \ldots \wedge (a\,\mathsf{U}\,\mathsf{O}(b_n))$	LTL_f
	$\mathsf{O}(on_{a,t} \wedge on_{b,t} \wedge on_{c,t} \wedge \mathsf{YO}(on_{c,b} \wedge on_{b,a} \wedge \mathsf{YO}(on_{a,c} \wedge on_{c,b})))$	PPLTL
	$\mathsf{F}(on_{a,c} \wedge on_{c,b} \wedge \mathsf{XF}(on_{c,b} \wedge on_{b,a} \wedge \mathsf{XF}(on_{a,t} \wedge on_{b,t} \wedge on_{c,t})))$	LTL_f
ROVERS (1,2,3,4)	$\mathsf{O}(g_1) \wedge \mathsf{O}(g_2) \wedge \ldots \wedge \mathsf{O}(g_n)$	PPLTL
	$\mathsf{F}(g_1) \wedge \mathsf{F}(g_2) \wedge \ldots \wedge \mathsf{F}(g_n)$	LTL_f
	$\mathsf{O}(\mathsf{O}(g_1) \wedge \mathsf{O}(g_2) \wedge \ldots \wedge \mathsf{O}(g_n))$	PPLTL
	$\mathsf{F}(\mathsf{F}(g_1) \wedge \mathsf{F}(g_2) \wedge \ldots \wedge \mathsf{F}(g_n))$	LTL_f
	$\mathsf{O}(\mathsf{O}(g_1) \vee \mathsf{O}(g_2) \vee \ldots \vee \mathsf{O}(g_n))$	PPLTL
	$\mathsf{F}(\mathsf{F}(g_1) \vee \mathsf{F}(g_2) \vee \ldots \vee \mathsf{F}(g_n))$	LTL_f
	$\mathsf{H}(\mathsf{H}(\mathsf{H}(g_1) \vee \mathsf{H}(g_2) \vee \ldots \vee \mathsf{H}(g_n))) \vee (g_1 \vee g_2 \vee \ldots \vee g_n)$	PPLTL
	$\mathsf{G}(\mathsf{F}(\mathsf{G}(g_1) \vee \mathsf{G}(g_2) \vee \ldots \vee \mathsf{G}(g_n)))$	LTL_f
COFFEE (1,2,3,4)	$\mathsf{O}(C_{o_1}) \wedge \ldots \wedge \mathsf{O}(C_{o_n})$	PPLTL
	$\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n})$	LTL_f
	$\mathsf{O}(\mathsf{O}(C_{o_1}) \wedge \ldots \wedge \mathsf{O}(C_{o_n}))$	PPLTL
	$\mathsf{F}(\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n}))$	LTL_f
	$\mathsf{O}(C_{o_1}) \wedge \ldots \wedge \mathsf{O}(C_{o_n}) \wedge \mathsf{H}(\mathsf{Y}(R_{o_i}) \Rightarrow \neg(R_{o_j})) \wedge \neg R_{o_i} \wedge \ldots$	PPLTL
	$\mathsf{F}(C_{o_1}) \wedge \ldots \wedge \mathsf{F}(C_{o_n}) \wedge \mathsf{G}(R_{o_i} \Rightarrow \mathsf{X}(\neg R_{o_j})) \wedge \ldots$	LTL_f
	$\mathsf{X}(\mathsf{X}(\ldots \mathsf{X}(R_{kitchen})))$	PPLTL
	$\mathsf{O}(R_{kitchen} \wedge \mathsf{Y}(\mathsf{Y}(\ldots \mathsf{Y}(start))))$	LTL_f

Data availability

All code and benchmarks are publicly available.

References

[1] W. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113.
[2] N. Alechina, B. Logan, M. Dastani, Modeling norm specification and verification in multiagent systems, IfCoLog J. Log. Appl. 5 (2018).
[3] B. Aminof, G. De Giacomo, S. Rubin, Stochastic fairness and language-theoretic fairness in planning in nondeterministic domains, in: Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26–30, 2020, AAAI Press, 2020, pp. 20–28.
[4] A. Artale, L. Geatti, N. Gigante, A. Mazzullo, A. Montanari, A singly exponential transformation of LTL[X, F] into pure past LTL, in: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2–8, 2023, 2023, pp. 65–74.
[5] F. Bacchus, C. Boutilier, A. Grove, Rewarding behaviors, in: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, vol. 2, IAAI 96, Portland, Oregon, USA, August 4–8, 1996, AAAI Press / The MIT Press, 1996, pp. 1160–1167.
[6] F. Bacchus, C. Boutilier, A. Grove, Structured solution methods for non-Markovian decision processes, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27–31, 1997, AAAI Press / The MIT Press, Providence, Rhode Island, USA, 1997, pp. 112–117.
[7] F. Bacchus, F. Kabanza, Planning for temporally extended goals, in: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, vol. 2, IAAI 96, Portland, Oregon, USA, August 4–8, 1996, AAAI Press, 1996, pp. 1215–1222.
[8] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. Artif. Intell. 22 (1998) 5–27.
[9] F. Bacchus, F. Kabanza, Using temporal logics to express search control knowledge for planning, Artif. Intell. 116 (2000) 123–191.
[10] C. Baier, J.P. Katoen, K. Guldstrand Larsen, Principles of Model Checking, MIT Press, 2008.
[11] J.A. Baier, F. Bacchus, S.A. McIlraith, A heuristic search approach to planning with temporally extended preferences, Artif. Intell. 173 (2009) 593–618.
[12] J.A. Baier, C. Fritz, M. Bienvenu, S.A. McIlraith, Beyond classical planning: procedural control knowledge and preferences in state-of-the-art planners, in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13–17, 2008, AAAI Press, 2008, pp. 1509–1512.
[13] J.A. Baier, S.A. McIlraith, Planning with first-order temporally extended goals using heuristic search, in: Proceedings, the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16–20, 2006, AAAI Press, Boston, Massachusetts, USA, 2006, pp. 788–795.
[14] J.A. Baier, S.A. McIlraith, Planning with temporally extended goals using heuristic search, in: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6–10, 2006, AAAI, 2006, pp. 342–345.
[15] S. Bansal, L.E. Kavraki, M.Y. Vardi, A.M. Wells, Synthesis from satisficing and temporal goals, in: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, the Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 9679–9686.

[16] H. Barringer, M. Fisher, D.M. Gabbay, G. Gough, R. Owens, METATEM: a framework for programming in temporal logic, in: Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, the Netherlands, May 29 – June 2, 1989, Springer, 1989, pp. 94–129.
[17] P. Baumgartner, S. Thiébaux, F.W. Trevizan, Heuristic search planning with multi-objective probabilistic LTL constraints, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018, AAAI Press, 2018, pp. 415–424.
[18] J. Benton, A.J. Coles, A. Coles, Temporal planning with preferences and time-dependent continuous costs, in: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25–19, 2012, AAAI, 2012.
[19] M. Bienvenu, C. Fritz, S.A. McIlraith, Specifying and computing preferred plans, Artif. Intell. 175 (2011) 1308–1345.
[20] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A. Gerevini, E. Scala, Planning for temporally extended goals in pure-past linear temporal logic, in: Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8–13, 2023, AAAI Press, 2023, pp. 61–69.
[21] L. Bonassi, G. De Giacomo, M. Favorito, F. Fuggitti, A.E. Gerevini, E. Scala, FOND planning for pure-past linear temporal logic goals, in: ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023), IOS Press, 2023, pp. 279–286.
[22] L. Bonassi, A.E. Gerevini, F. Percassi, E. Scala, On planning with qualitative state-trajectory constraints in PDDL3 by compiling them away, in: Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (Virtual), August 2–13, 2021, AAAI Press, 2021, pp. 46–50.
[23] L. Bonassi, A.E. Gerevini, E. Scala, Planning with qualitative action-trajectory constraints in PDDL, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23–29 July 2022, ijcai.org, 2022, pp. 4606–4613.
[24] L. Bonassi, A.E. Gerevini, E. Scala, Dealing with numeric and metric time constraints in PDDL3 via compilation to numeric planning, in: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20–27, 2024, AAAI Press, Vancouver, Canada, 2024, pp. 20036–20043.
[25] L. Bonassi, G.D. Giacomo, A.E. Gerevini, E. Scala, Shielded FOND: planning with safety constraints in pure-past linear temporal logic, in: ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024), IOS Press, 2024, pp. 1262–1269.
[26] A. Camacho, S.A. McIlraith, Strong fully observable non-deterministic planning with LTL and LTLf goals, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10–16, 2019, ijcai.org, 2019, pp. 5523–5531.
[27] A. Camacho, E. Triantafillou, C.J. Muise, J.A. Baier, S.A. McIlraith, Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4–9, 2017, AAAI Press, San Francisco, California, USA, 2017, pp. 3716–3724.
[28] L. Ceriani, A.E. Gerevini, Planning with always preferences by compilation into STRIPS with action costs, in: Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel, AAAI Press, 2015, pp. 161–165.
[29] A. Chandra, D. Kozen, L. Stockmeyer, Alternation, J. ACM 28 (1981).
[30] A. Cimatti, L. Geatti, N. Gigante, A. Montanari, S. Tonetta, Reactive synthesis from extended bounded response LTL specifications, in: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21–24, 2020, IEEE, 2020, pp. 83–92.
[31] A. Cimatti, F. Giunchiglia, E. Giunchiglia, P. Traverso, Planning via model checking: a decision procedure for AR, in: Recent Advances in AI Planning, 4th European Conference on Planning, Proceedings, ECP'97, Toulouse, France, September 24–26, 1997, Springer, 1997, pp. 130–142.
[32] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artif. Intell. 147 (2003) 35–84.
[33] G. De Giacomo, A. Di Stasio, F. Fuggitti, S. Rubin, Pure-past linear temporal and dynamic logic on finite traces, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 4959–4965.
[34] G. De Giacomo, M. Favorito, J. Li, M.Y. Vardi, S. Xiao, S. Zhu, LTLf synthesis as AND-OR graph search: knowledge compilation at work, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23–29 July 2022, ijcai.org, 2022, pp. 2591–2598.
[35] G. De Giacomo, S. Rubin, Automata-theoretic foundations of FOND planning for LTLf and LDLf goals, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 4729–4735.
[36] G. De Giacomo, M.Y. Vardi, Automata-theoretic approach to planning for temporally extended goals, in: Recent Advances in AI Planning, 5th European Conference on Planning, Proceedings, ECP'99, Durham, UK, September 8–10, 1999, Springer, 1999, pp. 226–238.
[37] G. De Giacomo, M.Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013, IJCAI/AAAI, 2013, pp. 854–860.
[38] S. Edelkamp, On the compilation of plan constraints and preferences, in: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6–10, 2006, AAAI, 2006, pp. 374–377.
[39] E.A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), Elsevier and MIT Press, 1990, pp. 995–1072.
[40] M. Fisher, M. Wooldridge, Temporal reasoning in agent-based systems, in: Handbook of Temporal Reasoning in Artificial Intelligence, in: Foundations of Artificial Intelligence, vol. 1, Elsevier, 2005, pp. 469–495.
[41] R. Fraga Pereira, F. Fuggitti, F. Meneguzzi, G. De Giacomo, Temporally extended goal recognition in fully observable non-deterministic domain models, Appl. Intell. 54 (2024) 470–489.
[42] R. Fraga Pereira, A. Grahl Pereira, F. Messa, G. De Giacomo, Iterative depth-first search for FOND planning, in: Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (Virtual), June 13–24, 2022, AAAI Press, 2022, pp. 90–99.
[43] F. Fuggitti, Efficient Techniques for Automated Planning for Goals in Linear Temporal Logics on Finite Traces, PhD Dissertation, Sapienza University & York University, 2023.
[44] F. Fuggitti, T. Chakraborti, NL2LTL - a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas, in: Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7–14, 2023, AAAI Press, 2023, pp. 16428–16430.
[45] A. Gabaldon, Non-Markovian control in the situation calculus, Artif. Intell. 175 (2011) 25–48.
[46] D.M. Gabbay, A. Pnueli, S. Shelah, J. Stavi, On the temporal analysis of fairness, in: Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980, ACM Press, 1980, pp. 163–173.
[47] L. Geatti, M. Montali, A. Rivkin, Foundations of reactive synthesis for declarative process specifications, in: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20–27, 2024, AAAI Press, Vancouver, Canada, 2024, pp. 17416–17425.
[48] H. Geffner, B. Bonet, A Concise Introduction to Models and Methods for Automated Planning, Morgan & Claypool Publishers, 2013.
[49] A. Gerevini, P. Haslum, D. Long, A. Saetti, Y. Dimopoulos, Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners, Artif. Intell. 173 (2009) 619–668.
[50] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning - Theory and Practice, Elsevier, 2004.
[51] F. Giunchiglia, P. Traverso, Planning as model checking, in: Recent Advances in AI Planning, 5th European Conference on Planning, Proceedings, ECP'99, Durham, UK, September 8–10, 1999, Springer, 1999, pp. 1–20.

[52] P. Haslum, Optimal delete-relaxed (and semi-relaxed) planning with conditional effects, in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013, IJCAI/AAAI, 2013, pp. 2291–2297.
[53] K. He, A.M. Wells, L.E. Kavraki, M.Y. Vardi, Efficient symbolic reactive synthesis for finite-horizon tasks, in: International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20–24, 2019, IEEE, 2019, pp. 8993–8999.
[54] M. Helmert, The fast downward planning system, J. Artif. Intell. Res. 26 (2006) 191–246.
[55] J. Hoffmann, S. Edelkamp, The deterministic part of IPC-4: an overview, J. Artif. Intell. Res. 24 (2005) 519–579.
[56] J. Hoffmann, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (2001) 253–302.
[57] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
[58] C. Hsu, B.W. Wah, R. Huang, Y. Chen, Constraint partitioning for solving planning problems with trajectory constraints and goal preferences, in: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6–12, 2007, 2007, pp. 1924–1929.
[59] M. Knobbout, M. Dastani, J.C. Meyer, A dynamic logic of norm change, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, the Hague, the Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), IOS Press, 2016, pp. 886–894.
[60] O. Lichtenstein, A. Pnueli, L.D. Zuck, The glory of the past, in: Logics of Programs, Conference, Proceedings, Brooklyn College, New York, NY, USA, June 17–19, 1985, Springer, 1985, pp. 196–218.
[61] I. Mallett, S. Thiébaux, F.W. Trevizan, Progression heuristics for planning with probabilistic LTL constraints, in: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, the Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021, AAAI Press, 2021, pp. 11870–11879.
[62] Z. Manna, Verification of Sequential Programs: Temporal Axiomatization, Springer, Netherlands, 1982, pp. 53–102.
[63] C.J. Muise, S.A. McIlraith, J.C. Beck, Improved non-deterministic planning by exploiting state relevance, in: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25–19, 2012, AAAI, 2012, pp. 172–180.
[64] B. Nebel, On the compilability and expressive power of propositional planning formalisms, J. Artif. Intell. Res. 12 (2000) 271–315.
[65] F. Patrizi, N. Lipovetzky, G. De Giacomo, H. Geffner, Computing infinite plans for LTL goals using a classical planner, in: IJCAI 2011, in: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011, Barcelona, Catalonia, Spain, July 16–22, IJCAI/AAAI, 2011, pp. 2003–2008.
[66] F. Patrizi, N. Lipovetzky, H. Geffner, Fair LTL synthesis for non-deterministic systems using strong cyclic planners, in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013, IJCAI/AAAI, 2013, pp. 2343–2349.
[67] F. Percassi, A.E. Gerevini, On compiling away PDDL3 soft trajectory constraints without using automata, in: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11–15, 2019, AAAI Press, 2019, pp. 320–328.
[68] S. Richter, M. Westphal, The LAMA planner: guiding cost-based anytime planning with landmarks, J. Artif. Intell. Res. 39 (2010) 127–177.
[69] J. Rintanen, Complexity of planning with partial observability, in: Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), Whistler, June 3—7 2004, British Columbia, Canada, AAAI, 2004, pp. 345–354.
[70] G. Röger, F. Pommerening, M. Helmert, Optimal planning in the presence of conditional effects: extending LM-cut with context splitting, in: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), IOS Press, 2014, pp. 765–770.
[71] S. Sohrabi, J. Baier, S. McIlraith, Preferred explanations: theory and generation via planning, in: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7–11, 2011, AAAI Press, 2011, pp. 261–267.
[72] S. Thiébaux, C. Gretton, J.K. Slaney, D. Price, F. Kabanza, Decision-theoretic planning with non-Markovian rewards, J. Artif. Intell. Res. 25 (2006) 17–74.
[73] S. Thiébaux, J. Hoffmann, B. Nebel, In defense of PDDL axioms, Artif. Intell. 168 (2005) 38–69.
[74] J. Torres, J.A. Baier, Polynomial-time reformulations of LTL temporally extended goals into final-state goals, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, AAAI Press, 2015, pp. 1696–1703.