Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic

Luigi Bonassi[1], **Giuseppe De Giacomo**[2,3], **Marco Favorito**[4,*], **Francesco Fuggitti**[3,5,†], **Alfonso Emilio Gerevini**[1], **Enrico Scala**[1]

[1] University of Brescia, Italy
[2] University of Oxford, UK
[3] Sapienza University, Italy
[4] Bank of Italy
[5] York University, Canada
luigi.bonassi@unibs.it, degiacomo@diag.uniroma1.it, fuggitti@diag.uniroma1.it, marco.favorito@gmail.com, alfonso.gerevini@unibs.it, enrico.scala@unibs.it

Abstract

We study classical planning for temporally extended goals expressed in Pure-Past Linear Temporal Logic (PPLTL). PPLTL is as expressive as Linear-time Temporal Logic on finite traces (LTL_f), but as shown in this paper, it is computationally much better behaved for planning. Specifically, we show that planning for PPLTL goals can be encoded into classical planning with minimal overhead, introducing only a number of new fluents that is at most linear in the PPLTL goal and no spurious additional actions. Based on these results, we implemented a system called Plan4Past, which can be used along with state-of-the-art classical planners, such as LAMA. An empirical analysis demonstrates the practical effectiveness of Plan4Past, showing that a classical planner generally performs better with our compilation than with other existing compilations for LTL_f goals over the considered benchmarks.

Introduction

In AI Planning, a temporally extended goal is a (possibly complex) property that the state-trace induced by a plan has to satisfy. Planning for temporally extended goals has a long tradition in AI Planning, including pioneering work in the late '90s (Bacchus, Boutilier, and Grove 1996; Bacchus and Kabanza 1996; Bacchus, Boutilier, and Grove 1997; Bacchus and Kabanza 2000), work on planning via Model Checking (Cimatti et al. 1997; De Giacomo and Vardi 1999; Giunchiglia and Traverso 1999), and work on declarative and procedural constraints (Baier and McIlraith 2006; Baier et al. 2008). Also, the standard language PDDL3 (Gerevini et al. 2009) incorporates a class of temporally extended goals called state-trajectory constraints.

Linear-time Temporal Logic (LTL) is a powerful formalism to express temporally extended goals, which has been advocated as an excellent tool to express properties of processes in Formal Methods (Baier, Katoen, and Guldstrand Larsen 2008). Since AI Planning is usually interested in tasks that terminate, a finite-trace variant of LTL, namely LTL_f, has often been employed (Bacchus and Kabanza 1996; Baier and McIlraith 2006; De Giacomo and Vardi 2013). Notably, an alternative to LTL_f is the Pure-Past Linear Temporal Logic, or PPLTL (Lichtenstein, Pnueli, and Zuck 1985), which has been attractive in expressing non-Markovian rewards in MDPs (Bacchus, Boutilier, and Grove 1996), normative properties in multi-agent systems (Fisher and Wooldridge 2005; Knobbout, Dastani, and Meyer 2016; Alechina, Logan, and Dastani 2018), explanations in dynamical systems (Sohrabi, Baier, and McIlraith 2011), or synthesis specifications (Cimatti et al. 2020). PPLTL looks at the trace backward instead of forward as LTL_f and does so by expressing properties on traces using past operators only. PPLTL and LTL_f have the same expressive power, but translating a formula from one into the other (and vice versa) can be prohibitive since the best-known algorithms are 3EXPTIME. See (De Giacomo et al. 2020) for a survey on PPLTL.

Planning for LTL_f goals in deterministic domains requires some properties to be achieved along the execution of a plan and has already been studied in, e.g., (Baier and McIlraith 2006; De Giacomo and Vardi 2013; Torres and Baier 2015). Similarly, planning for PPLTL goals requires reaching, from a specified initial state, a certain state satisfying the PPLTL goal, i.e., the state-trace produced to reach such a state satisfies the goal formula.

From the literature, it is well-known that LTL and variants have a convenient fixpoint characterization that allows for splitting any formula into a propositional formula to be checked at the current instant and a temporal formula to be checked at the next instant (Gabbay et al. 1980; Manna 1982; Emerson 1990). This property has already been exploited in AI, e.g., in the MetateM approach (Barringer et al. 1989), and later under the name of "formula progression" in (Bacchus and Kabanza 1996), which is perhaps the most influential work on planning for temporally extended goals.

Analogously, when we consider PPLTL formulas, such a fixpoint characterization splits the formula into a propositional formula on the current instant and a temporal formula on the past to be checked at the *previous* instant. However, while the future has not happened yet and needs to be guessed, the past has already happened and needs only to be

[*] Views and opinions expressed are of the author's own and are not representative of the Bank of Italy's official position.

[†] Corresponding Author.

read. This implies that PPLTL formulas can be easily evaluated by recursively applying their fixpoint characterization. Moreover, the evaluation of PPLTL formulas can be done by storing previous values of a small number of formulas (at most linear in the original formula), à la dynamic programming. A similar line of reasoning was exploited to conveniently handle non-Markovian rewards expressed using PPLTL in (Bacchus, Boutilier, and Grove 1997), but never fully formalized. In this paper, we formalize it rigorously, proving its correctness, and exploit it to show that planning for PPLTL goals can be encoded into classical planning with minimal overhead by only introducing few new fluents, at most linear in the size of the PPLTL goal, and without adding any spurious actions. These new fluents keep track of the satisfaction of *few* key subformulas of the temporal goal at planning time, reducing planning for temporally extended goals to classical planning. The use of PPLTL is crucial to obtain such nice results since it avoids any form of guessing about the future. Conversely, if we specify goals in LTL_f, the encoding into classical planning, while clearly possible, results either in worst-case exponential encodings (Baier and McIlraith 2006) or in encodings that include additional spurious actions significantly increasing the plan length (Torres and Baier 2015).

We have implemented our approach in a system called Plan4Past, which can be used along with state-of-the-art classical planners. The experimental analysis using the state-of-the-art planner LAMA (Richter and Westphal 2010) shows the practical effectiveness of Plan4Past by comparing it against existing techniques for LTL_f goals.

Background

Classical Planning

Following (Geffner and Bonet 2013), a planning domain model is a tuple $\mathcal{D} = \langle 2^{\mathcal{F}}, A, \alpha, tr \rangle$, where $2^{\mathcal{F}}$ is the set of possible states and $\mathcal{F}$ is a set of fluents (atomic propositions); A is the set of actions; $\alpha(s) \subseteq A$ is the set of applicable actions in state s; and $tr(s, a)$ is the *deterministic* transition function determining the successor state s' that follows action a in state s. Such a (deterministic) domain model $\mathcal{D}$ is assumed to be compactly represented (e.g., in PDDL (McDermott et al. 1998)), i.e., its size is $|\mathcal{F}|$, and a state is an assignment for all fluents in $\mathcal{F}$. Given the set of literals of $\mathcal{F}$ as $Lits(\mathcal{F}) := \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$, every action $a \in A$ is a pair $\langle Pre_a, \mathit{Eff}_a \rangle$, where $Pre_a \subseteq Lits(\mathcal{F})$ represents action preconditions and Eff_a represents action effects given in the form of conditional effects. A conditional effect is of the form $c \triangleright e$ with $c, e \subseteq Lits(\mathcal{F})$ and c possibly empty. An action a can be applied in a state s if the set of literals in Pre_a holds true in s. A conditional effect $c \triangleright e$ is triggered in a state s if c is true in s. Applying a in s results in a successor state s' determined by Eff_a in such a way that $\forall f \in \mathcal{F}$, f holds true in s' if and only if either (*i*) f was true in s and no conditional effect $c \triangleright e$ triggered in s deletes it ($\neg f \in e$) or (*ii*) there is a conditional effect $c \triangleright e$ triggered in s that adds it ($f \in e$). In case of conflicting effects, similarly to other works (Röger, Pommerening, and Helmert 2014), we assume a delete-before-adding semantics. A classical planning problem combines a domain with an initial state and a goal, and consists in looking for a sequence of actions that transforms the initial state into a desired goal state. Formally, a planning problem is a tuple $\Gamma = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a domain model, s_0 is the initial state, i.e., an initial assignment to fluents in $\mathcal{F}$, and G is a set of literals over $\mathcal{F}$ called the *reachability* goal. 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* $s_0, \ldots, s_n$, where $s_{i+1} = tr(s_i, a_i)$ and $a_i \in \alpha(s_i)$ for $i = 0, \ldots, n-1$, and $G \subseteq s_n$.

Classical Planning with Pure-Past Linear Temporal Logic Goals

Pure-Past Linear Temporal Logic (PPLTL) is the variant of LTL_f that talks about the past instead of the future. PPLTL has been recently surveyed in (De Giacomo et al. 2020), where it is denoted as PLTL_f. 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. 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 $\mathit{true} \equiv p \vee \neg p$, $\mathit{false} \equiv \neg \mathit{true}$. Also, $\mathsf{start} \equiv \neg\mathsf{Y}(\mathit{true})$ expresses that the trace has started.

PPLTL formulas are interpreted on *finite nonempty* traces, also called *histories*, $\tau = s_0 \cdots s_n$ 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 $\mathsf{last}(\tau) = s_n$.

Given a trace $\tau = s_0 \cdots s_n$, we denote by $\tau_{i,j}$, with $0 \leq i \leq j \leq n$, the sub-trace $s_i \ldots s_j$ obtained from τ starting from position i and ending in position j. We define the satisfaction relation $\tau, i \models \varphi$, stating that φ holds at instant i, as follows:

- $\tau, i \models p$ iff $\mathsf{length}(\tau) \geq 1$ and $p \in s_i$ (for $p \in \mathcal{P}$);
- $\tau, i \models \neg\varphi$ iff $\tau, i \not\models \varphi$;
- $\tau, i \models \varphi_1 \wedge \varphi_2$ iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \mathsf{Y}\varphi$ iff $i \geq 1$ and $\tau, i-1 \models \varphi$;
- $\tau, i \models \varphi_1 \,\mathsf{S}\, \varphi_2$ iff there exists k, with $0 \leq k \leq i < \mathsf{length}(\tau)$ such that $\tau, k \models \varphi_2$ and for all j, with $k < j \leq i$, we have that $\tau, j \models \varphi_1$.

A PPLTL formula φ is *true* in τ, denoted $\tau \models \varphi$, if $\tau, \mathsf{length}(\tau){-}1 \models \varphi$. We denote by $\mathsf{sub}(\varphi)$ the set of all subformulas of φ obtained from the abstract syntax tree of φ (De Giacomo and Vardi 2013). For instance, if $\varphi = a \wedge \neg\mathsf{Y}(b \vee (c \vee d))$, where a, b, c, d are atomic, then $\mathsf{sub}(\varphi) = \{a, b, c, d, (c \vee d), b \vee (c \vee d), \mathsf{Y}(b \vee (c \vee d)), \neg\mathsf{Y}(b \vee (c \vee d)), a \wedge \neg\mathsf{Y}(b \vee (c \vee d))\}$. Thus, $|\mathsf{sub}(\varphi)|$ defines the size of a PPLTL formula φ.

PPLTL can be used to specify goals on planning problems.

PDDL3 Operator	PPLTL Formula	LTL$_f$ Formula
at-end θ	θ	$\mathsf{F}(\theta \wedge \mathsf{end})$
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	$\mathsf{H}(\theta \rightarrow (\theta \mathsf{S} (\mathsf{H}(\neg\theta) \vee \mathsf{start})))$	$\mathsf{G}(\theta \rightarrow (\theta \mathsf{U} (\mathsf{G}(\neg\theta) \vee \mathsf{end})))$
hold-during $n_1\ n_2\ \theta$	$\bigvee_{0\leq i\leq n_1}(\theta \wedge \mathsf{Y}^i(\mathsf{start})) \vee$ $\bigwedge_{n_1<i\leq n_2}\mathsf{H}(\theta \vee \mathsf{WY}^i(\mathsf{Y}\mathit{true}))$	$\bigvee_{0\leq i\leq n_1}\mathsf{X}^i(\theta \wedge \mathsf{end}) \vee$ $\bigwedge_{n_1<i\leq n_2}\mathsf{WX}^i(\theta)$
$*$ hold-after $n\ \theta$	$\bigvee_{0\leq i\leq n}(\theta \wedge \mathsf{Y}^i(\mathsf{start})) \vee$ $\mathsf{O}(\theta \wedge \mathsf{Y}^{n+1}(\mathsf{O}(\mathsf{start})))$	$\bigvee_{0\leq i\leq n}\mathsf{X}^i(\theta \wedge \mathsf{end}) \vee$ $\mathsf{X}^{n+1}(\mathsf{F}(\theta))$

Table 1: PDDL3 operators, 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$_f$ (De Giacomo and Vardi 2013). $*$ tags an operator with the corrected LTL$_f$ translation.

Definition 1. *A planning problem with* PPLTL *goals is a tuple* $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$*, where* $\mathcal{D} = \langle 2^{\mathcal{F}}, A, \alpha, tr \rangle$ *is a domain model,* s_0 *is the initial state, i.e., an initial assignment to all fluents in* $\mathcal{F}$*, and* φ *is a* PPLTL *formula over* $\mathcal{F}$*.*

A sequence of actions from A is a solution for the planning problem $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$ if the induced sequence of states $\tau = s_0, \ldots, s_n$ satisfies φ. Here, temporally extended goals specify a requirement over the entire induced trajectory of states. 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[1], compute their product, and check non-emptiness on the resulting automaton returning a plan if any (De Giacomo and Vardi 2013; De Giacomo and Rubin 2018). Hence, classical planning for PPLTL goals is PSPACE-complete in the domain and in the goal formula (De Giacomo et al. 2020).

Several properties are natural to express using PPLTL. For instance, some common PPLTL patterns have been employed in the context of MDP rewards in (Bacchus, Boutilier, and Grove 1996) or as norms in multi-agent systems (Fisher and Wooldridge 2005; Knobbout, Dastani, and Meyer 2016; Alechina, Logan, and Dastani 2018). Further examples appear in (De Giacomo et al. 2020). Here, we provide the translation to equivalent PPLTL formulas of PDDL3 patterns (Table 1) (Gerevini et al. 2009) and DECLARE templates (Table 2), the *de-facto* standard modeling language for Business Processes (van der Aalst, Pesic, and Schonenberg 2009). Notably, a systematic translation between LTL$_f$ and PPLTL (and vice versa) does exist, but it is impractical (De Giacomo et al. 2020) as it is not simply based on an induction on the structure of the formula. We formally check the correctness of translations for both tables and formulas in the experiments by checking equivalence in terms of languages, i.e., by translating them to DFA and solving graph isomorphism.

[1]In fact, for PPLTL, the automaton is directly a DFA (De Giacomo et al. 2020).

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)	$\mathsf{H}(b \rightarrow \neg\mathsf{Y}(a))$	$\mathsf{G}(a \rightarrow \neg\mathsf{X}(b))$

Table 2: DECLARE templates, their equivalent PPLTL and LTL$_f$ formulas. a, b are atomic propositions.

Handling PPLTL Goals

In this section, we develop the bases for our technique. First, we observe that any sequence of actions produces a trace on which PPLTL formulas can be evaluated. Therefore, while the planning process goes on, sequences of actions are produced, traces are generated, and over them PPLTL goals can be evaluated. The difficulty is that evaluating PPLTL formulas requires a trace, and searching through traces is quite demanding. Instead, our technique does not consider traces at all. In particular, it exploits the following observations: (*i*) to evaluate the PPLTL goal formula we only need the truth value of its subformulas; (*ii*) every PPLTL formula can be put in a form where its evaluation depends only on the current propositional evaluation and the evaluation of a key set of PPLTL subformulas at the previous instant; (*iii*) one can recursively compute and keep the value of such a small set of formulas as additional propositional variables in the state of the planning domain. Next, we detail these observations.

Temporal operators in LTL and LTL$_f$ can be decomposed into present and future components, giving a fixpoint characterization of the *until* operator (U) – we remind the reader that other temporal operators as *eventually* (F) and *always* (G) are abbreviations of formulas involving U:

$$\phi_1 \,\mathsf{U}\, \phi_2 \;\equiv\; \phi_2 \vee (\phi_1 \wedge \mathsf{X}(\phi_1 \,\mathsf{U}\, \phi_2)).$$

Analogously, PPLTL formulas can be decomposed into present and past components, given the fixpoint characterization of the since operator:

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

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

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

- $\mathsf{pnf}(\phi_1 \wedge \phi_2) = \mathsf{pnf}(\phi_1) \wedge \mathsf{pnf}(\phi_2)$;
- $\mathsf{pnf}(\neg\phi) = \neg\mathsf{pnf}(\phi)$.

For convenience, we add $\mathsf{pnf}(\mathsf{O}\phi) = \mathsf{pnf}(\phi) \vee \mathsf{Y}(\mathsf{O}\phi)$.

A formula resulting from the application of $\mathsf{pnf}(\cdot)$ 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. 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 the size of the formula (i.e., $|\mathsf{sub}(\varphi)|$). Moreover, $\mathsf{pnf}(\varphi)$ is equivalent to φ.*

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

Now, we show that to evaluate a PPLTL formula φ, we only need to keep track of the truth values of some key subformulas of φ. To do so, we introduce Σ_φ as the set of *propositions* of the form "$\mathsf{Y}\phi$" containing:

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

To keep track of the truth of each proposition in Σ_φ, we define a specific interpretation σ:

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

Intuitively, given an instant i, σ_i tells us which propositions in Σ_φ are true at instant i. By suitably maintaining the value of propositions in Σ_φ in σ_i, we can evaluate a PPLTL formula just by using the propositional interpretation in the current instant i and the truth value assigned by σ_i to propositions related to the previous instant.

Definition 2. *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:*

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

Intuitively, the $\mathsf{val}(\phi, \sigma_i, s_i)$ predicate allows us to determine what is the truth value of any PPLTL formula $\phi \in \mathsf{sub}(\varphi)$ by reading a propositional interpretation s_i from trace τ and keeping track of the truth value of propositions in Σ_φ by means of σ_i. Observe that rules in Definition 2 basically follow the PNF transformation rules where subformulas within the Y-scope are interpreted as propositions.

Now, given a trace $\tau = s_0 \cdots s_n$ over $\mathcal{P}$, we compute a corresponding trace $\tau^{[\varphi]} = \sigma_0 \cdots \sigma_n$ over Σ_φ, where:

- $\sigma_0($"$\mathsf{Y}\phi$"$) \doteq \bot$ for each "$\mathsf{Y}\phi$" $\in \Sigma_\varphi$;
- $\sigma_i($"$\mathsf{Y}\phi$"$) \doteq \mathsf{val}(\phi, \sigma_{i-1}, s_{i-1})$, for all i with $0 < i \leq n$.

First, the following result for traces of length 1 holds.

Lemma 1. *Let φ be* PPLTL *formula over $\mathcal{P}$, $\phi \in \mathsf{sub}(\varphi)$ a subformula of φ, and $\tau = s_0$ a trace over $\mathcal{P}$ of length 1. Then, $s_0 \models \phi$ iff $\mathsf{val}(\phi, \sigma_0, s_0)$.*

Proof. By structural induction on the formula ϕ.

- $\phi = p$. By definition of $\mathsf{val}(\cdot)$, $\mathsf{val}(p, \sigma_0, s_0)$ iff $s_0 \models p$.
- $\phi = \mathsf{Y}\phi'$. By definition of σ_0, $\sigma_0($"$\mathsf{Y}\phi'$"$) = \bot$, and by the semantics, $s_0 \not\models \mathsf{Y}\phi'$. Therefore, the thesis holds.
- $\phi = \phi_1 \,\mathsf{S}\, \phi_2$. $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \sigma_i, s_i)$ iff $\mathsf{val}(\phi_2, \sigma_i, s_i) \vee (\mathsf{val}(\phi_1, \sigma_i, s_i) \wedge \sigma_i \models$ "$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$"). By definition of σ_0, $\sigma_0($"$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$"$) = \bot$, hence the formula above simplifies to $\mathsf{val}(\phi_2, \sigma_i, s_i)$. On the other hand, by the semantics, $s_0 \models \phi_1 \,\mathsf{S}\, \phi_2$ iff $s_0 \models \phi_2$. Hence, by induction the thesis holds.
- $\phi = \phi_1 \wedge \phi_2$ or $\phi = \neg\phi'$. The thesis holds by structural induction.

□

Next, we extend the previous result to traces of any length.

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,*

$$\tau \models \phi \text{ iff } \mathsf{val}(\phi, \mathsf{last}(\tau^{[\varphi]}), \mathsf{last}(\tau)).$$

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: $\tau = s_0$. By Lemma 1, the thesis holds.
- Inductive step: Let $\tau = \tau_{n-1} \cdot s_n$. By inductive hypothesis, the thesis holds for the trace τ_{n-1} of length $n-1$:

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

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

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

  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 $\tau_{n-1} \cdot s_n \models p$ iff $s_n \models p$. For the $\mathsf{val}(\cdot)$ predicate we have that $s_n \models p$ iff $\mathsf{val}(p, \sigma_n, s_n)$. Therefore, the thesis holds.
  - $\phi = \mathsf{Y}\phi'$. We have that $\tau_{n-1} \cdot s_n \models \mathsf{Y}\phi'$ iff $\tau_{n-1} \models \phi'$. By inductive hypothesis, $\tau_{n-1} \models \phi'$ 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 $\sigma_n \models$ "$\mathsf{Y}\phi'$", which corresponds to $\mathsf{val}(\phi', \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$. Hence, the thesis holds.
  - $\phi = \phi_1 \,\mathsf{S}\, \phi_2$. In this case it suffices to remember that $\tau_{n-1} \cdot s_n \models \phi_1 \,\mathsf{S}\, \phi_2$ iff $\tau_{n-1} \cdot s_n \models \phi_2 \vee (\phi_1 \wedge \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2))$. On the other hand, $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \sigma_n, s_n)$ iff $\mathsf{val}(\phi_2, \sigma_n, s_n) \vee (\mathsf{val}(\phi_1, \sigma_n, s_n) \wedge \sigma_n \models$ "$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$"). By structural induction we have that $\tau_{n-1} \cdot s_n \models \phi_1$ iff $\mathsf{val}(\phi_1, \sigma_n, s_n)$, and $\tau_{n-1} \cdot s_n \models \phi_2$ iff $\mathsf{val}(\phi_2, \sigma_n, s_n)$. Moreover, $\tau_{n-1} \cdot s_n \models \mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$ iff $\tau_{n-1} \models \phi_1 \,\mathsf{S}\, \phi_2$, and $\sigma_n \models$

"$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$" iff $\mathsf{val}(\phi_1 \,\mathsf{S}\, \phi_2, \mathsf{last}(\tau_{n-1}^{[\varphi]}), \mathsf{last}(\tau_{n-1}))$. Finally, we have that $\tau_{n-1} \models \phi_1 \,\mathsf{S}\, \phi_2$ 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 bases of our technique as it guarantees that by keeping a suitably updated trace σ, we can evaluate our PPLTL goal only using the propositional interpretation in the current instant and the truth value of the "$\mathsf{Y}\phi$" formulas in σ, without considering the entire trace.

Encoding PPLTL Goals in Planning

We devise a new encoding for planning for PPLTL goals exploiting Theorem 1. The key idea behind our approach is that, given a PPLTL formula and a planning domain we keep track of values of formulas in σ as actions are applied.

Similarly to other encoding-based approaches dealing with temporally extended goals, e.g., (Baier and McIlraith 2006; Torres and Baier 2015), we address planning for temporally extended goals in three steps. First, compile the original planning problem Γ with the temporally extended goal into a planning problem Γ' with a reachability goal. Second, invoke any off-the-shelf sound and complete planner to compute a plan solving the compiled problem Γ'. Third, rework the computed plan to get the solution for the original problem Γ. In our approach, we exploit Theorem 1 to do the encoding in the first step, and since no extra control actions are introduced step three trivializes.

Given a planning problem $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$, where $\mathcal{D} = \langle 2^{\mathcal{F}}, A, \alpha, tr \rangle$ is a planning domain, s_0 the initial state and φ a PPLTL goal, the compiled planning problem is $\Gamma' = \langle \mathcal{D}', s_0', G' \rangle$, where $\mathcal{D}' = \langle 2^{\mathcal{F}'}, A, \alpha', tr' \rangle$ is the compiled planning domain, s_0' the new initial state and G' is the new reachability goal. Next, we describe the encoding from Γ to Γ' using their compact representations. In particular, α' and tr' will be induced by the introduction of axioms and a reformulation of the actions' preconditions and effects.

Fluents. $\mathcal{F}'$ contains the fluents of $\mathcal{F}$, as well as one fluent for each proposition "$\mathsf{Y}\phi$" in Σ_φ to keep track of propositional interpretations σ_i, and the set of predicates P_{der} defined below. Formally, $\mathcal{F}' = \mathcal{F} \cup \Sigma_\varphi \cup P_{der}$.

Axioms. We employ *axioms* (Thiébaux, Hoffmann, and Nebel 2005), which have the form $d \leftarrow \psi$, where $d \in P_{der}$ is a positive literal called *derived predicate*, and ψ is a propositional formula over a set of predicates. Let s be a state, axiom $d \leftarrow \psi$ determines that the derived predicate d holds true in s if and only if $s \models \psi$.

Here, we include an axiom $\mathsf{val}_\phi \leftarrow \psi$ for every subformula $\phi \in \mathsf{sub}(\varphi)$, and P_{der} is defined as the set of all derived predicates val_ϕ, i.e., $P_{der} = \{\mathsf{val}_\phi \mid \phi \in \mathsf{sub}(\varphi)\}$. These axioms are intended to be such that the current state $(\sigma_i, s_i) \models \mathsf{val}_\phi$ iff $\mathsf{val}(\phi, \sigma_i, s_i)$. By mimicking rules in Definition 2, we get the following axioms:

- $\mathsf{val}_p \leftarrow p$;
- $\mathsf{val}_{\mathsf{Y}\phi} \leftarrow$ "$\mathsf{Y}\phi$";
- $\mathsf{val}_{\phi_1 \,\mathsf{S}\, \phi_2} \leftarrow (\mathsf{val}_{\phi_2} \vee (\mathsf{val}_{\phi_1} \wedge$ "$\mathsf{Y}(\phi_1 \,\mathsf{S}\, \phi_2)$"$))$;
- $\mathsf{val}_{\phi_1 \wedge \phi_2} \leftarrow (\mathsf{val}_{\phi_1} \wedge \mathsf{val}_{\phi_2})$;
- $\mathsf{val}_{\neg\phi} \leftarrow \neg\mathsf{val}_\phi$.

Clearly, we have that $(\sigma_i, s_i) \models \mathsf{val}_\phi$ iff $\mathsf{val}(\phi, \sigma_i, s_i)$. Hence, to define Γ', we build a set of axioms for every subformula ϕ in $\mathsf{sub}(\varphi)$, i.e., $\{\mathsf{val}_\phi \leftarrow \phi \mid \phi \in \mathsf{sub}(\varphi)\}$. Axioms allow us to elegantly model the mathematics of the previous section (i.e., the $\mathsf{val}(\phi, \sigma_i, s_i)$) and are often convenient when dealing with more sophisticated forms of planning (e.g., (Borgwardt et al. 2022)). They also simplify the action schema and goal descriptions without adding control predicates among fluents, thus simplifying the search, as shown in (Thiébaux, Hoffmann, and Nebel 2005).

Initial State. The initial state is the same as the original problem Γ for the original fluents in $\mathcal{F}$, whereas the new fluents "$\mathsf{Y}\phi$" $\in \Sigma_\varphi$ are assigned to the truth value given by σ_0. That is $s_0' = (\sigma_0, s_0)$.

Domain Actions. Every domain's action in A is modified on its effects by adding a way to update the assignments of propositions in Σ_φ. For each "$\mathsf{Y}\phi$" $\in \Sigma_\varphi$, we model assignments updates by a set of conditional effects of the form:

$$\mathsf{val}_\phi \rhd \quad \text{"}\mathsf{Y}\phi\text{"}$$
$$\neg\mathsf{val}_\phi \rhd \neg\text{"}\mathsf{Y}\phi\text{"}$$

These effects are exactly the same for every action $a \in A$. Also, since σ_i maintains values of "$\mathsf{Y}\phi$" in Σ_φ they are independent of 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 either which action has been executed or which effect such action has had on the original fluents.

Formally, let $\mathsf{Val}_\varphi = \{\mathsf{val}_\phi \rhd$ "$\mathsf{Y}\phi$", $\neg\mathsf{val}_\phi \rhd \neg$"$\mathsf{Y}\phi$" $\mid$ "$\mathsf{Y}\phi$" $\in \Sigma_\varphi\}$. The set of actions A in Γ' remains the same, as in the original problem Γ. For all $a \in A$, we have that the preconditions in Γ' are $Pre_a' = Pre_a$ and the effects in Γ' are $\mathit{Eff}_a' = \mathit{Eff}_a \cup \mathsf{Val}_\varphi$. Observe that, the auxiliary part Val_φ in Eff_a' *deterministically* updates subformulas values in Σ_φ, without affecting any fluent $f \in \mathcal{F}$ of the original domain model. This is crucial for the encoding's correctness.

Goal. The goal in Γ' is encoded as $G' = \{\mathsf{val}_\varphi\}$. That is, $\mathsf{val}(\varphi, \sigma_n, s_n)$, associated to the original PPLTL goal formula φ, has to hold true in the last instant, as per Theorem 1.

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

Theorem 2. *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 the size of the temporally extended* PPLTL *goal φ of Γ.*

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 $\tau' \mid_{\mathcal{F}} = \tau = s_0, \ldots, s_n$ 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, actions preconditions do not change, and additional conditional effects on the original actions are deterministic.

Theorem 3 (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 $\pi = a_0, \ldots, a_{n-1}$ is a plan for Γ iff $\pi = a_0, \ldots, a_{n-1}$ is a plan for Γ'.*

Proof. 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, 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 $\tau \models \varphi$. By Theorem 1, we have that $\tau \models \varphi$ 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 $\mathsf{last}(\tau') \models \mathsf{val}_\varphi$. Hence, the thesis holds. □

A direct consequence of Theorem 3 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.

Experiments

We implemented the approach of the previous section in a tool called Plan4Past[2] (P4P). P4P takes as input a PDDL description and a PPLTL formula, and gives as output a new PDDL description, which can be processed by any classical planner supporting axioms and conditional effects. We also tested an alternative version of P4P where all axioms are compiled into conditional effects, two for each sub-formula ϕ encoding the truth value of ϕ after each action. However, the resulting compilation proved much less effective than that using axioms, so we do not consider it in our analysis.

Our analysis sheds some light on the effectiveness of temporally extended goals formulated in PPLTL and handled by P4P, and *semantically* equivalent temporally extended goals formulated in LTL_f and handled by either Exp (Baier and McIlraith 2006) or Poly (Torres and Baier 2015).

To our knowledge, Exp and Poly are the best approaches for planning for LTL_f goals. In particular, Baier and McIlraith (2006) build an NFA for the LTL_f formula and compute the Cartesian product with the planning domain (cf. (De Giacomo and Rubin 2018)), incurring in a worst-case exponential increase in the number of states of the NFA. Similarly, Torres and Baier (2015) implicitly construct the NFA for the goal formula. While the approach in (Torres and Baier 2015) is optimal with respect to the computational complexity, it significantly increases the plan length. Indeed, working with NFAs during planning requires choosing not only the next actions in the plan but also the right nondeterministic transition of the NFA. This translates into extra dummy (synchronization) actions to insert into the plan, making the overall search harder for the planner. Dealing with spurious actions has already been studied in (Nebel 2000) theoretically and practically from a heuristic perspective in, e.g., (Haslum 2013). Therefore 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.

Next, we want to determine whether this theoretical advantage manifests itself in actual planning performance from a practical perspective. To this end, we tested the three considered systems 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 (Richter and Westphal 2010), a planner built on top of FastDownward (Helmert 2006), and $\text{FF}_\mathcal{X}$ (Thiébaux, Hoffmann, and Nebel 2005). 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} (Hoffmann and Nebel 2001) and the landmarks counting heuristics. LAMA yields solution plans of decreasing plan cost incrementally; for our analysis, we take the first generated plan. $\text{FF}_\mathcal{X}$ is yet another satisficing planner based on heuristic search and enforced hill climbing, and is the one originally used with Poly and Exp. 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. All experiments were run on an Intel Xeon Gold 6140M 2.3 GHz, with runtime and memory limits of 1800s and 8GB, respectively.

Benchmark Domains

Our benchmark suite includes BLOCKSWORLD, ELEVATOR, ROVERS, and OPENSTACKS domains. These domains were introduced in past International Planning Competitions, and all but ELEVATOR have also been used by Tor-

[2]Source code, benchmarks and supplementary material are publicly available at https://github.com/whitemech/Plan4Past.

res and Baier (2015). For BLOCKSWORLD, ROVERS, and OPENSTACKS we have a set of instances with the same temporally extended goals defined by Torres and Baier (2015) (hereinafter referred to as TB) and a second set of instances with temporally extended goals defined by us (hereinafter referred to as BF). For ELEVATOR, we only have BF. TB were originally specified in LTL$_f$, and for P4P we manually translated them to PPLTL. We did so for all but one type of temporally extended goal used in (Torres and Baier 2015), namely that 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. For each LTL$_f$ formula that we translated in PPLTL, we formally and automatically proved their *semantic* equivalence by verifying that the two formulations yield the same minimal DFA. BF were designed in PPLTL directly, and analogously to what was done for TB, we formulated an equivalent formulation in LTL$_f$. TB are based on predefined families of formulas that are independent from the domain. Instead, BF are specific for each domain, and were designed to stress all compilations and understand the planner's scalability over non-trivial and large instances. Indeed, all instances with TB proved trivial for Plan4Past. For TB, we have 15 instances for BLOCKSWORLD, 7 for ROVERS, and 10 for OPENSTACKS. Their definition is provided by Torres and Baier (2015). BF are instead described below.

BLOCKSWORLD. BF were formulated to study the reach of all compilations with complex temporally extended goals. Here, BF specify two intertwined goals, both requiring the existence in the state trajectory of the plan of a particular sequence of states. Consider a problem with n blocks, and let $on_{i,j}$ be the predicate modeling block i being on block j. The first goal in PPLTL is $\mathsf{O}(on_{1,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,3} \wedge \mathsf{Y}(\mathsf{O}(... \wedge \mathsf{Y}(\mathsf{O}(on_{n-1,n})))))))$. Its translation in LTL$_f$ is $\mathsf{F}(on_{n-1,n} \wedge \mathsf{X}(\mathsf{F}(on_{n-2,n-1} \wedge \mathsf{X}(\mathsf{F}(... \wedge \mathsf{X}(\mathsf{F}(on_{1,2})))))))$. The second goal (for an even number of blocks) in PPLTL is $\bigwedge_{j\in\{6,8,\ldots,n\}} \mathsf{O}(on_{j,j-1} \wedge \mathsf{Y}(\phi))$ and ϕ is formula $\mathsf{O}(on_{4,3} \wedge \mathsf{Y}(\mathsf{O}(on_{3,2} \wedge \mathsf{Y}(\mathsf{O}(on_{2,1})))))$ encoding the construction of a bigger stack. The same constraint formulated in LTL$_f$ is $\mathsf{F}(on_{2,1} \wedge \mathsf{X}(\mathsf{F}(on_{3,2} \wedge \mathsf{X}(\mathsf{F}(on_{4,3} \wedge \bigwedge_{j\in\{6,8,\ldots,n\}} \mathsf{X}(\mathsf{F}(on_{j,j-1}))))))))$. 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. Here, BF require that a valid plan ships all specified requests following a specific production order. The PPLTL formula $\mathsf{H}(made_{p_3} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_2}))) \wedge \mathsf{H}(made_{p_2} \rightarrow \mathsf{Y}(\mathsf{O}(made_{p_1})))$ encodes that p_1 is made strictly before p_2, that in turn must be made before p_3. The equivalent LTL$_f$ formula is $(made_{p_2}) \,\mathsf{R}\, (\neg made_{p_3}) \wedge (made_{p_1}) \,\mathsf{R}\, (\neg made_{p_2})$. Every order must be shipped, and this is encoded with $\mathsf{O}(shipped_{order})$ in PPLTL, and with $\mathsf{F}(shipped_{order})$ in LTL$_f$.

ROVERS. The goal of this domain is to gather and communicate data about soil, rock and images to the Earth using a set of rovers. BF enforce a total order over the communications of the data. This temporally extended goal implicitly requires the data to be eventually communicated, and is encoded in PPLTL as $\mathsf{O}(data_{soil} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{rock}))) \wedge \mathsf{O}(data_{rock} \wedge \mathsf{WY}(\mathsf{H}(\neg data_{image}))) \wedge \mathsf{O}(data_{image})$, and in LTL$_f$ as $(\neg data_{rock}) \,\mathsf{U}\, (data_{soil}) \wedge (\neg data_{image}) \,\mathsf{U}\, (data_{rock}) \wedge \mathsf{F}(data_{image})$. Also, when the rover reaches the lander, that rover must re-calibrate all cameras (e.g., if the lander is at waypoint w_l and the rover r has 2 cameras, c_1 and c_2, we have, in PPLTL, the formula $((\neg at_{r,w_l} \,\mathsf{S}\, calibrated_{c_1}) \wedge (\neg at_{r,w_l} \,\mathsf{S}\, calibrated_{c_2})) \vee \mathsf{H}(\neg at_{r,w_l})$, and, in LTL$_f$, $\mathsf{G}(at_{r,w_l} \rightarrow (\mathsf{F}(calibrated_{c_1}) \wedge \mathsf{F}(calibrated_{c_2}))))$.

ELEVATOR. This domain models the problem of scheduling passengers in the use of an elevator. In BF, we split the passengers into half VIP and half regular passengers, where VIP passengers must be served before every regular one. For instance, we enforce this in PPLTL with $\mathsf{O}(served_{p_2} \wedge served_{p_3}) \wedge \mathsf{O}(served_{p_0} \wedge served_{p_1} \wedge \mathsf{WY}(\mathsf{H}(\neg served_{p_2} \wedge \neg served_{p_3})))$, and in LTL$_f$ with $\mathsf{F}(served_{p_2} \wedge served_{p_3}) \wedge (\neg served_{p_2} \wedge \neg served_{p_3}) \,\mathsf{U}\, (served_{p_0} \wedge served_{p_1})$. Also, we model that no passenger may share the elevator with another passenger, and do so in PPLTL (resp. LTL$_f$) with $\mathsf{H}(boarded_{p_0} \rightarrow (\neg boarded_{p_1} \wedge \neg boarded_{p_2}))$ (resp. $\mathsf{G}(boarded_{p_0} \rightarrow (\neg boarded_{p_1} \wedge \neg boarded_{p_2}))$).

Experimental Results

Table 3 reports on the overall performance of all compilations across all domains. Coverage-wise, P4P performs equally to or better than both Poly and Exp over most instances. For the TB instances, P4P achieves the same coverage as Poly (the best LTL$_f$-based compilation) but is much faster in terms of average runtime: P4P is roughly one order of magnitude faster than Poly; this seems to be justified by a great reduction in the number of expanded nodes (up to two orders of magnitude in OPENSTACKS). This is somehow expected. Indeed, for each planning action taken, Poly interleaves quite a complex automaton synchronization phase, from the initial state all the way to the goal. On average, in TB instances, 94.3% of actions in plans obtained with Poly come from the automaton's synchronization phases.

The situation is different if we look at the BF instances. Here, the best performing LTL$_f$ compilation is Exp, which is superior to Poly over all instances. BF are of increasing dimensions and have been constructed to be computationally more challenging. For example, in BLOCKSWORLD, the hardest instance in TB requires a 22 actions plan, while BF instances require up to 652 actions. In the case of Poly, the planner has to cope with too many synchronization phases and struggles to find solutions. If we compare P4P and Exp, we observe that P4P is again the system performing generally better. The only exception is for one instance of OPENSTACKS. Exp solves this instance in roughly 739s while P4P times out. By looking at the average number of expanded nodes, LAMA's search turned out to be slightly less informed with P4P in this domain, which leads to timing out in that particular instance. For BLOCKSWORLD, P4P is instead much more effective than Exp, which manages to compile only 7 instances. The compilation time seems to be an issue for both LTL$_f$ compilations.

Indeed, if we look at Figure 1 (right), P4P compiles

Domain		I	Coverage P4P	Poly	Exp	Avg RT P4P	Poly	Exp	Avg EN P4P	Poly	Exp	Avg PL P4P	Poly	Exp
ROVERS	TB	7	**7**	**7**	6	**1.43**	21.11	1.98	**12.67**	616.83	**12.67**	**5.33**	5.67 (74.50)	**5.33**
	BF	40	**33**	6	22	35.36	–	**24.24**	**7665.36**	–	7826.32	43.68	–	**43.50**
BLOCKSWORLD	TB	15	**15**	**15**	8	**1.41**	20.43	13.13	22.12	821.62	**21.75**	7.50	7.88 (132.88)	**7.25**
	BF	21	**21**	1	1	–	–	–	–	–	–	–	–	–
OPENSTACKS	TB	10	**10**	**10**	6	**6.11**	31.66	8.75	**52.67**	3863.33	52.83	22.00	**21.67** (349.00)	22.00
	BF	30	7	5	**8**	**11.88**	68.86	19.50	99.80	1207764.80	**72.40**	**24.00**	**24.00** (841.00)	**24.00**
ELEVATOR	BF	29	**29**	4	**29**	231.83	–	**228.09**	**1712076.48**	–	1712090.79	**75.48**	–	**75.48**
Total			**122**	48	80									

Table 3: Coverage, average Run-Time (Avg RT), Expanded Nodes (Avg EN) and Plan Length (Avg PL) achieved by P4P, Poly and Exp. For Poly, we report in parenthesis the average PL considering the actions added by the compilation. Averages are only among instances solved by those systems that obtain at least half of the coverage of the best performer. Column I is the number of instances in a domain. "–" indicates when a system is excluded by the comparison. Bolds are for best performers.

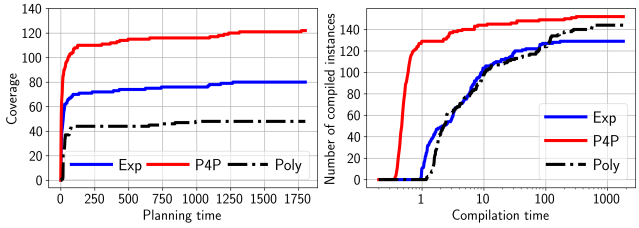
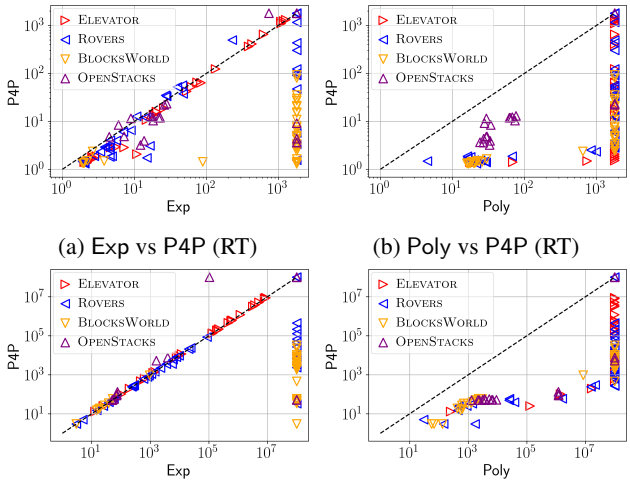


Figure 1: Number of solved instances (left) and compiled instances (right) versus computation time.



(a) Exp vs P4P (RT) (b) Poly vs P4P (RT)

(c) Exp vs P4P (EN) (d) Poly vs P4P (EN)

Figure 2: Pairwise comparison between P4P and Exp (left plots) and between P4P and Poly (right plots) in terms of Run-Time (above) and Expanded Nodes (below).

94.7% of the instances within 10s, while both Poly and Exp converge much more slowly. Figure 1 (left) displays the number of benchmark instances solved with a given timeout. All systems achieve their maximum coverage quite quickly, with P4P leaving the others well behind right after the start.

Figure 2 reports on a pairwise comparison P4P vs Exp and P4P vs Poly over the number of expanded nodes and runtime, instance by instance. P4P is generally faster than Exp, apart from 15 instances. The number of expanded nodes between these two systems is surprisingly similar. Looking at our raw data, we observe that, for most of the instances, Exp spends much more time than P4P in compilation and slightly more time in evaluating a node of the search. The comparison P4P vs Poly confirms our expectation on the number of expanded nodes. P4P expands nodes more slowly than Poly, and therefore the runtime advantage of P4P is related to the fact that P4P leads LAMA to do much less search than Poly.

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

Conclusion

In this paper, we studied classical planning for PPLTL goals. PPLTL is a compelling formalism to express sophisticated planning goals and, compared to LTL$_f$-based approaches, allows for a polynomial-time encoding that is optimal with respect to the computational complexity and does not increase the plan length. Moreover, handling PPLTL goals is remarkably simple and elegant, given the direct mapping between the theoretical formulation and the encoding compilation without sacrificing efficiency. We devised an encoding of planning for PPLTL goals into classical planning for reachability goals and demonstrated its practical effectiveness through extensive experiments. Here, we focused only on PPLTL. However, in principle, our approach can be extended to goals expressed in Pure-Past Linear Dynamic Logic (PPLDL) (De Giacomo et al. 2020), a strictly more expressive variant of PPLTL involving regular expressions. Indeed, also PPLDL has a fixpoint characterization of the temporal operators. This extension remains for future work. Finally, although our focus is on classical planning for PPLTL goals, the theoretical and practical advantages observed in this paper suggest that PPLTL could become a promising candidate for expressing temporally extended properties in other forms of planning, such as nondeterministic planning.

Acknowledgments

This work has been partially supported by the EU H2020 project AIPlan4EU (No. 101016442), the ERC-ADG WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013).

References

Alechina, N.; Logan, B.; and Dastani, M. 2018. Modeling Norm Specification and Verification in Multiagent Systems. *FLAP*, 5(2): 457–490.

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *AAAI*, 1160–1167.

Bacchus, F.; Boutilier, C.; and Grove, A. 1997. Structured solution methods for non-Markovian decision processes. In *AAAI*, 112–117.

Bacchus, F.; and Kabanza, F. 1996. Planning for Temporally Extended Goals. In *AAAI*, 1215–1222. AAAI Press.

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

Baier, C.; Katoen, J.-P.; and Guldstrand Larsen, K. 2008. *Principles of Model Checking*. MIT.

Baier, J. A.; Fritz, C.; Bienvenu, M.; and McIlraith, S. A. 2008. Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners. In *AAAI*, 1509–1512. AAAI.

Baier, J. A.; and McIlraith, S. A. 2006. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 788–795. AAAI.

Barringer, H.; Fisher, M.; Gabbay, D. M.; Gough, G.; and Owens, R. 1989. METATEM: A Framework for Programming in Temporal Logic. In *REX Workshop*, volume 430 of *LNCS*, 94–129. Springer.

Borgwardt, S.; Hoffmann, J.; Kovtunova, A.; Krötzsch, M.; Nebel, B.; and Steinmetz, M. 2022. Expressivity of Planning with Horn Description Logic Ontologies. In *AAAI*, 5503–5511. AAAI Press.

Cimatti, A.; Geatti, L.; Gigante, N.; Montanari, A.; and Tonetta, S. 2020. Reactive Synthesis from Extended Bounded Response LTL Specifications. In *FMCAD*, 83–92. IEEE.

Cimatti, A.; Giunchiglia, F.; Giunchiglia, E.; and Traverso, P. 1997. Planning via Model Checking: A Decision Procedure for *AR*. In *ECP*, 130–142. Springer.

De Giacomo, G.; Di Stasio, A.; Fuggitti, F.; and Rubin, S. 2020. Pure-Past Linear Temporal and Dynamic Logic on Finite Traces. In *IJCAI*, 4959–4965. ijcai.org.

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals. In *IJCAI*, 4729–4735.

De Giacomo, G.; and Vardi, M. Y. 1999. Automata-Theoretic Approach to Planning for Temporally Extended Goals. In *ECP*, 226–238. Springer.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860. IJCAI/AAAI.

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

Fisher, M.; and Wooldridge, M. 2005. Temporal Reasoning in Agent-based Systems. In *FAI*, 469–495. Elsevier.

Gabbay, D. M.; Pnueli, A.; Shelah, S.; and Stavi, J. 1980. On the Temporal Analysis of Fairness. In *POPL*, 163–173. ACM Press.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

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

Giunchiglia, F.; and Traverso, P. 1999. Planning as Model Checking. In *ECP*, 1–20. Springer.

Haslum, P. 2013. Optimal Delete-Relaxed (and Semi-Relaxed) Planning with Conditional Effects. In *IJCAI*, 2291–2297. IJCAI.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26(1): 191–246.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14: 253–302.

Knobbout, M.; Dastani, M.; and Meyer, J. C. 2016. A Dynamic Logic of Norm Change. In *ECAI*, 886–894.

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

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

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – the planning domain definition language. Technical report, ICAPS.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *JAIR*, 12: 271–315.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39: 127–177.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting. In *ECAI*, 765–770.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2011. Preferred explanations: Theory and generation via planning. In *AAAI*, volume 25, 261–267.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *AIJ*, 168(1-2): 38–69.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*, 1696–1703. AAAI Press.

van der Aalst, W.; Pesic, M.; and Schonenberg, H. 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - R&D*, 23(2): 99–113.