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

Luigi Bonassi\textsuperscript{1}, Giuseppe De Giacomo\textsuperscript{2,3}, Marco Favorito\textsuperscript{4,*}, Francesco Fuggitti\textsuperscript{3,5,†}, Alfonso Emilio Gerevini\textsuperscript{1}, Enrico Scala\textsuperscript{1}

\textsuperscript{1} University of Brescia, Italy
\textsuperscript{2} University of Oxford, UK
\textsuperscript{3} Sapienza University, Italy
\textsuperscript{4} Bank of Italy
\textsuperscript{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\textsubscript{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\textsubscript{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 Gulstrand Larsen 2008). Since AI Planning is usually interested in tasks that terminate, a finite-trace variant of LTL, namely LTL\textsubscript{f}, has often been employed (Bacchus and Kabanza 1996; Baier and McIlraith 2006; De Giacomo and Vardi 2013). Notably, an alternative to LTL\textsubscript{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\textsubscript{f} and does so by expressing properties on traces using past operators only. PPLTL and LTL\textsubscript{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\textsubscript{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
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, 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\_\_\_\_\_\_\_ goals.

**Background**

**Classical Planning**

Following (Geffner and Bonet 2013), a planning domain model is a tuple $D = (2^F, A, \alpha, \text{tr})$, where $2^F$ is the set of possible states and $F$ is a set of fluents (atomic propositions); $A$ is the set of actions; $\alpha(s)$ is the set of applicable actions in state $s$; and $\text{tr}(s, a)$ is the deterministic transition function determining the successor state $s'$ that follows action $a$ in state $s$. Such a (deterministic) domain model $D$ is assumed to be compactly represented (e.g., in PDDL (McDermott et al. 1998)), i.e., its size is $|F|$, and a state is an assignment for all fluents in $F$. Given the set of literals of $F$ as $\text{Lits}(F) = F \cup \{\neg f \mid f \in F\}$, every action $a \in A$ is a pair $\langle \text{Pre}_a, \text{Eff}_a \rangle$, where $\text{Pre}_a \subseteq \text{Lits}(F)$ represents action preconditions and $\text{Eff}_a$ represents action effects given in the form of conditional effects. A conditional effect is of the form $c \rightarrow e$ with $c, e \subseteq \text{Lits}(F)$ and $c$ possibly empty. An action $a$ can be applied in a state $s$ if the set of literals in $\text{Pre}_a$ holds true in $s$. A conditional effect $c \rightarrow e$ is triggered in a state $s$ if $c$ is true in $s$. Applying $a$ in $s$ results in a successor state $s'$ determined by $\text{Eff}_a$ in such a way that $\forall f \in F, \neg f$ holds true in $s'$ if and only if either (i) $f$ was true in $s$ and no conditional effect $c \rightarrow e$ triggered in $s$ deletes it ($\neg f \in e$) or (ii) there is a conditional effect $c \rightarrow e$ 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 $\Gamma = (D, s_0, G)$, where $D$ is a domain model, $s_0$ is the initial state, i.e., an initial assignment to fluents in $F$, and $G$ is a set of literals over $F$ called the reachability goal. A solution to planning problem $\Gamma$ 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 $s_{i+1} = \text{tr}(s_i, a_i)$ 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 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\_\_\_. Here, we summarize its main characteristics and give some examples. Given a set $P$ of propositions, PPLTL is defined as:

$$\varphi := p | \neg \varphi | \varphi \land \varphi | \forall \varphi | \varphi S \varphi$$

where $p \in P$, $Y$ is the yesterday operator and $S$ is the since operator. We define the following common abbreviations: $\varphi_1 \lor \varphi_2 \equiv \neg (\neg \varphi_1 \land \neg \varphi_2)$, the once operator $O\varphi \equiv \text{true}\ S\varphi$, the historically operator $H\varphi \equiv \neg O \neg \varphi$, the weak-yesterday operator $WY\varphi \equiv \neg Y\neg \varphi$, and the propositional Boolean constants $true \equiv p \lor \neg p$, $false \equiv \neg true$. Also, $start \equiv \neg Y(\text{true})$ 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 $2^P$. We denote the length of $\tau$ by length($\tau$) = $n + 1$ and the last element of $\tau$ by 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 $s_i \cdots s_j$ obtained from $\tau$ starting from position $i$ and ending in position $j$. We define the satisfaction relation $\tau, i \models \varphi$, stating that $\varphi$ holds at instant $i$, as follows:

- $\tau, i \models p$ iff length($\tau$) $\geq 1$ and $p \in s_i$ (for $p \in P$);
- $\tau, i \models \neg \varphi$ iff $\tau, i \not\models \varphi$;
- $\tau, i \models \varphi_1 \lor \varphi_2$ iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \forall \varphi$ iff $\tau, i \geq 1$ and $\tau, i-1 \models \varphi$;
- $\tau, i \models \varphi_1 S \varphi_2$ iff there exists $k$, with $0 \leq k \leq i < \text{length}(\tau)$ 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 $\varphi$ is true in $\tau$, denoted $\tau \models \varphi$, if $\tau, \text{length}(\tau)-1 \models \varphi$. We denote by sub($\varphi$) the set of all subformulas of $\varphi$ obtained from the abstract syntax tree of $\varphi$ (De Giacomo and Vardi 2013). For instance, if $\varphi = a \land \neg Y(b \lor (c \lor d))$, where $a, b, c, d$ are atomic, then sub($\varphi$) = \{a, b, c, d, (c \lor d), b \lor (c \lor d), Y(b \lor (c \lor d)), \neg Y(b \lor (c \lor d)), a \land \neg Y(b \lor (c \lor d))\}. Thus, $\text{sub}(\varphi)$ defines the size of a PPLTL formula $\varphi$.

PPLTL can be used to specify goals on planning problems.
**Table 1: PDDL3 operators, their equivalent PPLTL and LTL\(f\) formulas.** Superscripts abbreviate nested temporal operators. \(θ\) is a propositional formula on planning fluents; \(X, WX, U, V\) and \(G\) are the next, weak next, eventually, until and always operators of LTL\(f\): \(ϕ1 Rϕ2 \equiv ¬(¬ϕ1 U ¬ϕ2)\), 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.

<table>
<thead>
<tr>
<th>PDDL3 Operator</th>
<th>PPLTL Formula</th>
<th>LTL(f) Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>at-end (θ)</td>
<td>(θ)</td>
<td>(F(θ ∧ end))</td>
</tr>
<tr>
<td>always (θ)</td>
<td>(Gθ)</td>
<td>(Gθ)</td>
</tr>
<tr>
<td>sometime (θ)</td>
<td>0(θ)</td>
<td>(Fθ)</td>
</tr>
<tr>
<td>sometime-(\neg) (θ)</td>
<td>(θ)</td>
<td>(Gθ)</td>
</tr>
<tr>
<td>at-most-(\neg) (θ)</td>
<td>(θ)</td>
<td>(Gθ)</td>
</tr>
<tr>
<td>hold-during (n_1 n_2 θ)</td>
<td>(θ \land Y'(start))(V)</td>
<td>(θ \land Y'(end))(V)</td>
</tr>
<tr>
<td>(Χ(θ \land end))(V)</td>
<td>(Χ(θ \land end))(V)</td>
<td>(Χ(θ \land end))(V)</td>
</tr>
<tr>
<td>(Χ(θ \land start))(V)</td>
<td>(Χ(θ \land start))(V)</td>
<td>(Χ(θ \land start))(V)</td>
</tr>
</tbody>
</table>

**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:

\[
ϕ₁ U ϕ₂ \equiv ϕ₂ ∨ (ϕ₁ ∧ X(ϕ₁ U ϕ₂)).
\]

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

\[
ϕ₁ S ϕ₂ \equiv ϕ₂ ∨ (ϕ₁ ∧ Y(ϕ₁ S ϕ₂)).
\]

Exploiting this equivalence, the formula decomposition can be computed by recursively applying the following transformation function \(pnf(·)\):

- \(pnf(p) = ϕ\);
- \(pnf(Yφ) = Yϕ\);
- \(pnf(φ₁ S ϕ₂) = pnf(ϕ₂) ∨ (pnf(φ₁) ∧ Y(φ₁ S ϕ₂))\);
• \( \text{pnf}(\phi_1 \land \phi_2) = \text{pnf}(\phi_1) \land \text{pnf}(\phi_2) \);
• \( \text{pnf}(\neg \phi) = \neg \text{pnf}(\phi) \).

For convenience, we add \( \text{pnf}(O\phi) = \text{pnf}(\phi) \lor Y(O\phi) \).

A formula resulting from the application of \( \text{pnf}(\cdot) \) 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 \( Y\phi \) in \( \text{pnf}(\phi) \) are such that \( \phi \in \text{sub}(\phi) \). It is easy to see that the following holds:

**Proposition 1.** Every PPLTL formula \( \phi \) can be converted to its PNF form \( \text{pnf}(\phi) \) in linear-time in the size of the formula (i.e., \( |\text{sub}(\phi)| \)). Moreover, \( \text{pnf}(\phi) \) is equivalent to \( \phi \).

**Proof.** Immediate from the definition of \( \text{pnf}(\cdot) \) and the semantics of PPLTL formulas. \( \square \)

Now, we show that to evaluate a PPLTL formula \( \phi \), we only need to keep track of the truth values of some key subformulas of \( \phi \). To do so, we introduce \( \Sigma_\phi \) as the set of propositions of the form \( Y\phi \):

• \( Y\phi \) for each subformula of \( \phi \) of the form \( Y\phi \);
• \( Y(\phi_1 \land \phi_2) \) for each subformula of \( \phi \) of the form \( \phi_1 \land \phi_2 \).

To keep track of the truth of each proposition in \( \Sigma_\phi \), we define a specific interpretation \( \sigma : \Sigma_\phi \to \{ T, \bot \} \):

\[
\sigma : \Sigma_\phi \to \{ T, \bot \}
\]

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

**Definition 2.** Let \( \sigma_i \) be a propositional interpretation over \( \mathcal{P} \), \( \sigma_i \) a propositional interpretation over \( \Sigma_\phi \), and \( \phi \) a PPLTL subformula in \( \text{sub}(\phi) \), we define the predicate \( \text{val}(\phi, \sigma_i) \), recursively as follows:

- \( \text{val}(p, \sigma_i) \iff s_i \models p \);
- \( \text{val}(Y\phi', \sigma_i) \iff \sigma_i \models Y\phi' \);
- \( \text{val}(\phi_1 \land \phi_2, \sigma_i) \iff \text{val}(\phi_1, \sigma_i) \land \text{val}(\phi_2, \sigma_i) \);
- \( \text{val}(\neg \phi', \sigma_i) \iff \neg \text{val}(\phi', \sigma_i) \).

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

Now, given a trace \( \tau = s_0 \ldots s_n \) over \( \mathcal{P} \), we compute a corresponding trace \( \tau^{[\cdot]} = s_0 \ldots s_n \) over \( \Sigma_\phi \), where:

- \( \sigma_0 = (Y\phi') \models \bot \) for each \( Y\phi' \in \Sigma_\phi \);
- \( \sigma_i = (Y\phi') \models \text{val}(\phi, \sigma_{i-1}, s_{i-1}), \) for all \( i \) with \( 0 < i \leq n \).

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

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

**Proof.** By structural induction on the formula \( \phi \):

- \( \phi = p. \) By definition of \( \text{val}(\cdot) \), \( \text{val}(p, \sigma_0, s_0) \iff s_0 \models p \);
- \( \phi = Y\phi'. \) By definition of \( \sigma_0, \sigma_0(Y\phi') = \bot \), and by the semantics, \( s_0 \models Y\phi' \). Therefore, the thesis holds.
- \( \phi = \phi_1 \land \phi_2, \) \( \text{val}(\phi_1 \land \phi_2, \sigma_1, s_1) \iff \text{val}(\phi_1, \sigma_1, s_1) \land \text{val}(\phi_2, \sigma_1, s_1) \);
- \( \phi = \neg \phi', \) \( \text{val}(\neg \phi', \sigma_1, s_1) \iff \neg \text{val}(\phi', \sigma_1, s_1) \).

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

**Theorem 1.** Let \( \phi \) be a PPLTL formula over \( \mathcal{P} \), \( \phi \in \text{sub}(\phi) \) a subformula of \( \phi \), \( \tau \) a trace over \( \mathcal{P} \), and \( \tau^{[\cdot]} \) the corresponding trace over \( \Sigma_\phi \). Then,

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

**Proof.** We prove the thesis by double induction on the length of the trace \( \tau \) and on the structure of the formula \( \phi \):

- 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 \( \tau_{n-1} \) of length \( n - 1 \):

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

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

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

To prove the claim, we now proceed by structural induction on the formula, knowing that \( \text{last}(\tau_{n-1}^{[\cdot]}) = s_n \), and \( \text{last}(\tau_{n-1} \cdot s_n) = s_n \):

- \( \phi = p. \) We have that \( \tau_{n-1} \cdot s_n \models p \iff s_n \models p \). For the \( \text{val}(\cdot) \) predicate we have that \( s_n \models p \iff \text{val}(p, \sigma_n, s_n) \). Therefore, the thesis holds.
- \( \phi = Y\phi'. \) We have that \( \tau_{n-1} \cdot s_n \models Y\phi' \iff \tau_{n-1} \models \phi'. \) By inductive hypothesis, \( \tau_{n-1} \models \phi' \iff \text{val}(\phi', \text{last}(\tau_{n-1}^{[\cdot]}), \text{last}(\tau_{n-1})). \) For the \( \text{val}(\cdot) \) predicate \( \text{val}(Y\phi', \sigma_n, s_n) \iff \sigma_n = Y\phi' \), which corresponds to \( \text{val}(\phi', \text{last}(\tau_{n-1}^{[\cdot]}), \text{last}(\tau_{n-1})). \) Hence, the thesis holds.
- \( \phi = \phi_1 \land \phi_2. \) In this case it suffices to remember that \( \tau_{n-1} \cdot s_n \models \phi_1 \land \phi_2 \iff \tau_{n-1} \cdot s_n \models \phi_2 \lor (\phi_1 \land Y(\phi_2)). \) On the other hand, \( \text{val}(\phi_1 \land \phi_2, \sigma_n, s_n) \iff \text{val}(\phi_2, \sigma_n, s_n) \lor (\text{val}(\phi_1, \sigma_n, s_n) \land \sigma_n = Y(\phi_2)). \) By structural induction we have that \( \tau_{n-1} \cdot s_n \models \phi_1 \land \phi_2 \iff \text{val}(\phi_1, \sigma_n, s_n) \), and \( \tau_{n-1} \cdot s_n \models \phi_2 \iff \text{val}(\phi_2, \sigma_n, s_n) \). Moreover, \( \tau_{n-1} \cdot s_n \models Y(\phi_1 \land \phi_2) \iff \tau_{n-1} \models \phi_1 \land \phi_2 \), and \( \sigma_n = \)
“Y(ϕ₁ S ϕ₂)” iff \( \text{val}(ϕ₁ S ϕ₂, \text{last}([rₙ₋₁]), \text{last}(τₙ₋₁)) \).

Finally, we have that \( \phi₁ S ϕ₂ \) iff \( \text{val}(ϕ₁ S ϕ₂, \text{last}([rₙ₋₁]), \text{last}(τₙ₋₁)) \) holds by induction on the length of the trace.

- \( φ = ϕ₁ ∨ ϕ₂ \) or \( φ = ¬ ϕ₁ \). 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 “Yφ” 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., (Baird 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 \( Γ = ⟨D, s₀, ϕ⟩ \), where \( D = ⟨2^F, A, α, τr⟩ \) is a planning domain, \( s₀ \) the initial state and \( ϕ \) a PPLTL goal, the compiled planning problem is \( Γ' = ⟨D', s'₀, G'⟩, \) where \( D' = ⟨2^F, A, α', τr'⟩ \) is the compiled planning domain, \( s'₀ \) the new initial state and \( G' \) the new reachability goal. Next, we describe the encoding from \( Γ \) to \( Γ' \) using their compact representations. In particular, \( α' \) and \( τr' \) will be induced by the introduction of axioms and a re-formulation of the actions’ preconditions and effects.

**Fluents.** \( F' \) contains the fluents of \( F \), as well as one fluent for each proposition “Yφ” in \( Σ_ϕ \) to keep track of propositional interpretations \( σᵢ \), and the set of predicates \( P_{der} \) defined below. Formally, \( F' = F ∪ Σ_ϕ ∪ P_{der} \).

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

Here, we include an axiom \( \text{val}_φ → ψ \) for every subformula \( ϕ ∈ \text{sub}(ψ) \), and \( P_{der} \) is defined as the set of all derived predicates \( \text{val}_φ \), i.e., \( P_{der} = \{ \text{val}_φ \mid ϕ ∈ \text{sub}(ψ) \} \). These axioms are intended to be such that the current state \((σᵢ, sᵢ) \models \text{val}_φ \) iff \( \text{val}(ϕ, σᵢ, sᵢ) \). By mimicking rules in Definition 2, we get the following axioms:

- \( \text{val}_p \leftarrow p \);
- \( \text{val}_ψ \leftarrow “Yφ” \);
- \( \text{val}_φ \leftarrow (\text{val}_φ ∨ (\text{val}_ψ ∧ “Y(ϕ₁ S ϕ₂)”)) \);
- \( \text{val}_ψ \leftarrow (\text{val}_ψ ∧ \text{val}_φ) \);
- \( \text{val}_φ \leftarrow ¬ \text{val}_φ \).

Clearly, we have that \((σᵢ, sᵢ) \models \text{val}_φ \) iff \( \text{val}(ϕ, σᵢ, sᵢ) \).

Hence, to define \( Γ' \), we build a set of axioms for every subformula \( ϕ \) in \( \text{sub}(ψ) \), i.e., \( \{ \text{val}_φ \leftarrow ϕ \mid ϕ ∈ \text{sub}(ψ) \} \).

Axioms allow us to elegantly model the mathematics of the previous section (i.e., the \( \text{val}(ϕ, σᵢ, sᵢ) \) 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 \( F \), whereas the new fluents “Yφ” in \( Σ_ϕ \) are assigned to the truth value given by \( σ₀ \). That is \( s₀ = (σ₀, s₀) \).

**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 “Yφ” in \( Σ_ϕ \), we model assignments updates by a set of conditional effects of the form:

\[
\begin{align*}
\text{val}_φ & \leftarrow “Yφ” \\
¬ \text{val}_φ & \leftarrow ¬ “Yφ”
\end{align*}
\]

These effects are exactly the same for every action \( a \in A \). Also, since \( σᵢ \) maintains values of “Yφ” in \( Σ_ϕ \) they are independent of the effect of the action on the original fluents, which, instead, is maintained in the propositional interpretation \( sᵢ \). 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 \( \text{Val}_φ = \{ \text{val}_φ \leftarrow “Yφ”, ¬ \text{val}_φ \leftarrow ¬ “Yφ” \mid “Yφ” \in Σ_ϕ \} \). The set of actions \( A \) in \( Γ' \) remains the same, as in the original problem \( Γ \). For all \( a ∈ A \), we have that the preconditions in \( Γ' \) are \( \text{Pre}_a = \text{Pre}_a \) and the effects in \( Γ' \) are \( \text{Eff}_a = \text{Eff}_a ∪ \text{Val}_φ \). Observe that, the auxiliary part \( \text{Val}_φ \) in \( \text{Eff}_a \) deterministically updates subformulas values in \( Σ_ϕ \), without affecting any fluent \( f ∈ F \) of the original domain model. This is crucial for the encoding’s correctness.

**Goal.** The goal in \( Γ' \) is encoded as \( G' = \{ \text{val}_φ \} \). That is, \( \text{val}(ϕ, σᵢ, sᵢ) \), 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 \( ϕ \).

**Proof.** Immediate, by analyzing the construction. □
Let \( \langle D', s'_0, G' \rangle \) be the corresponding compiled planning problem as previously defined. Any trace \( \tau' = s'_0, \ldots, s'_n \) on \( D' \) can be seen as \( \tau' = z\text{ip}(\tau^{[v]}, \tau) \), with \( \tau^{[v]} = a_0, \ldots, a_n \in (2^F)^+ \) and \( \tau = s_0, \ldots, s_n \in (2^F)^+ \), where each element of \( \tau' \) is of the form \( s'_i = (\sigma_i, s_i) \) for all \( i \geq 0 \). Here, we use the \( z\text{ip}(\cdot, \cdot) \) function to represent the aggregation of the two traces \( \tau^{[v]} \) and \( \tau \). Given a trace \( \tau' = s'_0, \ldots, s'_n \) on the encoded planning domain \( D' \), there exists a single trace \( \tau' \) such that \( \tau = s_0, \ldots, s_n \) on the original planning domain \( D \). Conversely, given a trace \( \tau = s_0, \ldots, s_n \) on the original planning domain \( D \), there exists a unique corresponding trace \( \tau^{[v]} \), and hence a single \( \tau' = z\text{ip}(\tau^{[v]}, \tau) \) on the encoded domain \( D' \). Finally, we observe that every executable action sequence \( a_0, \ldots, a_{n-1} \) in the planning problem \( \Gamma \) with PPLTL goal \( \varphi \) is also executable in the encoded planning problem \( \Gamma' \) (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 \( \Gamma \) be a planning problem with a PPLTL goal \( \varphi \), and \( \Gamma' \) 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 \( \Gamma \) iff \( \pi = a_0, \ldots, a_{n-1} \) is a plan for \( \Gamma' \).

**Proof.** Every executable action sequence \( a_0, \ldots, a_{n-1} \) in the planning problem \( \Gamma \) with PPLTL goal \( \varphi \) is also executable in the encoded planning problem \( \Gamma' \) (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 \( \pi \) is a plan if its induced state trace \( \tau \) is such that \( \tau \models \varphi \). By Theorem 1, we have that \( \tau \models \varphi \) iff \( \text{val}(\varphi, \text{last}(\tau^{[v]}), \text{last}(\tau)) \). However, by construction of the encoding for \( \Gamma' \), we have that \( \text{val}(\varphi, \text{last}(\tau^{[v]}), \text{last}(\tau)) \) holds iff \( \text{val}_{\varphi} \) holds in the last state of the induced state trace for \( \Gamma' \), i.e., in \( \tau' = z\text{ip}(\tau^{[v]}, \tau) \). In other words, \( \text{val}(\varphi, \text{last}(\tau^{[v]}), \text{last}(\tau)) \) iff \( \text{last}(\tau') = \text{val}_{\varphi} \). Hence, the thesis holds. \( \square \)

A direct consequence of Theorem 3 is that every sound and complete planner returns a plan \( \pi \) for the encoded planning problem \( \Gamma' \) if a plan \( \pi \) for the original planning problem \( \Gamma \) with PPLTL goal exists. If no solution exists for \( \Gamma' \), then no solution exists for \( \Gamma \).

**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 \( \phi \) encoding the truth value of \( \phi \) 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 McIraith 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 McIraith (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 FF\(_X\) (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 \( h_{bf} \) (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. FF\(_X\) 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-
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, 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 “\( h : \alpha \cup \beta \)” where \( \alpha \) or \( \beta \) have \( n \) nested \( \cup \) operators, for which we did not find an easy translation into PPLTL. For each \( \text{LTL}_f \) formula that we translated in PPLTL, we formally and automatically proved their semantic equivalence by verifying that the two formulations yield the same minimal \( \text{DFA} \). BF were designed in PPLTL directly, and analogously to what was done for TB, we formulated an equivalent formulation in \( \text{LTL}_f \). 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 \( \text{LTL}_f \) is \( O(on_{1,2} \land Y(O(on_{2,3} \land Y(O(on_{3,4} \land \ldots \land Y(O(on_{n-1,n}))))))) \). Its translation in \( \text{PPLTL} \) is \( \bigwedge_{j \in \{1,2,3,\ldots,n\}} O(on_{j-1,j} \land Y(\phi)) \) and \( \phi \) is formula \( O(on_{1,2} \land \bigwedge_{i \in \{3,4,\ldots,n\}} X(F(on_{i,2} \land X(F(on_{i,3} \land X(F(on_{i,4} \land \ldots \land X(F(on_{i,j-2} \land X(F(on_{i,j-1}))))))))) \). The translation in \( \text{LTL}_f \) is \( F(on_{1,2} \land X(F(on_{2,3} \land X(F(on_{3,4} \land X(F(on_{4,5} \land X(F(on_{5,6})))))))) \). For \( \text{LTL}_f \)'s search turned out to be slightly less powerful than both Poly and Exp over most instances. For the TB instances, P4P achieves the same coverage as Poly (the best \( \text{LTL}_f \)-based compilation) but is much faster in terms of average runtime: Poly 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 \( \text{LTL}_f \) 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 \( \text{LTL}_f \) compilations.

Indeed, if we look at Figure 1 (right), P4P compiles
In this paper, we studied classical planning for making their overall performance the same in these terms.  

The comparison on the number of expanded nodes 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 AlPlan4EU (No. 101016442), the ERC-ADG White-Mech (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


