FOND Planning for Pure-Past Linear Temporal Logic Goals

Luigi Bonassi\textsuperscript{a}, Giuseppe De Giacomo\textsuperscript{b,c}, Marco Favorito\textsuperscript{d,*}, Francesco Fuggitti\textsuperscript{c,e,f,**}, Alfonso Emilio Gerevini\textsuperscript{a} and Enrico Scala\textsuperscript{a}

\textsuperscript{a}University of Brescia, Italy
\textsuperscript{b}University of Oxford, UK
\textsuperscript{c}Sapienza University, Italy
\textsuperscript{d}Bank of Italy
\textsuperscript{e}York University, Canada
\textsuperscript{f}IBM Research, USA

\textbf{Abstract.} Recently, Pure-Past Temporal Logic (PPLTL) has proven highly effective in specifying temporally extended goals in deterministic planning domains. In this paper, we show its effectiveness also for fully observable nondeterministic (FOND) planning, both for strong and strong-cyclic plans. We present a notably simple encoding of FOND planning for PPLTL goals into standard FOND planning for final-state goals. The encoding only introduces few fluents (at most linear in the PPLTL goal) without adding any spurious action and allows planners to lazily build the relevant part of the deterministic automaton for the goal formula on-the-fly during the search. We formally prove its correctness, implement it in a tool called Plan4Past, and experimentally show its practical effectiveness.

1 Introduction

Planning for temporally extended goals has a long tradition in AI Planning, including pioneering work in the late ‘90s [2, 3, 4, 5], work on planning via Model Checking [15, 17, 20, 29, 34], and work on declarative and procedural constraints [8, 9, 7], and many others.

The presence of trajectory constraints in PDDL3 [28] also witnesses the relevance of temporally extended goals.

In fact, formalisms such as Linear Temporal Logic (LTL) have been advocated by the Formal Methods community [6] as excellent tools to express properties of processes. While these properties of processes often have an infinite horizon, in AI Planning, tasks need to terminate. Thus, a finite-trace variant of LTL, namely LTL\textsubscript{f}, is usually more appropriate [8, 9, 21, 22].

Although seminal works only focused on planning with deterministic actions and LTL or LTL\textsubscript{f} goals as, e.g., in [8, 9, 21, 38], more recently, there has been a surge of interest in the use of LTL\textsubscript{f} goals in fully observable nondeterministic planning settings (FOND) as demonstrated in [22, 13, 19, 12]. By now, we have a clear picture. The complexity of FOND planning for LTL\textsubscript{f} goals is EXPTIME-complete in the domain as for standard reachability goals (i.e., final-state goals) [35] and 2EXPTIME-complete in the LTL\textsubscript{f} goal formula [19]. Specifically, in nondeterministic domains, since LTL\textsubscript{f} goals can specify non-Markovian properties [25], the added expressiveness of LTL\textsubscript{f} goals worsens the worst-case goal complexity to 2EXPTIME-complete, compared to the EXPTIME-complete of reachability goals. That is because, in FOND planning, LTL\textsubscript{f} goals must be (implicitly or explicitly) translated into a Deterministic Finite-state Automaton (DFA), which is double exponential in the worst case.

Interestingly, an alternative to LTL\textsubscript{f} is the Pure-Past Linear Temporal Logic, or PPLTL [31, 18]. PPLTL evaluates traces backward and expresses non-Markovian properties on traces using past operators only. While PPLTL and LTL\textsubscript{f} are equally expressive, translating one into the other (and vice versa) is generally prohibitive since the best-known algorithms require 3EXPTIME [18]. However, due to a property of reverse languages [14], the DFA corresponding to a PPLTL formula can be computed in single exponential time directly from the formula, or better, its corresponding Alternating Finite-state Automaton (AFA) [18]. These unique characteristics have been exploited in [11] to devise a very effective technique to deal with PPLTL goals in deterministic planning domains.

In this paper, we show that such effectiveness is actually general and seamlessly extends also to nondeterministic planning domains, giving formal foundations for the solution technique. Specifically, we introduce an approach for PPLTL goals that sidestep altogether the construction of the DFA by introducing few new fluents (at most linear in the PPLTL goal) without adding any spurious action. The technique allows planners to lazily build the relevant part of the DFA on-the-fly symbolically during the planning search. To do so, we exploit the fixpoint characterization [26, 32, 23] of PPLTL formulas that, similarly to the one of LTL [10, 4], recursively 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. The solution to this recursion can be obtained by storing previous values of a small number of formulas (at most linear in the original formula), à la dynamic programming. We formally show the correctness of such an approach and demonstrate its effectiveness through a comprehensive experimental evaluation.

The fixpoint characterization of PPLTL formulas [26, 32, 23] has also been exploited in early research [2, 3, 37] within the context of (factorized) MDPs with PPLTL rewards. However, while these previ-
ous works primarily focused on MDPs, this paper addresses the problem of FOND planning, where PPTL is used to express temporally extended goals. Furthermore, here we formally prove our solution technique, which can be adapted to formally establish the correctness of the aforementioned previous approaches.

Summarizing, the contributions of the paper are the following: we devise an encoding of FOND planning for PPTL goals into standard FOND planning for reachability goals, which is at most linear in the size of the formula, formally correct, and readily implementable in PDDL. Moreover, we devise a variant of the encoding that avoids the use of derived predicates, which are difficult to handle for some FOND planners. Also, for this variant, we prove that it is still polynomial in the size of the formula, correct, and readily implementable in PDDL. We implemented both encodings in the tool Plan4Past, which can be used along with state-of-the-art FOND planners, and we empirically demonstrate its practical effectiveness.

2 FOND Planning for Temporally Extended Goals

A Fully Observable NonDeterministic (FOND) domain model can be formalized as a tuple $D = (F, F_{der}, \mathcal{X}, A, pre, eff)$ where $F$ is a set of positive literals, $F_{der}$ is a set of derived predicates, $\mathcal{X}$ is a set of axioms, $A$ is a set of action labels, $pre$ and $eff$ are two functions that define the preconditions and effects of each action $a \in A$. A planning state $s$ is a subset of $F$, and a positive literal $f$ holds true in $s$ if $f \in s$; otherwise, $f$ is false in $s$. Axioms have the form $d \rightarrow \psi$ where $d \in F_{ax}$ and $\psi$ is a formula over $\mathcal{F}$. An axiom $d \rightarrow \psi$ specifies that $d$ is derived from a true state $s$ if and only if we can prove that $s \models \psi$, possibly using other axioms from $\mathcal{X}$. We assume the set of axioms $\mathcal{X}$ is stratified [30]; this guarantees that given a state $s$ and a derived predicate $d$, it is possible to efficiently and uniquely determine whether $d$ holds true in $s$. Thus, it is always possible to determine whether a formula $\psi$ over $\mathcal{F}$ is satisfied by a state $s$. Both functions $pre$ and $eff$ take an action label $a \in A$ as an input and return a propositional formula over $\mathcal{F}$ and $\mathcal{F}_{der}$ and a set $\{eff_1, \ldots, eff_n\}$ of effects, respectively. Each effect $eff_i \in eff(a)$ is a set of conditional effects each of the form $c \rightarrow e$, where $c$ is a propositional formula over $\mathcal{F}$ and $\mathcal{F}_{der}$ and $e \in \mathcal{F} \cup \{-f | f \in \mathcal{F}\}$ is a set of literals from $\mathcal{F}$.

An action $a$ can be applied in a state $s$ if $pre(a)$ holds true in $s$ (i.e., $s \models pre(a)$). A conditional effect $c \rightarrow e$ is triggered in a state $s$ if $c$ is true in $s$. Applying $a$ in $s$ yields a successor state $s'$ determined by an outcome nondeterministically drawn from $eff(a)$. Let $eff \in eff(a)$ be the chosen nondeterministic effect, the new state $s'$ is such 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 $c \rightarrow e \in eff$ triggered in $s$ deletes it ($\neg f \in e$) or (ii) there is a conditional effect $c \rightarrow e \in eff$, triggered in $s$ that adds it ($f \in e$). In case of conflicting effects, similarly to other works [36], we assume delete-before-adding semantics. We use $tr(s, a)$ to denote the set of possible successor states $\{s'_1, \ldots, s'_n\}$ obtained by executing $a$ in $s$. Note that if $s \not\models pre(a)$ then $tr(s, a) = \emptyset$. A FOND planning problem is a tuple $\Gamma = (D, s_0, G)$, where $D$ is a domain model, $s_0 \subseteq \mathcal{F}$ is the initial state, and $G$ is a formula over $\mathcal{F}$ and $\mathcal{F}_{der}$, also called the reachability goal.

Solutions to $\Gamma$ are strategies (aka policies). A strategy is defined as a partial function $\pi : (2^D)^+ \rightarrow A$ mapping a sequence of non-goal states into an applicable action. A strategy $\pi$ for $\Gamma$, starting from the initial state $s_0$, induces a set of (possibly infinite) state trajectories (or executions) $\Lambda$ of the form $\pi = s_0, s_1, \ldots$, where $s_0$ is the initial state, $s_{i+1} \in tr(s_i, a_i)$, and $a_i = \pi(s_0, \ldots, s_i)$ for $i \geq 0$. If for a certain sequence of states $\pi = s_0, \ldots, s_n$, we have that $\pi(\tau)$ is undefined (no action prescribed), then the generated execution $\tau$ is a finite trace. As usual, we consider two kinds of solutions to FOND planning problems: strong solutions and strong-cyclic solutions [16].

A strategy $\pi$ is a strong solution to $\Gamma$ if every generated execution is a finite trace $\tau$ such that $s_n \models G$. A strategy is a strong-cyclic solution to $\Gamma$ if every generated execution that is a stochastic-fair trace is also a finite trace such that $s_n \models G$, cf. [1]. When a strategy $\pi$ is a solution (either strong or strong-cyclic, depending on the kind of solution we are interested in), we say that $\pi$ is winning.

Many real-world scenarios require achieving more general goals than just reachability goals. In most cases, realistic goals require properties to hold over time on a sequence of planning states. These types of temporal goals, also called temporally extended goals, are often expressed using temporal logics on finite or infinite sequences of states. FOND planning for temporally extended goals has been recently studied, e.g., in [13, 19, 12]. Here, we consider goals expressed in Pure-Past Linear Temporal Logic (PPTL), which we review in the next section.

Definition 1. A FOND planning problem for temporally extended goals is a tuple $\Gamma = \langle D, s_0, \varphi \rangle$, where $D$ is a nondeterministic domain model, $s_0$ is the initial state, and $\varphi$ is a temporal formula defined over $\mathcal{F}$.

In the case of temporally extended goals, the definition of strategy changes to a partial function $\tau : (2^D)^+ \rightarrow A$ mapping traces into applicable actions. Hence, given a temporal goal $\varphi$, $\pi$ is a solution to $\Gamma$ if every state trajectory induced by $\pi$ is finite and satisfies $\varphi$. When the strategy needs only finite memory, then it can be represented as a finite-state transducer [19].

3 Pure-Past Linear Temporal Logic

PPTL is the variant of LTL $f$ that refers to the past only. A survey on PPTL can be found in [18], where it is denoted as LTL $f$. Given a set $P$ of propositions, PPTL formulas are defined as:

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

where $p \in P$, $Y$ is the yesterday operator and $S$ is the since operator. We define the following common abbreviations: $\varphi_1 \land \neg \varphi_2 \equiv \neg (\neg \varphi_1 \land \varphi_2)$, the once operator $O \varphi \equiv true \land \varphi$, the historically operator $H \varphi \equiv \neg O \neg \varphi$, the weak-yesterday operator $W Y \varphi \equiv \neg Y \neg \varphi$, and the propositional Boolean constants $true \equiv p \lor \neg p$, $false \equiv \neg true$. Also, start $\neg Y(\text{true})$ expresses that the trace has started.

PPTL formulas are interpreted on finite nonempty traces, also called histories, $\tau = s_0, \ldots, s_n$, where $s_i$ at instant $i$ is a propositional interpretation over the alphabet $\mathcal{P}^\tau$. We denote by length($\tau$) the length $n+1$ of $\tau$, and by last($\tau$) the last element of the trace. Given a trace $\tau = s_0, \ldots, s_n$, we denote by $\tau_i$, with $0 \leq i \leq n$, the sub-trace $s_1, \ldots, s_i$ 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:

$$\begin{align*}
\tau, i \models & p \text{ if } \text{length}(\tau) \geq 1 \text{ and } p \in s_i \text{ (for } p \in P); \\
\tau, i \models & \neg \varphi \text{ if } i \not\models \varphi; \\
\tau, i \models & \varphi_1 \land \varphi_2 \text{ if } \tau, i \models \varphi_1 \text{ and } \tau, i \models \varphi_2; \\
\tau, i \models & Y \varphi \text{ if } i \geq 1 \text{ and } \tau, i-1 \models \varphi; \\
\tau, i \models & \varphi_1 S \varphi_2 \text{ if } \exists k, 0 \leq k \leq i < \text{length}(\tau) \text{ such that } \tau, k \models \varphi_2 \text{ and for all } j, k < j \leq i, \text{ we have that } \tau, j \models \varphi_1.
\end{align*}$$
A PPTL formula $\varphi$ is true in $\tau$, denoted $\tau \models \varphi$, if $\tau$, length($\tau$) $- 1 \models \varphi$. We denote by $\text{sub}(\varphi)$ the set of all subformulas of $\varphi$ [21]. For instance, if $\varphi = a \land \neg Y(b \lor c)$, where $a$, $b$, and $c$ are atomic, then $\text{sub}(\varphi) = \{a, b, c, (b \lor c), \neg Y(b \lor c), Y(b \lor c), a \land \neg Y(b \lor c)\}$. Subformulas correspond to the nodes of the syntactic tree of the formula, so they are linearly many. Also, $|\text{sub}(\varphi)|$ defines the size of a PPTL formula $\varphi$.

**Theorem 1** ([18]). PPTL is as expressive as LTL$_f$, i.e., as star-free regular expressions and first-order logic on finite sequences.

Now, we review the theoretical foundations and properties of PPTL, presented in [18, 11], that we exploit later in the paper. While PPTL is as expressive as LTL$_f$, it gives an exponential worst-case theoretical computational advantage in several applications. In fact, both LTL$_f$ and PPTL can be translated into an equivalent AFA in linear time. However, given that an AFA can be directly transformed in single exponential time into a DFA recognizing the reverse language [14], the conversion of PPTL formulas to their corresponding DFAs is worst-case single exponential time (vs. double exponential time for LTL$_f$ formulas) [18]. To exploit such an interesting computational advantage, properties of interest should naturally be expressible in PPTL, as the best-known algorithms to translate LTL$_f$ into PPTL (and vice-versa) are 3EXPTIME [18], and no automated tool exists. Nevertheless, many interesting properties expressed in PPTL, including the well-known DECLARE templates [39] and PDDL3 operators [28], are actually polynomially related to their translation into PPTL (and vice versa) [11, 27].

Given these observations, we can implicitly and incrementally build a DFA for the PPTL formula while looking for a strategy (i.e., during the search). To do so, we exploit the well-known fixpoint characterization of LTL and variants [26, 32, 23]. For PPTL, 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. In particular, PPTL formulas can be decomposed into present and past components, given the fixpoint characterization of the since operator: $\psi_1 S \psi_2 \equiv \psi_2 \lor (\psi_1 \land Y(\psi_1 S \psi_2))$. Given this equivalence, the formula decomposition can be computed by recursively applying the following transformation function $\text{pnf}(\cdot)$:

- $\text{pnf}(p) = p$;
- $\text{pnf}(Y\psi) = Y\psi$;
- $\text{pnf}(\psi_1 S \psi_2) = \text{pnf}(\psi_2) \lor (\text{pnf}(\psi_1) \land Y(\text{pnf}(\psi_1 S \psi_2)))$;
- $\text{pnf}(\psi_1 \land \psi_2) = \text{pnf}(\psi_1) \land \text{pnf}(\psi_2)$;
- $\text{pnf}(\neg \psi) = \neg \text{pnf}(\psi)$.

A formula resulting from the application of $\text{pnf}(\cdot)$ is in Previous Normal Form (PNF). 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 $Y\psi$ in $\text{pnf}(\varphi)$ are such that $\psi \in \text{sub}(\varphi)$. It is easy to see that the following proposition holds:

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

Interestingly, the PNF decomposition allows evaluating PPTL formulas by only knowing the truth value of some of its subformulas, those that are within the Y-scope. We interpret these specific subformulas as atomic propositions, denoting them with quotes, and collecting them in a set denoted as $\Sigma_\varphi$. Specifically, these propositions are: (i) “$Y\psi$” for each subformula of $\varphi$ of the form $Y\psi$, and (ii) “$\psi_1 S \psi_2$” for each subformula of $\varphi$ of the form $\psi_1 S \psi_2$. For instance, for the formula $a S Y b$, we only have $\Sigma_\varphi = \{\text{"$Y(a S Y b)$", \"$Yb$"}\}$. To evaluate the truth value of propositions in $\Sigma_\varphi$, we use the interpretation $\sigma : \Sigma_\varphi \rightarrow \{\top, \bot\}$, and characterize the evaluation of subformulas at a certain instant $i \leq n$ with $\sigma_i$. Intuitively, given an instant $i$, $\sigma_i$ tells us which propositions related to the previous instant (i.e., in $\Sigma_\varphi$) are true at the instant $i$. Therefore, a PPTL formula can simply be evaluated 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** ([11]). Let $s_i$ be a propositional interpretation over $\mathcal{P}$, $\sigma_i$ a propositional interpretation over $\Sigma_{\varphi}$, and $\phi$ a PPTL subformula in $\text{sub}(\varphi)$, the predicate $\text{val}(\phi, \sigma_i, s_i)$ is recursively defined as:

- $\text{val}(p, \sigma_i, s_i) \iff s_i \models p$;
- $\text{val}(Y\psi, \sigma_i, s_i) \iff \sigma_i \models \text{\"$Y\psi$\"}$;
- $\text{val}(\psi_1 S \psi_2, \sigma_i, s_i) \iff \text{\val}(\psi_2, \sigma_i, s_i) \lor (\text{\val}(\psi_1, \sigma_i, s_i) \land \sigma_i \models \text{\"$Y(\psi_1 S \psi_2)$\"})$;
- $\text{val}(\psi_1 \land \psi_2, \sigma_i, s_i) \iff \text{\val}(\psi_1, \sigma_i, s_i) \land \text{\val}(\psi_2, \sigma_i, s_i)$;
- $\text{val}(\neg \psi, \sigma_i, s_i) \iff \neg \text{\val}(\psi, \sigma_i, s_i)$.

The $\text{val}(\phi, \sigma_i, s_i)$ predicate determines what is the truth value of any PPTL formula $\phi \in \text{\sub}(\varphi)$ by reading a propositional interpretation $s_i$ from time $\tau$ and keeping track of the truth value of propositions in $\Sigma_\varphi$ by means of $\sigma_i$. Now, consider a trace $\tau = s_0, \ldots, s_n$ over $\mathcal{P}$, we can always compute a corresponding trace $\tau^{[i]} = s_0, \ldots, s_n$ over $\Sigma_\varphi$, where $\sigma_0(\text{\"$Y\psi$\"}) = \bot$ for each “$Y\psi$” in $\Sigma_\varphi$ and $\sigma_i(\text{\"$Y\psi$\"}) = \text{\val}(\phi, \sigma_{i-1}, s_{i-1})$, for all $i$ with $0 < i \leq n$. The following result holds.

**Theorem 2** ([11]). Let $\tau$ be a PPTL formula over $\mathcal{P}$, $\phi \in \text{\sub}(\varphi)$ a subformula of $\varphi$, $\tau$ a trace over $\mathcal{P}$, and $\tau^{[i]}$ the corresponding trace over $\Sigma_\varphi$. Then, $\tau \models \phi \iff \text{\val}(\phi, \sigma_n, s_n)$.

Theorem 2 states that a PPTL formula can be evaluated on the last instant of a trace $\tau$ by computing the $\text{\val}(\cdot)$ predicate on the last propositional interpretation of $\tau$ and on what propositions are true in $\Sigma_\varphi$ at the last instant. Moreover, as previously mentioned, Theorem 2 can be seen as another way to build the DFA corresponding to the PPTL formula $\varphi$. In fact, by computing which subformulas of $\varphi$ are true at every instant while scanning a given trace $\tau$, one is implicitly building the states of the DFA corresponding to $\varphi$ on-the-fly.

## 4 FOND Planning for PPTL Goals

We devise a particularly effective technique for FOND planning for PPTL goals by exploiting the result in Theorem 2. Given that PPTL formulas can be evaluated on traces generated by sequences of planning actions, the key idea is to keep track of the truth of the (quoted) propositions representing subformulas of the PPTL goal in each planning state and update them correspondingly while the planning process goes on. In this way, we sidestep altogether the standard construction based on computing the automaton for the PPTL goal $\varphi$ and then building the cross-product between such an automaton and the automaton corresponding to the domain, e.g., see [19].

Given a FOND planning problem $\Gamma = (\mathcal{D}, s_0, \varphi)$, where $\mathcal{D} = (\mathcal{F}, \mathcal{F}_d, \mathcal{X}, A, \text{pre}, \text{eff})$ is a nondeterministic domain, $s_0$ the initial state and $\varphi$ a PPTL goal, the encoded FOND planning problem is $\Gamma' = (\mathcal{D}', s_0', G')$, where $\mathcal{D}' = (\mathcal{F}', \mathcal{F}'_d, \mathcal{X}', A, \text{pre}, \text{eff}')$ is the encoded FOND planning domain, $s_0'$ is the new initial state and $G'$ is...
Theorem 4. Let \( \varphi \) be a PPLTL formula over \( \mathcal{P} \), \( \varphi \in \text{sub}(\varphi) \) a subformula of \( \varphi \), \( \tau \) a trace over \( \mathcal{P} \), and \( \tau^{[\varphi]} \) the corresponding trace over \( \Sigma_{\mathcal{P}} \). Then, \( \text{val}(\varphi, \sigma_i, s_n) \iff (\sigma_i, s_n) \models \varphi \).

Now, we examine the correctness. Let \( \Gamma' = (\mathcal{D}, s_0, \varphi') \) be a FOND planning problem, with a nondeterministic domain \( \mathcal{D} \), the initial state \( s_0 \), and a PPLTL goal formula \( \varphi' \), and let \( \Gamma'' = (\mathcal{D}', s'_0, G') \) be the corresponding encoded planning problem as previously defined.

Any trace \( \tau' = s'_0, \ldots, s'_n \) on \( \mathcal{D}' \) can be seen as \( \tau' = \text{zip}(\tau^{[\varphi]}, \tau) \), where \( \tau = s_0, \ldots, s_n \in (2^{\mathcal{F}_s})^+ \), \( \tau^{[\varphi]} = s_0, \ldots, s_n \in (2^{\mathcal{F}_s})^+ \), where each element of \( \tau^{[\varphi]} \) is of the form \( s_i = (\sigma_i, s_i) \) for all \( i \geq 0 \). Here, we use the \( \text{zip}(\cdot, \cdot) \) function to represent the aggregation of the two traces \( \tau^{[\varphi]} \) and \( \tau \). Given a trace \( \tau' = s'_0, \ldots, s'_n \) on the encoded planning domain \( \mathcal{D}' \), there exists a single trace \( \tau' = \tau = s_0, \ldots, s_n \) 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 \( \tau' = \text{zip}(\tau^{[\varphi]}, \tau) \) on the encoded domain \( \mathcal{D}' \).

For every strategy \( \pi : (2^{\mathcal{F}_s})^+ \rightarrow A \) for the FOND planning problem \( \Gamma' \) with PPLTL goal \( \varphi \), we can build the strategy \( \pi' : (2^{\mathcal{F}_s})^+ \rightarrow A \) for \( \Gamma'' \) as follows:

\[
\pi'(\tau') = a \iff \pi(\tau') = a
\]
\[
\pi'(\tau') \text{ is undefined} \iff \pi(\tau') \text{ is undefined.}
\]

Lemma 1. For every \( \pi : (2^{\mathcal{F}_s})^+ \rightarrow A \) that is a (strong or strong-cyclic) winning strategy for the FOND planning problem \( \Gamma' \) with PPLTL goal \( \varphi \), the corresponding \( \pi' : (2^{\mathcal{F}_s})^+ \rightarrow A \) is also a (strong or strong-cyclic, respectively) winning strategy for the encoded planning problem \( \Gamma'' \).

Now we consider the converse. For every strategy \( \pi' : (2^{\mathcal{F}_s})^+ \rightarrow A \) for the encoded planning problem \( \Gamma'' \), we can build the strategy \( \pi : (2^{\mathcal{F}_s})^+ \rightarrow A \) for the original problem \( \Gamma' \) with PPLTL goal \( \varphi \) as follows (where \( \tau' = \text{zip}(\tau^{[\varphi]}, \tau) \)):

\[
\pi(\tau) = a \iff \pi'(\tau') = a
\]
\[
\pi(\tau) \text{ is undefined} \iff \pi'(\tau') \text{ is undefined.}
\]

Lemma 2. For every \( \pi' : (2^{\mathcal{F}_s})^+ \rightarrow A \) that is a (strong or strong-cyclic) winning strategy for the encoded planning problem \( \Gamma'' \), the corresponding \( \pi : (2^{\mathcal{F}_s})^+ \rightarrow A \) is also a (strong or strong-cyclic, respectively) winning strategy for the FOND planning problem \( \Gamma' \) with PPLTL goal \( \varphi \).

By Lemma 1 and Lemma 2, we immediately get:

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

As a result, let \( \Gamma' \) be a FOND (strong or strong-cyclic) planning problem with a PPLTL goal \( \varphi \), and \( \Gamma'' \) be the corresponding encoded FOND (strong or strong-cyclic, resp.) planning problem with reachability goal \( G' \). Then, every sound and complete planner (FOND strong or FOND strong-cyclic, resp.) returns a winning strategy \( \pi' \) for \( \Gamma' \) if a winning strategy \( \pi \) for \( \Gamma \) exists. If no solution exists for \( \Gamma' \), then there is no solution for \( \Gamma \).

Here, it is important to observe that strategies returned by a FOND planner for \( \Gamma'' \) are going to be memory-less policies of the form \( \Pi'(s') = a \text{ or } \Pi'(s') \text{ undefined} \) at the goal. These can be immediately transformed in trace-based strategies by defining:

\[
\pi(\tau') = a \iff \Pi'(\text{last}(\tau')) = a
\]
\[
\pi(\tau') \text{ is undefined} \iff \Pi'(\text{last}(\tau')) \text{ is undefined.}
\]

This possibility is crucial since strategies for the original problem \( \Gamma \) with a PPLTL goal \( \varphi \) must be memory-full strategies. In other words,
they need to be finite-state controllers or transducers. We can use \( \sigma_i \) of \( s'_i = (\sigma_i, s_i) \) as the state of the transducer, \( \sigma_{i+1} = \text{val}(\phi, \sigma_i, s_i) \) (for each \( \text{"Y} \phi \text{"} \in \Sigma_F \)) as the factorized transition function, and \( \Pi'(s'_i) \) as the output function of the transducer.

5 Encoding without Derived Predicates

Here, we describe a variant of the encoding presented in Section 4 that does not add derived predicates. Like the previous encoding, this variant introduces a fresh atom of the form \( \text{"Y} \phi \text{"} \) for each temporal component of \( \varphi \). However, instead of using the \( \text{val}() \) predicate, it combines quoted atoms with propositions of the original domain to explicitly represent the PNF of a formula and does so by representing the formula in PPNF (Propositional Previous Normal Form).

Definition 3. Let \( \varphi \) be a PPLTL formula. \( \text{ppnf}(\varphi) \) is a propositional formula obtained by substituting every \( \text{Y} \phi \) with \( \text{"Y} \phi \text{"} \) in \( \text{pnf}(\varphi) \).

For example, if \( \varphi = a \land b \) then \( \text{ppnf}(\varphi) = b \lor (a \land \text{"Y} (a \land b) \text{"}) \).

For any formula \( \varphi \), the \( \text{ppnf}(\varphi) \) captures the truth of \( \text{pnf}(\varphi) \) without using temporal operators, provided that every \( \text{Y} \phi \in \Sigma_F \) reflects the truth of \( \text{Y} \phi \). Most importantly, \( \text{ppnf}(\varphi) \) is linear in the size of \( \varphi \).

Lemma 3. Let \( \varphi \) be a PPLTL formula. The size of \( \text{ppnf}(\varphi) \) is linear in the size of \( \varphi \).

Given a FOND planning problem \( \Gamma = \langle D, s_0, \varphi \rangle \) where \( D = \langle F, F_{\text{der}}, X, A, \text{pre}, \text{eff} \rangle \) and \( \varphi \) is a PPLTL goal, Table 2 formalizes the encoding of an equivalent FOND problem \( \Gamma'' = \langle D'', s'_0, G'' \rangle \) with \( D'' = \langle F'', F_{\text{der}}, X, A, \text{pre}, \text{eff}'' \rangle \) for a goal \( G'' \). Compared to the encoding of Table 1, the function \( \text{eff}'' \) uses the propositional formula \( \text{ppnf}(\phi) \) to update the truth of each quoted atom \( \text{Y} \phi \) in \( \Sigma_F \) after every action. Intuitively, if \( \text{ppnf}(\phi) \) holds in \( s' \), then \( \text{Y} \phi \) will hold in the successor state \( s'' \), and we keep track of this by ensuring that \( \text{Y} \phi \) holds true in \( s'' \), formalized via the conditional effect \( \text{ppnf}(\phi) \circ \text{Y} \phi \). Analogously, if a state \( s' \) does not satisfy \( \text{ppnf}(\phi) \), then the conditional effect \( \neg \text{ppnf}(\phi) \circ \text{Y} \phi \) will make \( \text{Y} \phi \) false in \( s'' \). These effects are added to each outcome of an action \( a \). Finally, goal \( G'' \) asks to satisfy \( \text{ppnf}(\varphi) \).

Table 2. Components of the encoded FOND planning problem

<table>
<thead>
<tr>
<th>Components</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluents ( F'' )</td>
<td>( F'' ) := ( F \cup \Sigma_F )</td>
</tr>
<tr>
<td>Derived Predicates ( X_{\text{der}}, X_{\text{der}}, i.e., unchanged</td>
<td></td>
</tr>
<tr>
<td>Axioms ( X )</td>
<td>( X := X ), i.e., unchanged</td>
</tr>
<tr>
<td>Action Labels ( A )</td>
<td>( A := A ), i.e., unchanged</td>
</tr>
<tr>
<td>Preconditions ( \text{pre} )</td>
<td>( \text{pre}(a) := \text{pre}(a) ) for every ( a \in A ), i.e., unchanged</td>
</tr>
<tr>
<td>Effects ( \text{eff}'' )</td>
<td>( \text{eff}''(a) := { [\text{eff} \circ \text{ppnf}(\phi) \mid \phi, \text{eff} \in \text{eff}(a)] \mid \phi \text{pre} } = { \text{ppnf}(\phi) \circ \text{Y} \phi } } \cup { \neg \text{Y} \phi } } } \cup \Sigma_F )</td>
</tr>
<tr>
<td>Initial state ( s''_0 )</td>
<td>( s''_0 := s_0 \cup s''_0 )</td>
</tr>
<tr>
<td>Goal ( G'' )</td>
<td>( G'' := \text{ppnf}(\varphi) )</td>
</tr>
</tbody>
</table>

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

Theorem 7. Let \( \Gamma \) be a FOND planning problem. The size of \( \Gamma'' \) obtained following the encoding of Table 2 is polynomial in the size of \( \Gamma \). In particular, it is linear in the size of the domain specification and quadratic in the size of the goal.

6 Experimental Evaluation

We implemented the approaches of Sections 4 and 5 in a tool called Plan4Past\(^1\), which can be run with either the intensive conditional effects compilation (P4P for short) or with derived predicates (P4P\(_X\) for short). Both configurations take as input a FOND planning problem written in PDDL and a PPLTL formula and give as output a PDDL description of a new FOND problem.

Preliminary experiments revealed how current FOND planners struggle to preprocess most problems compiled by P4P. To overcome this issue, we further optimize this encoding by aggregating the set \( \text{eff}_{\text{pseud}} \) of new conditional effects into a dummy action check. Then, we force the encoding to always execute one occurrence of check before any domain actions. Clearly, such a modification does not undermine our theoretical results and instead proves to be much more convenient with the current FOND planners. Thus, we will hereinafter refer to P4P as the compilation with this modification.

We used two state-of-the-art FOND planners: PRP [33] and Paladinus [24]. Both engines support basic conditional effects. However, although Paladinus fully support axioms, it does not support disjunctive conditional effects. Conversely, PRP supports conditional effects with disjunctive conditions but does not support axioms. Hence, since P4P requires disjunctive conditional effects and no axioms while P4P\(_X\) requires conjunctive conditional effects and axioms, we use P4P with PRP (P4P\(_{PP} \) for short), and Paladinus with P4P\(_X\) (P4P\(_{PPX} \) for short).

The empirical analysis aims to evaluate the effectiveness of temporally extended goals expressed in PPLTL and handled by Plan4Past, and semantically equivalent (polynomially related) temporally extended goals expressed in LTL\(_f\) and handled by ltlfond2fond [13] (ltlf2f for short). ltlf2f is a compilation that explicitly computes an automaton representing the LTL\(_f\) temporal goal. To our knowledge, this is the best approach for planning with LTL\(_f\) goals in FOND domains. The advantage of P4P seems clear: ltlf2f is exponential, while both compilations performed by P4P and P4P\(_X\) are polynomial in the size of the PPLTL goals. Yet, from a practical standpoint, the impact of this advantage in FOND planning is unclear. To this end, we tested the three systems over a benchmark set and measured the number of problems solved (Coverage), the time taken to get a solution (computation plus search), and the size of the policies (number of state-action pairs). Like for P4P, the problems encoded by ltlf2f are supported by FOND planners with conditional effects, but our findings show that ltlf2f performs significantly better with PRP. Hence, we only present the results of ltlf2f with PRP. All experiments ran up to 1800s on a Xeon-Gold 6140M 2.3 GHz with 8GB of memory.

6.1 Benchmark Domains

Our benchmark suite features the FOND domains ROVERS, BLOCKS, and COFFEE used in [13]. We tested all compilations on the same instances by [13] (C17 for short). C17 temporally extended goals were

---

\(^1\) Source code, benchmarks and supplementary material are publicly available at https://github.com/whitemech/Plan4Past
originally specified in $LTL_f$, and therefore, for comparison reasons, we manually translated them to $PPLTL$. Recall that although $LTL_f$ and $PPLTL$ have the same expressiveness, no tool to translate one into the other exists yet. Therefore, for each $LTL_f$ formula translated in $PPLTL$, we formally and automatically proved their semantic equivalence by verifying that the two formulations yield the same minimal DFA (modulo state renaming). We observed that all C17 instances were trivially solved by the three systems. Therefore, to study the scalability of the compilations, we generated a new set of instances of increasing dimensions for each domain (BF23 for short). The C17 instances are publicly available, while our newly generated instances are described below, along with other information about the domain and the translations from $PPLTL$ and $LTL_f$.

**Blocksworld.** In blocksworld, we want to arrange blocks in a particular configuration. An arm can pick-up and move blocks on top of other blocks or on the table. Starting from the 24 C17 instances, we generated bigger problems considering four types of formula employed by [13]:

\[
\begin{align*}
F(a) & \lor (F(b_1) \land F(b_2) \land \ldots \land F(b_n)) & (1) \\
(F(b_1) \lor F(b_2) \lor \ldots \lor F(b_n)) \lor F(a) & (2) \\
F(b_1) & \land F(b_2) \land \ldots \land F(b_n) \land a & (3) \\
(a \land F(b_1)) & \land (a \land F(b_2)) \land \ldots \land (a \land F(b_n)) & (4)
\end{align*}
\]

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

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

\[
F(\text{on}_{a,c} \land \text{on}_{b,c} \land \text{XF}(\text{on}_{a,b} \land \text{on}_{b,a} \land \text{XF}(\text{on}_{a,t} \land \text{on}_{b,t} \land \text{on}_{c,t}))),
\]

whereas in $PPLTL$ it is expressed as:

\[
O(\text{on}_{a,t} \land \text{on}_{b,t} \land \text{on}_{c,t}) \land YO(\text{on}_{a,b} \land \text{on}_{b,a} \land YO(\text{on}_{a,c} \land \text{on}_{c,b}))).
\]

We generated a new instance with a goal $\text{seq}_{i,j}$ for $i = 3, 4$ and with $j = 1, \ldots, i$, for a total of 30 “seq” problems. Therefore, the total number of instances for $\text{BLOCKS}$ sums up to 94 instances, 24 original instances from [13], 40 instances that we generated by scaling the temporal goals from [13], and 30 “seq” instances.

**Rovers.** Here, the goal is to gather data about soil, rocks, and images of a planet by planning the activities of one or more rovers. We used the 9 instances in C17 and generated other larger instances with 4 types of formulas:

\[
\begin{align*}
F(g_1) & \land F(g_2) \land \ldots \land F(g_n) & (1) \\
F(F(g_1) \land F(g_2) \land \ldots \land F(g_n)) & (2) \\
F(F(g_1) \lor F(g_2) \lor \ldots \lor F(g_n)) & (3) \\
G(F(G(g_1) \lor G(g_2) \lor \ldots \lor G(g_n))) & (4)
\end{align*}
\]

where every $g_i$ is an atom representing the completion of a task, i.e., the communication of data about soil, rock, or image. The first two formulas require that each task is accomplished in some state of the trajectory induced by the execution of the policy. Formula (3) requires that at least one task is eventually completed, while the last formula requires that all tasks are satisfied in the last state. The translation for formulas (1), (2), and (3) is direct; we can simply swap future operators with their past counterparts. For instance, we write formula (2) in $PPLTL$ as $O(O(g_1) \land O(g_2) \land \ldots \land O(g_n))$. The last formula requires one of the goals to be achieved in the last state; this could be represented in $PPLTL$ without using any temporal operator. However, for the sake of a fair comparison, we translated such $LTL_f$ formula into a $PPLTL$ formula with the same number of nested temporal operators, i.e.,

\[
H(H(\text{H}(g_1) \lor \text{H}(g_2) \lor \ldots \lor \text{H}(g_n))) \lor (g_1 \lor g_2 \lor \ldots \lor g_n).
\]

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

**Robot-Coffee.** A robot has to deliver coffee to different offices and can move between adjacent offices; coffee can be prepared in a kitchen. We considered the 10 problems from C17 and generated further larger instances with the following types of temporally extended goals:

\[
\begin{align*}
F(C_{o_1}) & \land \ldots \land F(C_{o_n}) & (1) \\
F(F(C_{o_1}) \land \ldots \land F(C_{o_n})) & (2) \\
F(C_{o_1}) & \land \ldots \land F(C_{o_n}) \land G(R_{o_j} \Rightarrow X(\neg R_{o_j})) \land \ldots \\
X(X(\ldots X(R_{k\text{kitchen}}))) & (3) \\
\end{align*}
\]

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

### 6.2 Experimental Results

Table 3 reports the overall results for all systems. The last three rows for C17, while BF23 is indicated by “domain-formula” (e.g., $\text{BLOCKS}$-I refers to blocksworld with goals of type (1)). $P4P^{\text{PPL}}$ achieves the highest coverage, followed by $\text{ltf2f}$ and $P4P^{\text{PPL}}$. C17 instances are small and trivially solved by all systems within seconds. In contrast, BF23 instances are larger and designed to challenge the
Table 3. Coverage, average Run-Time (Avg RT), and Policy size (Avg |π|) achieved by P4P, P4PPal, and ltlf2f. Averages are computed among instances solved by all systems obtaining at least 25% of the total coverage compared to the best performer. The policy size is reported without counting spurious actions added by compilations. Column I is the number of instances in a domain. “–” indicates a system excluded by the comparison. In bold are the best performers.

<table>
<thead>
<tr>
<th>Domain</th>
<th>I</th>
<th>Coverage</th>
<th>Avg RT</th>
<th>Avg</th>
<th>Policy size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>P4P</td>
<td>P4PPal</td>
<td>ltlf2f</td>
<td>P4P</td>
</tr>
<tr>
<td>BLOCKS-1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>69.56</td>
</tr>
<tr>
<td>BLOCKS-2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>27.11</td>
</tr>
<tr>
<td>BLOCKS-3</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>85.48</td>
</tr>
<tr>
<td>BLOCKS-4</td>
<td>10</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>30.04</td>
</tr>
<tr>
<td>BLOCKS-seq</td>
<td>30</td>
<td>30</td>
<td>8</td>
<td>20</td>
<td>2.16</td>
</tr>
<tr>
<td>COFFEE-1</td>
<td>15</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>5.14</td>
</tr>
<tr>
<td>COFFEE-2</td>
<td>15</td>
<td>11</td>
<td>5</td>
<td>2</td>
<td>14.96</td>
</tr>
<tr>
<td>COFFEE-3</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>COFFEE-4</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15.82</td>
</tr>
<tr>
<td>ROVERS-1</td>
<td>40</td>
<td>24</td>
<td>3</td>
<td>10</td>
<td>8.99</td>
</tr>
<tr>
<td>ROVERS-2</td>
<td>40</td>
<td>23</td>
<td>4</td>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>ROVERS-3</td>
<td>40</td>
<td>37</td>
<td>20</td>
<td>40</td>
<td>27.37</td>
</tr>
<tr>
<td>ROVERS-4</td>
<td>40</td>
<td>23</td>
<td>16</td>
<td>7</td>
<td>32.97</td>
</tr>
<tr>
<td>BLOCKS</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>2.66</td>
</tr>
<tr>
<td>COFFEE</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>2.46</td>
</tr>
<tr>
<td>ROVERS</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>3.42</td>
</tr>
<tr>
<td>Total</td>
<td>252</td>
<td>159</td>
<td>166</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Survival plot in linear (lhs) and logarithmic (rhs) scale.

Figure 2. Run-Time comparison of ltlf2f vs P4P and P4PPal.

The general theoretical and practical advantages of PPLTL observed so far may definitely pave the way for more effective solution techniques in handling control knowledge in planning. Future work concerns developing FOND planners that can natively handle PPLTL goals and exploring PPLTL-aware heuristics.

7 Conclusions

We study FOND planning for goals expressed in ppltl. We prove that the effectiveness of ppltl in expressing temporal goals in deterministic planning seamlessly extends to nondeterministic planning. We prove that FOND planning for ppltl goals can be polynomially encoded into FOND planning for reachability goals. We present two such encodings, which are able to solve a broader range of problems compared to the state-of-the-art encoding supporting temporally extended goals expressed in LTLf. The general theoretical and practical advantages of ppltl observed so far may definitely pave the way for more effective solution techniques in handling control knowledge in planning. Future work concerns developing FOND planners that can natively handle ppltl goals and exploring ppltl-aware heuristics.
Acknowledgements

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