*ECAI 2023*
*K. Gal et al. (Eds.)*
*© 2023 The Authors.*
*This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).*
*doi:10.3233/FAIA230281*

# FOND Planning for Pure-Past Linear Temporal Logic Goals

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

[a]University of Brescia, Italy
[b]University of Oxford, UK
[c]Sapienza University, Italy
[d]Bank of Italy
[e]York University, Canada
[f]IBM Research, USA

**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 $\text{LTL}_f$, is usually more appropriate [8, 9, 21, 22].

Although seminal works only focused on planning with deterministic actions and LTL or $\text{LTL}_f$ goals as, e.g., in [8, 9, 21, 38], more recently, there has been a surge of interest in the use of $\text{LTL}_f$ 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 $\text{LTL}_f$ goals is EXPTIME-complete in the domain as for standard reachability goals (i.e., final-state goals) [35] and 2EXPTIME-complete in the $\text{LTL}_f$ goal formula [19]. Specifically, in nondeterministic domains, since $\text{LTL}_f$ goals can specify non-Markovian properties [25], the added expressiveness of $\text{LTL}_f$ goals worsens the worst-case goal complexity to 2EXPTIME-complete, compared to the EXPTIME-complete of reachability goals. That is because, in FOND planning, $\text{LTL}_f$ 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 $\text{LTL}_f$ 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 $\text{LTL}_f$ 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-

* Views and opinions expressed are of the author's own and are not representative of the Bank of Italy's official position.
** Corresponding Author. Email: francesco.fuggitti@gmail.com

ous works primarily focused on MDPs, this paper addresses the problem of FOND planning, where PPLTL 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 PPLTL 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 $\mathcal{D} = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, eff \rangle$ where $\mathcal{F}$ is a set of positive literals, $\mathcal{F}_{der}$ is a set of derived predicates, $\mathcal{X}$ is a set of axioms, $A$ is a set of action labels, $pre$ and $eff$ are two functions that define the preconditions and effects of each action $a \in A$. A planning state $s$ is a subset of $\mathcal{F}$, and a positive literal $f$ holds true in $s$ if $f \in s$; otherwise, $f$ is false in $s$. Axioms have the form $d \leftarrow \psi$ where $d \in \mathcal{F}_{der}$ and $\psi$ is a formula over $\mathcal{F} \cup \mathcal{F}_{der}$. An axiom $d \leftarrow \psi$ specifies that $d$ is derived to be true from a state $s$ if and only if we can prove that $s \models \psi$, possibly using other axioms from $\mathcal{X}$. We assume 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 $\mathcal{F} \cup \mathcal{F}_{der}$ 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 $\mathcal{F} \cup \mathcal{F}_{der}$ and a set $\{\mathsf{eff}_1, \dots, \mathsf{eff}_n\}$ of effects, respectively. Each effect $\mathsf{eff}_i \in eff(a)$ is a set of conditional effects each of the form $c \triangleright e$, where $c$ is a propositional formula over $\mathcal{F} \cup \mathcal{F}_{der}$ and $e \subseteq \mathcal{F} \cup \{\neg f \mid f \in \mathcal{F}\}$ is a set of literals from $\mathcal{F}$.

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 $c \triangleright e$ 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 $\mathsf{eff}_i \in eff(a)$ 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 $c \triangleright e \in \mathsf{eff}_i$ triggered in $s$ deletes it ($\neg f \in e$) or (*ii*) there is a conditional effect $c \triangleright e \in \mathsf{eff}_i$ 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 $\{s'_1, \dots, s'_n\}$ 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 $\Gamma = \langle \mathcal{D}, s_0, G \rangle$, where $\mathcal{D}$ is a domain model, $s_0 \subseteq \mathcal{F}$ is the initial state, and $G$ is a formula over $\mathcal{F} \cup \mathcal{F}_{der}$, also called the reachability goal.

Solutions to $\Gamma$ are *strategies* (aka *policies*). A strategy is defined as a partial function $\pi : (2^{\mathcal{F}})^* \to A$ 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 $\tau = s_0, s_1, \dots$, where $s_0$ is the initial state, $s_{i+1} \in tr(s_i, a_i)$, and $a_i = \pi(s_0, \dots, s_i)$ for $i \geq 0$. If for a certain sequence of states $\tau = s_0, \dots, s_n$ 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 (PPLTL), which we review in the next section.

**Definition 1.** *A FOND planning problem for temporally extended goals is a tuple $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$, where $\mathcal{D}$ 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 $\pi : (2^{\mathcal{F}})^+ \to A$ 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

PPLTL is the variant of $\text{LTL}_f$ that refers to the past only. A survey on PPLTL can be found in [18], where it is denoted as $\text{PLTL}_f$. Given a set $\mathcal{P}$ of propositions, PPLTL formulas are defined as:

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

where $p \in \mathcal{P}$, $\mathsf{Y}$ is the *yesterday* operator and $\mathsf{S}$ 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 $\mathsf{WY}\varphi \equiv \neg\mathsf{Y}\neg\varphi$, 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*, $\tau = s_0, \dots, s_n$ where $s_i$ at instant $i$ is a propositional interpretation over the alphabet $2^{\mathcal{P}}$. We denote by $\mathsf{length}(\tau)$ the length $n{+}1$ of $\tau$, and by $\mathsf{last}(\tau)$ the last element of the trace. Given a trace $\tau = s_0, \dots, s_n$, we denote by $\tau_{i,j}$, with $0 \leq i \leq j \leq n$, the sub-trace $s_i, \dots, s_j$ 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 $\mathsf{length}(\tau) \geq 1$ 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 $\varphi$ is *true* in $\tau$, denoted $\tau \models \varphi$, if $\tau, \mathsf{length}(\tau)-1 \models \varphi$. We denote by $\mathsf{sub}(\varphi)$ the set of all subformulas of $\varphi$ [21]. For instance, if $\varphi = a \wedge \neg \mathsf{Y}(b \vee c)$, where $a, b, c$ are atomic, then $\mathsf{sub}(\varphi) = \{a, b, c, (b \vee c), \neg \mathsf{Y}(b \vee c), \mathsf{Y}(b \vee c), a \wedge \neg \mathsf{Y}(b \vee c)\}$. Subformulas correspond to the nodes of the syntactic tree of the formula, so they are linearly many. Also, $|\mathsf{sub}(\varphi)|$ defines the size of a PPLTL formula $\varphi$.

**Theorem 1** ([18]). *PPLTL 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 PPLTL, presented in [18, 11], that we exploit later in the paper. While PPLTL is as expressive as LTL$_f$, it gives an exponential worst-case theoretical computational advantage in several applications. In fact, both LTL$_f$ and PPLTL 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 PPLTL 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 PPLTL, as the best-known algorithms to translate LTL$_f$ into PPLTL (and vice-versa) are 3EXPTIME [18], and no automated tool exists. Nevertheless, many interesting properties expressed in LTL$_f$, including the well-known DECLARE templates [39] and PDDL3 operators [28], are actually *polynomially* related to their translation into PPLTL (and vice versa) [11, 27].

Given these observations, we can implicitly and incrementally build a DFA for the PPLTL 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 PPLTL, 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, PPLTL formulas can be decomposed into present and past components, given the fixpoint characterization of the *since* operator: $\phi_1 \mathsf{S} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \mathsf{Y}(\phi_1 \mathsf{S} \phi_2))$. Given this equivalence, the formula decomposition can be computed by recursively applying the following transformation function $\mathsf{pnf}(\cdot)$:

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

A formula resulting from the application of $\mathsf{pnf}(\cdot)$ 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 $\mathsf{Y}$ 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 proposition holds:

**Proposition 1.** *Every PPLTL formula $\varphi$ 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 $\varphi$.*

Interestingly, the PNF decomposition allows evaluating PPLTL formulas by only knowing the truth value of some of its subformulas, those that are within the $\mathsf{Y}$-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*) "$\mathsf{Y}\phi$" for each subformula of $\varphi$ of the form $\mathsf{Y}\phi$, and (*ii*) "$\mathsf{Y}(\phi_1 \mathsf{S} \phi_2)$" for each subformula of $\varphi$ of the form $\phi_1 \mathsf{S} \phi_2$. For instance, for the formula $a \mathsf{S} \mathsf{Y} b$, we only have $\Sigma_\varphi = \{$"$\mathsf{Y}(a \mathsf{S} \mathsf{Y} b)$", "$\mathsf{Y}b$"$\}$. To evaluate the truth value of propositions in $\Sigma_\varphi$, we use the interpretation $\sigma : \Sigma_\varphi \to \{\top, \bot\}$, 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 PPLTL 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 PPLTL subformula in $\mathsf{sub}(\varphi)$, the predicate $\mathsf{val}(\phi, \sigma_i, s_i)$ is recursively defined as:*

- $\mathsf{val}(p, \sigma_i, s_i)$ *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* $\neg\mathsf{val}(\phi', \sigma_i, s_i)$.

The $\mathsf{val}(\phi, \sigma_i, s_i)$ predicate determines what is the truth value of any PPLTL formula $\phi \in \mathsf{sub}(\varphi)$ by reading a propositional interpretation $s_i$ from trace $\tau$ 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 $\tau^{[\varphi]} = \sigma_0, \ldots, \sigma_n$ over $\Sigma_\varphi$, where $\sigma_0($"$\mathsf{Y}\phi$"$) \doteq \bot$ for each "$\mathsf{Y}\phi$" $\in \Sigma_\varphi$ and $\sigma_i($"$\mathsf{Y}\phi$"$) \doteq \mathsf{val}(\phi, \sigma_{i-1}, s_{i-1})$, for all $i$ with $0 < i \leq n$. The following result holds.

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

Theorem 2 states that a PPLTL formula can be evaluated on the last instant of a trace $\tau$ by computing the $\mathsf{val}(\cdot)$ 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 PPLTL 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 PPLTL Goals

We devise a particularly effective technique for FOND planning for PPLTL goals by exploiting the result in Theorem 2. Given that PPLTL 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 PPLTL 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 PPLTL 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 $\Gamma = \langle \mathcal{D}, s_0, \varphi \rangle$, where $\mathcal{D} = \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff} \rangle$ is a nondeterministic domain, $s_0$ the initial state and $\varphi$ a PPLTL goal, the encoded FOND planning problem is $\Gamma' = \langle \mathcal{D}', s'_0, G' \rangle$, where $\mathcal{D}' = \langle \mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, \mathit{eff}' \rangle$ is the encoded FOND planning domain, $s'_0$ is the new initial state and $G'$ is

**Table 1.** Components of the encoded FOND planning problem $\Gamma' = \langle\langle\mathcal{F}', \mathcal{F}'_{der}, \mathcal{X}', A, pre, \mathit{eff}'\rangle, s'_0, G'\rangle$ using axioms for a given FOND planning problem $\Gamma = \langle\langle\mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff}\rangle, s_0, \varphi\rangle$.

| Components | Encoding |
|---|---|
| Fluents $\mathcal{F}'$ | $\mathcal{F}' := \mathcal{F} \cup \Sigma_\varphi$ |
| Derived Predicates $\mathcal{F}'_{der}$ | $\mathcal{F}'_{der} := \mathcal{F}_{der} \cup \{\mathsf{val}_\phi \mid \phi \in \mathsf{sub}(\varphi)\}$ |
| Axioms $\mathcal{X}'$ | $\mathcal{X}' := \mathcal{X} \cup \{x_\phi \mid \phi \in \mathsf{sub}(\varphi)\}$ where $x_\phi$ is $\begin{cases} \mathsf{val}_p \leftarrow p & (\phi = p) \\ \mathsf{val}_{\mathsf{Y}\phi'} \leftarrow \text{“}\mathsf{Y}\phi'\text{”} & (\phi = \mathsf{Y}\phi') \\ \mathsf{val}_{\phi_1 \mathsf{S} \phi_2} \leftarrow (\mathsf{val}_{\phi_2} \vee (\mathsf{val}_{\phi_1} \wedge \text{“}\mathsf{Y}(\phi_1 \mathsf{S} \phi_2)\text{”})) & (\phi = \phi_1 \mathsf{S} \phi_2) \\ \mathsf{val}_{\phi_1 \wedge \phi_2} \leftarrow (\mathsf{val}_{\phi_1} \wedge \mathsf{val}_{\phi_2}) & (\phi = \phi_1 \wedge \phi_2) \\ \mathsf{val}_{\neg\phi'} \leftarrow \neg\mathsf{val}_{\phi'} & (\phi = \neg\phi') \end{cases}$ |
| Action Labels $A$ | $A := A$, i.e., unchanged |
| Preconditions $pre$ | $pre(a) := pre(a)$ for every $a \in A$, i.e., unchanged |
| Effects $\mathit{eff}'$ | $\mathit{eff}'(a) := \{\mathsf{eff}_i \cup \mathsf{eff}_{\mathsf{val}} \mid \mathsf{eff}_i \in \mathit{eff}(a)\}$, where $\mathsf{eff}_{\mathsf{val}} = \{\mathsf{val}_\phi \rhd \{\text{“}\mathsf{Y}\phi\text{”}\}, \neg\mathsf{val}_\phi \rhd \{\neg\text{“}\mathsf{Y}\phi\text{”}\} \mid \text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi\}$ |
| Initial State $s'_0$ | $s'_0 := \sigma_0 \cup s_0$ |
| Goal $G'$ | $G' := \mathsf{val}_\varphi$ |

the new reachability goal. The formal construction of $\Gamma'$ is reported in Table 1.

In this encoding, we employ axioms to determine which subformula $\phi$ of the goal $\varphi$ is true in a planning state $s$. In particular, the new FOND domain includes an axiom $\mathsf{val}_\phi \leftarrow \psi$ for every subformula $\phi \in \mathsf{sub}(\varphi)$. Given a sequence of states $(\sigma_0, s_0), \ldots, (\sigma_n, s_n)$, these axioms mimic rules in Definition 2 and are intended to be such that the current state $(\sigma_i, s_i) \models \mathsf{val}_\phi$ iff $\mathsf{val}(\phi, \sigma_i, s_i)$ (without loss of generality, in this section, we assume that $\sigma$ and $s$ represent set of positive literals, and we use $(\sigma_i, s_i)$ to denote the state $\sigma_i \cup s_i$). Axioms not only elegantly model the mathematics behind Theorem 2 (i.e., the $\mathsf{val}(\phi, \sigma_i, s_i)$), but also simplify the action schema and goal descriptions without adding control predicates among fluents.

From Table 1, it is also easy to see that no new action is added to the encoded FOND domain and that the precondition function $pre$ remains unchanged. In fact, every domain's action $a \in A$ is only modified on its effects $\mathit{eff}(a)$ by adding a way to update the assignments of propositions in $\Sigma_\varphi$. These additional effects are exactly the same for every action in $A$. Moreover, since $\sigma_i$ maintains values of "$\mathsf{Y}\phi$" in $\Sigma_\varphi$, 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 $\sigma$ without knowing neither which action has been executed nor which effect such action has had on the original fluents. Observe that the auxiliary part $\mathsf{eff}_{\mathsf{val}}$ in $\mathit{eff}'(a)$ *deterministically* updates subformulas values in $\Sigma_\varphi$, without affecting any fluent $f \in \mathcal{F}$ of the original domain model. This is crucial for the encoding's correctness.

It is easy to see that our encoding is polynomially related to the original problem and the satisfaction of the PPLTL goal $\varphi$ on the trace $\tau'$ corresponds to the satisfaction of $\mathsf{val}_\varphi$ in the last instant of $\tau'$.

**Theorem 3.** *Let $\Gamma$ be a FOND planning problem. The size of $\Gamma'$ obtained following the encoding of Table 1 is polynomial in the size of $\Gamma$. In particular, it is linear in the size of the domain specification and linear in the size of the goal.*

**Theorem 4.** *Let $\varphi$ be a* PPLTL *formula over $\mathcal{P}$, $\phi \in \mathsf{sub}(\varphi)$ a subformula of $\varphi$, $\tau$ a trace over $\mathcal{P}$, and $\tau^{[\varphi]}$ the corresponding trace over $\Sigma_\varphi$. Then,* $\mathsf{val}(\phi, \sigma_n, s_n)$ *iff* $(\sigma_n, s_n) \models \mathsf{val}_\phi$.

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

Any trace $\tau' = s'_0, \ldots, s'_n$ on $\mathcal{D}'$ can be seen as $\tau' = zip(\tau^{[\varphi]}, \tau)$, where $\tau = s_0, \ldots, s_n \in (2^{\mathcal{F}})^+$, $\tau^{[\varphi]} = \sigma_0, \ldots, \sigma_n \in (2^{\Sigma_\varphi})^+$, where each element of $\tau'$ 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 $\tau$. 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 $\mathcal{D}'$.

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

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

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

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

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

**Lemma 2.** *For every $\pi' : (2^{\mathcal{F}'})^+ \to A$ that is a (strong or strong-cyclic) winning strategy for the encoded planning problem $\Gamma'$, the corresponding $\pi : (2^{\mathcal{F}})^+ \to A$ 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 iff $\Gamma'$ 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 $\Pi'(s') = a$ or $\Pi'(s')$ undefined at the goal. These can be immediately transformed in trace-based strategies by defining:

$$\begin{array}{lll} \pi(\tau') = a & \text{iff} & \Pi'(\mathsf{last}(\tau')) = a \\ \pi(\tau') \text{ is undefined} & \text{iff} & \Pi'(\mathsf{last}(\tau')) \text{ is undefined.} \end{array}$$

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, $\sigma_{i+1}(\text{“}\mathsf{Y}\phi\text{”}) = \mathsf{val}(\phi, \sigma_i, s_i)$ (for each “$\mathsf{Y}\phi$” $\in \Sigma_\varphi$) 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 “$\mathsf{Y}\phi$” for each temporal component of $\varphi$. However, instead of using the $\mathsf{val}(\cdot)$ predicates, 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.* $\mathsf{ppnf}(\varphi)$ *is a propositional formula obtained by substituting every* $\mathsf{Y}(\phi)$ *with* “$\mathsf{Y}(\phi)$” *in* $\mathsf{pnf}(\phi)$.

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

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

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

**Table 2.** Components of the encoded FOND planning problem $\Gamma'' = \langle \langle \mathcal{F}'', \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff}'' \rangle, s''_0, G'' \rangle$ without additional derived predicates for a given FOND planning problem $\Gamma = \langle \langle \mathcal{F}, \mathcal{F}_{der}, \mathcal{X}, A, pre, \mathit{eff} \rangle, s_0, \varphi \rangle$.

| Components | Encoding |
|---|---|
| Fluents $\mathcal{F}''$ | $\mathcal{F}'' := \mathcal{F} \cup \Sigma_\varphi$ |
| Derived Predicates $\mathcal{F}_{der}$ | $\mathcal{F}_{der} := \mathcal{F}_{der}$, i.e., unchanged |
| Axioms $\mathcal{X}$ | $\mathcal{X} := \mathcal{X}$, i.e., unchanged |
| Action Labels $A$ | $A := A$, i.e., unchanged |
| Preconditions $pre$ | $pre(a) := pre(a)$ for every $a \in A$, i.e., unchanged |
| Effects $\mathit{eff}''$ | $\mathit{eff}''(a) := \{\mathsf{eff}_i \cup \mathsf{eff}_{\mathsf{ppnf}} \mid \mathsf{eff}_i \in \mathit{eff}(a)\}$, where $\mathsf{eff}_{\mathsf{ppnf}} = \{\mathsf{ppnf}(\phi) \rhd \{\text{“}\mathsf{Y}\phi\text{”}\}, \neg\mathsf{ppnf}(\phi) \rhd \{\neg\text{“}\mathsf{Y}\phi\text{”}\} \mid \text{“}\mathsf{Y}\phi\text{”} \in \Sigma_\varphi\}$ |
| Initial State $s''_0$ | $s''_0 := \sigma_0 \cup s_0$ |
| Goal $G''$ | $G'' := \mathsf{ppnf}(\varphi)$ |

**Theorem 6** (Correctness)**.** *Let $\Gamma$ be a FOND planning problem and let $\Gamma''$ be the problem encoded following Table 2. Then $\Gamma$ has a winning strategy $\pi$ iff so does $\Gamma''$.*

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$_\mathcal{X}$ 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 $\mathsf{eff}_{\mathsf{ppnf}}$ 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$_\mathcal{X}$ requires conjunctive conditional effects and axioms, we use P4P with PRP (P4P$^{\mathsf{PRP}}$ for short), and Paladinus with P4P$_\mathcal{X}$ (P4P$^{\mathsf{Pal}}_\mathcal{X}$ 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$_\mathcal{X}$ 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 (compilation 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 $\text{LTL}_f$, and therefore, for comparison reasons, we manually translated them to PPLTL. Recall that although $\text{LTL}_f$ and PPLTL have the same expressiveness, no tool to translate one into the other exists yet. Therefore, for each $\text{LTL}_f$ 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 $\text{LTL}_f$.

**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]:

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

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

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

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

where atoms $a$ and $b_i$ represent whether a block is on the table or on top of another block. Formulas (1) and (4) require each atom $b_i$ to eventually be true in some state, while (2) expresses that $a$ eventually becomes true. In all three formulas, the "Until" ($\mathsf{U}$) 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 $(a\,\mathsf{S}\,\mathsf{O}(b_1))\wedge(a\,\mathsf{S}\,\mathsf{O}(b_2))\wedge\ldots\wedge(a\,\mathsf{S}\,\mathsf{O}(b_n))$. (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 $\mathsf{O}(b_1\wedge\mathsf{O}(a))\wedge\mathsf{O}(b_2\wedge\mathsf{O}(a))\wedge\ldots\wedge\mathsf{O}(b_n\wedge\mathsf{O}(a))$. Each temporal goal is used over 10 instances obtained by adding more blocks 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., $seq_{i,j}$ where $i$ is the number of blocks, and $j$ is the number of towers to build in a specific order. For example, formula $seq_{3,2}$ requires building two towers made of three blocks. In the end, such blocks must be on the table (*t*). In $\text{LTL}_f$, this formula is expressed as:

$$\mathsf{F}(on_{a,c}\wedge on_{c,b}\wedge \mathsf{XF}(on_{c,b}\wedge on_{b,a}\wedge \mathsf{XF}(on_{a,t}\wedge on_{b,t}\wedge on_{c,t}))),$$

whereas in PPLTL it is expressed as:

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

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

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

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

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

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

where every $g_i$ is an atom representing the completion of a task, i.e., the communication of data about soil, rock, or image. The first two formulas require that each task is accomplished in some state of the 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 $\mathsf{O}(\mathsf{O}(g_1)\wedge\mathsf{O}(g_2)\wedge\ldots\wedge\mathsf{O}(g_n))$. The last formula requires one of the goals to be achieved in the last state; this could be represented in PPLTL without using any temporal operator. However, for the sake of a fair comparison, we translated such $\text{LTL}_f$ formula into a PPLTL formula with the same number of nested temporal operators, i.e.,

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

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

**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:

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

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

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

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

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 $\mathsf{H}(\mathsf{Y}(R_{o_i})\Rightarrow\neg(R_{o_j}))\wedge\neg R_{o_i}$. The last temporally extended goal requires the robot to be in the kitchen in the $k$+1-th state, where $k$ is equal to the number of nested $\mathsf{X}$ operators. In PPLTL, this can be captured by $\mathsf{O}(R_{kitchen}\wedge\mathsf{Y}(\mathsf{Y}(\ldots\mathsf{Y}(start))))$, where the number of $\mathsf{Y}$ 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 are for C17, while BF23 is indicated by "domain-formula" (e.g., BLOCKS-1 refers to blocksworld with goals of type (1)). $\mathsf{P4P}^{\mathsf{PRP}}$ achieves the highest coverage, followed by ltlf2f and $\mathsf{P4P}^{\mathsf{Pal}}_{\mathcal{X}}$. 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 $|\pi|$) achieved by P4P, P4P$_{\mathcal{X}}$, 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.

| Domain | I | Coverage | | | Avg RT | | | Avg $\|\pi\|$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | P4P$^{\mathsf{PRP}}$ | P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ | ltlf2f | P4P$^{\mathsf{PRP}}$ | P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ | ltlf2f | P4P$^{\mathsf{PRP}}$ | P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ | ltlf2f |
| BLOCKS-1 | 10 | **10** | **10** | 1 | 69.56 | **14.98** | – | 4.00 | **3.00** | – |
| BLOCKS-2 | 10 | **10** | **10** | **10** | 27.11 | 10.99 | **2.72** | 2.00 | **1.00** | 2.00 |
| BLOCKS-3 | 10 | 6 | **10** | 2 | 85.48 | **5.99** | – | 5.00 | **2.00** | – |
| BLOCKS-4 | 10 | 6 | **10** | 4 | 30.04 | **11.47** | 19.56 | **3.00** | 3.00 | **3.00** |
| BLOCKS-seq | 30 | **30** | 8 | 20 | **2.16** | 32.25 | 2.29 | 14.12 | 14.00 | **13.75** |
| COFFEE-1 | 15 | **12** | 5 | 4 | **5.14** | 53.99 | 53.34 | 56.00 | **50.50** | 56.00 |
| COFFEE-2 | 15 | **11** | 5 | 2 | **14.96** | 361.63 | – | 60.00 | **54.00** | – |
| COFFEE-3 | 15 | **2** | 0 | 1 | – | – | – | – | – | – |
| COFFEE-4 | 15 | **15** | **15** | **15** | 15.82 | 2.59 | **2.04** | 15.00 | **14.47** | 15.00 |
| ROVERS-1 | 40 | **24** | 3 | 10 | **8.99** | – | 23.47 | **28.40** | – | 30.50 |
| ROVERS-2 | 40 | **23** | 4 | 7 | – | – | – | – | – | – |
| ROVERS-3 | 40 | 37 | 20 | **40** | 27.37 | 258.75 | **2.81** | 6.05 | 5.90 | **5.30** |
| ROVERS-4 | 40 | **23** | 16 | 7 | **32.97** | 52.34 | – | 5.62 | **5.06** | – |
| BLOCKS | 24 | **24** | **24** | **24** | 2.66 | 1.85 | **1.71** | 7.71 | **5.21** | 6.75 |
| COFFEE | 10 | **10** | **10** | **10** | 2.46 | 10.03 | **1.97** | 14.22 | **12.33** | 13.44 |
| ROVERS | 9 | **9** | **9** | **9** | 3.42 | 7.41 | **1.85** | **9.11** | 12.56 | **9.11** |
| **Total** | | **252** | 159 | 166 | | | | | | |

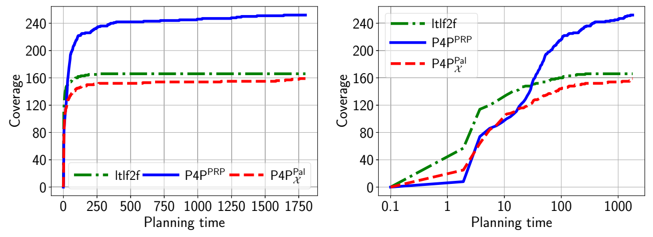

**Figure 1.** Survival plot in linear (lhs) and logarithmic (rhs) scale.

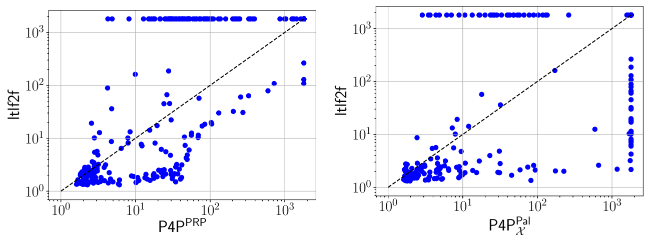

**Figure 2.** Run-Time comparison of ltlf2f vs P4P$^{\mathsf{PRP}}$ and P4P$^{\mathsf{Pal}}_{\mathcal{X}}$.

compilations. Indeed, we observe that P4P$^{\mathsf{PRP}}$ solves 80.2% more instances than P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ and 69.9% more instances than ltlf2f, yet P4P$^{\mathsf{PRP}}$ does not dominate the other compilations. P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ is effective in handling BLOCKS-3 and BLOCKS-4, while P4P$^{\mathsf{PRP}}$ times out. Instead, P4P$^{\mathsf{PRP}}$ works better with goals of type "seq". ltlf2f is extremely effective at handling ROVERS-3. Here, temporal goals can be represented by a very compact automaton, enabling ltlf2f to quickly solve all problems. This advantage can be exploited by neither P4P nor P4P$_{\mathcal{X}}$; such compilations work on the syntactic structure of formulas and are unable to exploit the semantics of temporal goals. Instead, in rovers and robot-coffee, P4P$^{\mathsf{PRP}}$ is more effective than ltlf2f when temporal goals are of type (1) and (2). The automaton blows up when such instances become larger, and ltlf2f either fails at compilation time or produces problems too complex for PRP to handle. In ROVERS-4, P4P$^{\mathsf{PRP}}$ and P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ perform well, while we observed that in many instances ltlf2f crashes during the automaton computation. COFFEE-4 is easily handled by all systems, while COFFEE-3 proved to be challenging for every compilation. Finally, all systems computed policies of comparable dimensions.

On average, ltlf2f has the lowest runtime in 6 domains (3 are C17 domains featuring small instances), whereas P4P$^{\mathsf{PRP}}$ and P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ combined have the lowest runtime in 8 domains. To shed some light on this aspect, Figure 1 displays coverage over time for all systems. ltlf2f dominates the other compilations at the start (visible in the logarithmic scale plot on the right). Instead, P4P$^{\mathsf{PRP}}$ solves more instances than ltlf2f after 29.7 seconds. In an instance-by-instance comparison (Figure 2, left), we observe that ltlf2f solves many problems (130) before P4P$^{\mathsf{PRP}}$ does. P4P$^{\mathsf{PRP}}$ introduces complex formulas in conditional effects and goals, and we observed that the preprocessing of PRP often exceeds by orders of magnitude the compilation time. Overcoming this issue without introducing axioms is an open question for future work. The runtime pairwise comparison of ltlf2f with P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ (Figure 2, right), shows that ltlf2f is faster than P4P$^{\mathsf{Pal}}_{\mathcal{X}}$ in most instances, and this behavior can be attributed to Paladinus being slower than PRP for those instances.

## 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 LTL$_f$. 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

[1] Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin, 'Stochastic Fairness and Language-Theoretic Fairness in Planning in Nondeterministic Domains', in *ICAPS*, pp. 20–28. AAAI Press, (2020).

[2] Fahiem Bacchus, Craig Boutilier, and Adam Grove, 'Rewarding Behaviors', in *AAAI*, pp. 1160–1167, (1996).

[3] Fahiem Bacchus, Craig Boutilier, and Adam Grove, 'Structured Solution Methods for non-Markovian Decision Processes', in *AAAI*, pp. 112–117, (1997).

[4] Fahiem Bacchus and Froduald Kabanza, 'Planning for Temporally Extended Goals', *Ann. Math. Artif. Intell.*, **22**(1-2), 5–27, (1998).

[5] Fahiem Bacchus and Froduald Kabanza, 'Using Temporal Logics to Express Search Control Knowledge for Planning', *AIJ*, **116**(1-2), 123–191, (2000).

[6] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen, *Principles of Model Checking*, MIT, 2008.

[7] Jorge A. Baier, Christian Fritz, Meghyn Bienvenu, and Sheila A. McIlraith, 'Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners', in *AAAI*, pp. 1509–1512. AAAI, (2008).

[8] Jorge A. Baier and Sheila A. McIlraith, 'Planning with First-Order Temporally Extended Goals using Heuristic Search', in *AAAI*, pp. 788–795. AAAI, (2006).

[9] Jorge A. Baier and Sheila A. McIlraith, 'Planning with Temporally Extended Goals Using Heuristic Search', in *ICAPS*, pp. 342–345. AAAI, (2006).

[10] Howard Barringer, Michael Fisher, Dov M. Gabbay, Graham Gough, and Richard Owens, 'METATEM: A Framework for Programming in Temporal Logic', in *REX Workshop*, volume 430 of *LNCS*, pp. 94–129. Springer, (1989).

[11] Luigi Bonassi, Giuseppe De Giacomo, Marco Favorito, Francesco Fuggitti, Alfonso Emilio Gerevini, and Enrico Scala, 'Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic', in *ICAPS*, volume 33, pp. 61–69, (Jul. 2023).

[12] Alberto Camacho, Meghyn Bienvenu, and Sheila A McIlraith, 'Towards a Unified View of AI Planning and Reactive Synthesis', in *ICAPS*, volume 29, pp. 58–67, (2019).

[13] Alberto Camacho, Eleni Triantafillou, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith, 'Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces', in *AAAI*, pp. 3716–3724. AAAI Press, (2017).

[14] A. Chandra, D. Kozen, and L. Stockmeyer, 'Alternation', *J. of the ACM*, **28**(1), (1981).

[15] Alessandro Cimatti, Fausto Giunchiglia, Enrico Giunchiglia, and Paolo Traverso, 'Planning via Model Checking: A Decision Procedure for *AR*', in *ECP*, volume 1348 of *LNCS*, pp. 130–142. Springer, (1997).

[16] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso, 'Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking', *AIJ*, **147**(1-2), 35–84, (2003).

[17] Alessandro Cimatti, Marco Roveri, and Paolo Traverso, 'Strong Planning in Non-Deterministic Domains Via Model Checking', in *AIPS*, pp. 36–43. AAAI, (1998).

[18] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin, 'Pure-Past Linear Temporal and Dynamic Logic on Finite Traces', in *IJCAI*, pp. 4959–4965. ijcai.org, (2020).

[19] Giuseppe De Giacomo and Sasha Rubin, 'Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals', in *IJCAI*, pp. 4729–4735, (2018).

[20] Giuseppe De Giacomo and Moshe Y. Vardi, 'Automata-Theoretic Approach to Planning for Temporally Extended Goals', in *ECP*, volume 1809 of *LNCS*, pp. 226–238. Springer, (1999).

[21] Giuseppe De Giacomo and Moshe Y. Vardi, 'Linear Temporal Logic and Linear Dynamic Logic on Finite Traces', in *IJCAI*, pp. 854–860. IJCAI/AAAI, (2013).

[22] Giuseppe De Giacomo and Moshe Y. Vardi, 'Synthesis for LTL and LDL on Finite Traces', in *IJCAI*, pp. 1558–1564. AAAI Press, (2015).

[23] E. Allen Emerson, 'Temporal and Modal Logic', in *Handbook of Theoretical Computer Science, Chapter 16*, (1990).

[24] Ramon Fraga Pereira, André Grahl Pereira, Frederico Messa, and Giuseppe De Giacomo, 'Iterative Depth-First Search for FOND Planning', in *ICAPS*, pp. 90–99. AAAI Press, (2022).

[25] Alfredo Gabaldon, 'Non-Markovian Control in the Situation Calculus', *AIJ*, **175**(1), 25–48, (2011).

[26] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi, 'On the Temporal Analysis of Fairness', in *POPL*, pp. 163–173. ACM Press, (1980).

[27] Luca Geatti, Marco Montali, and Andrey Rivkin, 'Reactive Synthesis for DECLARE via Symbolic Automata', *CoRR*, **abs/2212.10875**, (2022).

[28] Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos, 'Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners', *AIJ*, **173**(5-6), 619–668, (2009).

[29] Fausto Giunchiglia and Paolo Traverso, 'Planning as Model Checking', in *ECP*, volume 1809 of *LNCS*, (1999).

[30] Jörg Hoffmann and Stefan Edelkamp, 'The Deterministic Part of IPC-4: An Overview', *JAIR*, **24**, 519–579, (2005).

[31] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck, 'The Glory of the Past', in *Logic of Programs*, volume 193 of *LNCS*, pp. 196–218. Springer, (1985).

[32] Zohar Manna, *Verification of Sequential Programs: Temporal Axiomatization*, 53–102, Springer Netherlands, 1982.

[33] Christian Muise, Sheila McIlraith, and Christopher Beck, 'Improved Non-Deterministic Planning by Exploiting State Relevance', in *ICAPS*, volume 22, pp. 172–180, (2012).

[34] Marco Pistore and Paolo Traverso, 'Planning as Model Checking for Extended Goals in Non-deterministic Domains', in *IJCAI*, pp. 479–486. Morgan Kaufmann, (2001).

[35] Jussi Rintanen, 'Complexity of Planning with Partial Observability', in *ICAPS*, pp. 345–354, (2004).

[36] Gabriele Röger, Florian Pommerening, and Malte Helmert, 'Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting', in *ECAI*, volume 263, pp. 765–770, (2014).

[37] Sylvie Thiébaux, Charles Gretton, John K. Slaney, David Price, and Froduald Kabanza, 'Decision-Theoretic Planning with non-Markovian Rewards', *JAIR*, **25**, 17–74, (2006).

[38] Jorge Torres and Jorge A. Baier, 'Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals', in *IJCAI*, pp. 1696–1703. AAAI Press, (2015).

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