Abstract

LTL\(_f\) and LDL\(_f\) are well-known logics on finite traces. We review PLTL\(_f\) and PLDL\(_f\), their pure-past versions. These are interpreted backward from the end of the trace towards the beginning. Because of this, we can exploit a foundational result on reverse languages to get an exponential improvement, w.r.t LTL\(_f\)/LDL\(_f\), in computing the corresponding DFA. This exponential improvement is reflected in several forms of sequential decision making involving temporal specifications, such as planning and decision problems in non-deterministic and non-Markovian domains. Interestingly, PLTL\(_f\) (resp. PLDL\(_f\)) has the same expressive power as LTL\(_f\) (resp. LDL\(_f\)), but transforming a PLTL\(_f\) (resp. PLDL\(_f\)) formula into its equivalent in LTL\(_f\) (resp. LDL\(_f\)) is quite expensive. Hence, to take advantage of the exponential improvement, properties of interest must be directly expressed in PLTL\(_f\)/PLDL\(_f\).

1 Introduction

Several research areas in AI are attracted by the clarity and ease of the future fragment of Linear-time Temporal Logic (LTL) [Pnueli, 1977]. Specifically, LTL has been employed in reasoning about actions and planning: as a specification mechanism for temporally extended goals [Bacchus and Kabanza, 1998; De Giacomo and Vardi, 1999; Calvanese et al., 2002; Camacho et al., 2017], for constraints on plans [Bacchus and Kabanza, 2000], for expressing preferences and soft constraints [Bienvenu et al., 2006], for specifying multi-agent systems [Fagin et al., 1995], and for specifying norms [Fisher and Wooldridge, 2005].

Recently, a variant of LTL on finite traces, LTL\(_f\), and its extension LDL\(_f\), inspired by the dynamic logic PDL [Harel et al., 2000], has attracted great interest [De Giacomo and Vardi, 2013], and has found application in several contexts, including reactive synthesis [De Giacomo and Vardi, 2015; Camacho et al., 2018], planning [Camacho et al., 2017; De Giacomo and Rubin, 2018], MDPs with non-Markovian rewards [Bacchus et al., 1996; Brafman et al., 2018], reinforcement learning [De Giacomo et al., 2019; Camacho et al., 2019], and non-Markovian planning and decision problems [Brafman and De Giacomo, 2019a; Brafman and De Giacomo, 2019b]. The main reason for this interest is due to the possibility of transforming LTL\(_f\)/LDL\(_f\) formulas into Deterministic Finite-state Automaton (DFA) which then can be suitably employed in different contexts, as mentioned above.

LTL\(_f\)/LDL\(_f\) expresses temporal properties in a “pure-future fashion”, i.e., referring only to the present and the future. However, it has been observed that sometimes specifications are easier and more natural to express referring to the past [Lichtenstein et al., 1985]. For instance, the formula TaskDone \(\land\) (SafeArea\(_s\) \& Clean) requires accomplishing the task as well as having cleaned at some point and only traversed safe areas since then. Notable uses of past temporal logics are for specifying non-Markovian models in reasoning about actions [Gabaldon, 2011], non-Markovian rewards in MDPs [Bacchus et al., 1996], and normative properties for Agents [Fisher and Wooldridge, 2005; Knobouf et al., 2016; Alechina et al., 2018].

In this paper, we review the pure-past versions of LTL\(_f\) and LDL\(_f\), respectively PLTL\(_f\) and PLDL\(_f\). Due to the finite nature of traces, PLTL\(_f\) and PLDL\(_f\) have a very natural interpretation: they specify formulas that must be true at the end of the trace and evaluate the trace backwards. In fact, PLTL\(_f\) has already been introduced in the literature [Maler and Pnueli, 1990; Zhu et al., 2019], but only as a technical means to get results for LTL when past is also considered. Here, instead, we consider PLTL\(_f\), and its extension PLDL\(_f\), as first class citizens.

Working with PLTL\(_f\)/PLDL\(_f\) gives us an exponential (worst-case) computational advantage with respect to LTL\(_f\)/LDL\(_f\). Such an advantage stems from the fact that, as LTL\(_f\)/LDL\(_f\), also PLTL\(_f\)/PLDL\(_f\) can be translated into Alternating Finite-state Automaton (AFA) in polynomial time, but, in the case PLTL\(_f\)/PLDL\(_f\), we can exploit a well-known result on regular languages, which states that an AFA can be transformed, in single exponential time, into a DFA that recognizes the reverse language [Chandra et al., 1981]. This should be contrasted with the fact that the DFA for the language itself (not its reverse) can be double-exponentially larger than the AFA.

This language theoretic property has a deep impact on the conversion of PLTL\(_f\)/PLDL\(_f\) formulas to their corresponding DFAs. Indeed, PLTL\(_f\)/PLDL\(_f\) formulas can be transformed into DFAs in only single exponential time (vs double exponential time for LTL\(_f\)/LDL\(_f\) formulas).

This exponential improvement affects the computational complexity of several well-known problems in all the con-
texts mentioned before that make use of LTL$_f$/LDL$_f$, such as planning in fully observable non-deterministic domains (FOND) [Camacho et al., 2017; De Giacomo and Rubin, 2018; Camacho et al., 2018], non-Markovian reward decision processes [Bacchus et al., 1996; Brafman et al., 2018], reinforcement learning with non-Markovian rewards [De Giacomo et al., 2019; Camacho et al., 2019], and non-Markovian planning domain and decision processes, where the dynamic model of the world is expressed in temporal logic [Brafman and De Giacomo, 2019a; Brafman and De Giacomo, 2019b].

Interestingly, future and past variations of these logics have the same expressive power. However, it is not obvious how to transform a past formula into its equivalent future formula. Indeed, we provide an algorithm that transforms a PLTL$_f$ (resp. PLDL$_f$) into an LTL$_f$ (resp. LDL$_f$) formula whose size is at most triple-exponentially (resp. double-exponentially) larger in the worst-case. These are the best-known bounds.

Hence, it is not computationally sensible to start from LTL$_f$/PLTL$_f$ formulas and transform them into the equivalent PLTL$_f$/PLDL$_f$ formulas to take advantage of the exponential improvement. To get such an advantage, the property of interest should be succinctly expressible in PLTL$_f$/PLTL$_f$, as is often the case when the property naturally talks about the past.

2 Preliminaries

AFA, NFA, and DFA. An alternating finite-state automaton (AFA) is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where (i) $\Sigma = 2^P$ is a finite input alphabet; (ii) $Q$ is a finite set of states; (iii) $q_0 \in Q$ is the initial state; (iv) $F \subseteq 2^Q$ is the set of accepting states; (v) $\delta : Q \times \Sigma \rightarrow B^*(Q)$ is the transition function, where $B^*(Q)$ is the set of positive Boolean formulas over $Q$ (i.e., built from the states using $\land$, $\lor$, and the constants $true$ and $false$).

For instance, $\delta(q_1, a) = q_2 \lor (q_3 \land q_4)$ means that for the automaton to accept the input $ar$, from state $q_1$, it should either accept the input $r$ from $q_2$ or from both $q_3$ and $q_4$. For $V \subseteq Q$ and $\varphi \in B^*(Q)$, we write $V \models \varphi$ if the assignment that maps states in $V$ to $true$ and states in $Q \setminus V$ to $false$ satisfies the formula $\varphi$.

A nondeterministic finite-state automaton (NFA) is an AFA in which no transition uses $\land$. For instance, the transition $\delta(q, a) = (q_1 \lor q_2 \lor q_3)$ is allowed. A deterministic finite-state automaton (DFA) is an NFA in which no transition uses $\lor$. For instance, the transition $\delta(q, a) = q_1$ is allowed.

The size of $A$, denoted $|A|$, is the number of bits required to represent the transition function, i.e., $\sum_{q \in Q} \sum_{a \in \Sigma} |\delta(q, a)|$ which is bounded by $|Q||\Sigma|K$ where $K$ is an upper bound on the lengths of the formulas in the transition function.

An accepting run of an AFA $A$ is defined by introducing the function $\text{Acc} : \Sigma^* \rightarrow 2^Q$, where $q \in \text{Acc}(\tau)$ is read “input $\tau$ is accepted from state $q$”, inductively given as follows:

1. $\text{Acc}(\varepsilon) := F$,
2. $q \in \text{Acc}(ar)$ iff $V \models \delta(q, a)$ for some $V \subseteq \text{Acc}(\tau)$.

A run $\tau$ is accepted by $A$ if $q_0 \in \text{Acc}(\tau)$. Note that in the special case that $\delta(q, a) = true$, we have that $q \in \text{Acc}(ar)$ for all $\tau$ (since $\emptyset \models true$). One way to visualize this definition is via run-trees [1], see [Vardi, 1996].

Given an AFA it is possible to obtain (in single exponential time) an NFA that accepts the same language and whose size is at most single exponential in the size of the AFA [Chandra et al., 1981], and hence a DFA that accepts the same language whose size is at most double exponential in the AFA.

We make use of regular expressions, collectively denoted RE [Hopcroft and Ullman, 1979]. In particular, Kleene’s Theorem says that one can translate DFA to RE in exponential time (and thus to a RE that is at most exponentially-larger than the DFA). We can use RE as temporal specifications on finite traces, see, e.g., [De Giacomo and Vardi, 2013].

LTL$_f$ and LDL$_f$. LTL$_f$ is a variant of Linear-time Temporal Logic (LTL) interpreted over finite, instead of infinite, traces [De Giacomo and Vardi, 2013]. Given a set $P$ of atomic propositions, LTL$_f$ formulas $\varphi$ are as follows:

$$\varphi ::= a | \neg \varphi | \varphi_1 \land \varphi_2 | \bigcirc \varphi | \varphi_1 \bigvee \varphi_2$$

where $a$ denotes an atomic propositions in $P$, $\bigcirc$ is the next operator, and $\bigvee$ is the until operator. Apart from the boolean ones, we use the following abbreviations eventually as $\varphi \bigvee := true \land \varphi$, always as $\square \varphi := \neg \neg \varphi$, weak next as $\varphi := \neg \neg \varphi$ (we will see that on finite traces $\neg \neg \varphi$ is not equivalent to $\neg \neg \varphi$ and last (time point of the trace) last $:= \bullet false$.

Formulas of LTL$_f$ are interpreted over finite traces $\pi$ over the alphabet $2^P$. We write $\pi = \pi_0 \pi_1 \cdots \pi_n$, $\text{lst}(\pi) := n$, and for $i \leq j$ let $\pi_{i,j} := \pi_i \pi_{i+1} \cdots \pi_{j}$ such that $m := \min(j - 1, n + 1)$.

Given $\pi$, we define when an LTL$_f$ formula $\varphi$ holds at position $i$, written $\pi, i \models \varphi$, inductively on $\varphi$, as follows:

$$\begin{align*}
\pi, i \models a & \iff a \in \pi_i \quad (a \in P); \\
\pi, i \models \neg \varphi & \iff \pi, i \not\models \varphi; \\
\pi, i \models \varphi_1 \land \varphi_2 & \iff \pi, i \models \varphi_1 \land \pi, i \models \varphi_2; \\
\pi, i \models \bigcirc \varphi & \iff \pi, i+1 \models \varphi; \\
\pi, i \models \varphi_1 \bigvee \varphi_2 & \iff \pi, i \models \varphi_1 \bigvee \varphi_2 \text{ for some } j \text{ such that } i \leq j \leq \text{lst}(\pi)
\end{align*}$$

Here $\text{lst}(\pi)$ is the last position (i.e., index) in the string $\pi$. We write $\pi \models \varphi$, if $\pi, 0 \models \varphi$ and say that $\varphi$ satisfies or is a model of $\varphi$.

LDL$_f$ is a proper extension of LTL$_f$ that is able to capture regular expressions over traces. Here, following [Brafman et al., 2018], we consider a notational variant of LDL$_f$. Formulas $\varphi$ of LDL$_f$ are defined as follows:

$$\varphi ::= tt | \neg \varphi | \varphi_1 \land \varphi_2 | \langle \emptyset \rangle \varphi$$

where $\phi$ denotes propositional formulas over $P$ (we use the usual abbreviations, e.g., the Boolean constant $true$ is defined as $a \land \neg a$, for some fixed $a \in P$) and $tt$ stands for logical true (not to be confused with the Boolean constant $true$).

all nodes at depth $|\tau|$ are labeled by final states, and ii) if an internal node $x$ is labeled by $q$, and $X$ is the set of labels of the children of $x$, then $X \models \delta(q, x)$. Note that not all branches need to reach depth $|\tau|$ since an internal node at depth $i$ may be labeled $q$ where $\delta(q, \tau_i) = true$. Thus a branch in a run-tree either reaches a final state after reading the word, or hits the transition $true$. An easy induction shows that $q \in \text{Acc}(\tau)$ iff there is a run-tree on input $\tau$ whose root is labeled by $q$. [1]
true). Expressions of the form $\varphi$ are regular expressions (RE)
over propositional formulas $\phi$ and the test construct $\varphi^?$. typi-
cal of Propositional Dynamic Logic (PDL). We abbreviate $[\varphi] := \neg(\varphi)$ as in PDL, $f \doteq \neg t$ for false, and $\phi := (\varphi)tt$ to denote the occurrence of the propositional formula $\phi$. We also use $end \doteq [\text{true}]ff$ to express that the trace has ended.

Intuitively, $[\varphi]\varphi$ states that, from the current step in the
trace, there exists an execution satisfying the RE $\varphi$ such that
its last step satisfies $\varphi$, while $[\varphi]\varphi$ states that, from the current
step, all executions satisfying the RE $\varphi$ are such that their last
step satisfies $\varphi$. Tests are used to insert into the execution path
checks for satisfaction of additional LDLf formulas.

Given a finite trace $\pi$, an LDLf formula $\varphi$ and a non-negative
integer $i$, we define when $\varphi$ holds at $i$, written $\pi, i \Vdash \varphi$, by (mutual) induction, as follows:

1. $\pi, i \Vdash tt$;
2. $\pi, i \Vdash \lnot \varphi$ iff $\pi, i \not\Vdash \varphi$;
3. $\pi, i \Vdash \varphi_1 \land \varphi_2$ iff $\pi, i \Vdash \varphi_1$ and $\pi, i \Vdash \varphi_2$;
4. $\pi, i \Vdash \langle \varphi \rangle \varphi$ iff there is a $j$ such that $i \leq j$ and $\pi, j \Vdash \varphi$.

where the relation $\pi, i \Vdash \varphi$ is inductively defined as:

- $\pi, j \Vdash \varphi$ if $j = i + 1$, $i \leq \text{lst}(\pi)$, and $\pi, i \Vdash \varphi$;
- $\pi, i \Vdash \varphi(?)$ if $j = i$ and $\pi, i \Vdash \varphi$;
- $\pi, i \Vdash \varphi(1\cap 2)$ if $\pi, j \Vdash \varphi(1)$ or $\pi, j \Vdash \varphi(2)$;
- $\pi, i \Vdash \varphi(1; 2)$ if there exists $k \in \{i, j\}$ such that $\pi, k \Vdash \varphi(1)$ and $\pi, k \Vdash \varphi(2)$;
- $\pi, i \Vdash \varphi$ if $i = j$ or there exists $k$ such that $\pi, k \Vdash \varphi(1)$ and $\pi, k \Vdash \varphi(2)$.

Note that if $i > \text{lst}(\pi)$, the above definitions still apply, al-
though $\pi, i \not\Vdash \varphi$.

We say that a trace $\pi$ satisfies an LDLf formula $\varphi$, written
$\pi \Vdash \varphi$, if $\pi, 0 \Vdash \varphi$. Also, sometimes we denote by $\mathcal{R}(\varphi)$
the set of traces that satisfy $\varphi$, i.e., $\mathcal{R}(\varphi) := \{\pi \in (2^P)^* | \pi \Vdash \varphi\}$.

**LTLf/LDLf to AFA and DFA.** For every LTLf/PLTLf formula
$\varphi$ there is an equivalent AFA $A_{\varphi}$ accepting exactly the traces
satisfying $\varphi$, which is linear in the size of $\varphi$ [De Giacomo and
Vardi, 2013]. Moreover, any AFA can be translated into an equiva-
lent DFA, i.e., a DFA recognizing the same language,
whose size is at most double-exponential, which can be
computed in 2EXPTIME in the size of the AFA [Chandra et al.,
1981]. Hence, we have a 2EXPTIME algorithm for transla-
ting a LTLf/PLTLf formula into an equivalent DFA [De Giacomo
and Vardi, 2013]:

**Algorithm 1: Translating LTLf/PLTLf to DFA**

1. Compute an AFA equivalent to $\varphi$ (lin)
2. Compute an NFA equivalent to the AFA (1exp)
3. Determine the NFA obtaining an equivalent DFA (1exp)

**PLTLf and PLDLf**

We study the pure-past version of LTLf, denoted as PLTLf, and
the pure-past version of LTLf, denoted as PLDLf. These are
logics on finite traces that refer only to the past. PLTLf/PLDLf
have a natural interpretation on finite traces: they are satisfied
if they hold in the last position of the trace.

**Example 1.** The property “we are now at location $p_{23}$ and we
have passed through location $p_{12}$” is immediately expressible in PLTLf
as: $p_{23} \land \neg p_{12}$. It is also expressible in LTLf, although with a more
complex formula, i.e., $\Delta(p_{12} \land \Delta(p_{23} \land \text{last}))$.

**Example 2.** Consider the property “every time you took the bus,
you bought a new ticket before”. This can be naturally expressed in
PLTLf as:

$$\Delta(\text{takeB} \Rightarrow \Diamond(\neg \text{takeB} \land \text{buyT}))$$

while the corresponding LTLf formula is:

$$(\text{buyT} \land \text{takeB}) \land \Diamond(\text{takeB} \Rightarrow (\text{buyT} \lor \Diamond(\text{buyT} \land \text{takeB})))$$

We will see later, all PLTLf formulas are translatable into LTLf and
vice versa, however the translation can be quite involved.

**PLDLf.** PLDLf is the generalization of PLTLf by allowing
regular expressions in the eventuality. The syntax of PLDLf
is similar to the syntax of LDLf except that we use a backward
diamond operator. Intuitively, $\langle \varphi \rangle \varphi$ states that there exists
a point in the past, reachable (going backward) through the
regular expression $\varphi$ from the current instant, where $\varphi$ holds.
Formally, the syntax of PLDLf is defined as follows:

$$\varphi := \text{tt} | \varphi \lor \varphi_1 \land \varphi_2 | \langle \varphi \rangle \varphi | \varphi^? | \varphi^*$$

Define the satisfaction relation as for LDLf, except that

- $\pi, i \Vdash \langle \varphi \rangle \varphi$ if there exists $j$, $0 \leq j \leq i$ such that $\pi, j \in \mathcal{R}_{\text{past}}(\varphi)$ and $\pi, j \Vdash \varphi$,
- $\pi, i \Vdash \langle \varphi \rangle \varphi$ if there exists $j$, $0 \leq j \leq i$ such that $\pi, j \in \mathcal{R}_{\text{past}}(\varphi)$ and $\pi, j \Vdash \varphi$.

Define the satisfaction relation as for LDLf, except that

- $\pi, i \Vdash \langle \varphi \rangle \varphi$ if there exists $j$, $0 \leq j \leq i$ such that $\pi, j \in \mathcal{R}_{\text{past}}(\varphi)$ and $\pi, j \Vdash \varphi$.

We say that $\pi$ satisfies a PLDLf formula $\varphi$, written $\pi \Vdash \varphi$,
if $\pi, \text{lst}(\pi) \Vdash \varphi$. As before, we use the abbreviation $[\varphi] \varphi \doteq \neg[\langle \varphi \rangle \varphi] \varphi$. Moreover, we define $\text{start} \doteq [\text{true}]ff$ to express
the fact that the trace has just started.
Example 3. Consider the property “every time, if the cargo-ship departed, then there was an alternation of grab and unload of containers before”. This can be expressed in PLDL₁ as:

\[ \text{[true]} \left( (\text{csDep}) \Rightarrow (\text{unload} \lor \text{grab}) \right) \]

whereas the corresponding LDLᵢ formulas is:

\[ (\text{csDep} \lor \text{unload}) \]

Again, we will see that any PLDL₁ formula can be translated into LDLᵢ (and vice versa), although not always in a simple way. □

One important feature of PLTL₁/PLDL₁, shared with LTL₁/LDL₁, is that one can build a DFA \( \mathcal{A}_ϕ \) accepting the models of \( ϕ \). However, since PLTL₁/PLDL₁ formulas are evaluated from the end of the trace towards the beginning, we can build a DFA whose size is single exponential in the size of the formula (vs. double-exponential, as for LTL₁/LDL₁). The crux of this result lays on the possibility of obtaining an AFA a DFA for the reverse language in single exponential time.

4 Reverse Languages and BCK Corresponding to the Shift from FWD to BCK

5 From PLTL₁/PLDL₁ to DFA

We take advantage of the single exponential reduction of AFA to DFA for the reverse language to get a DFA for PLTL₁/PLDL₁ formulas which is single exponential in the size of the formula. To do so, first, we introduce the syntactic notion of swap, which gives a PLTL₁/PLDL₁ formula produces a LTL₁/LDL₁ formula by syntactically replacing each past operator with its corresponding future operator. Specifically, \( ϕ \) corresponds to \( ψ \), and \( S \) corresponds \( U \), \( \langle \varrho \rangle \) corresponds to \( \langle \varrho \rangle \) and \( \langle X \rangle \) to \( \langle X \rangle \), where \( \varrho \) is the regular expression \( \varrho \) with all formulas in tests replaced by the corresponding swapped formulas. Formally, we define \( ϕ^S \) by induction:

(i) \( a^S := a \) (for all \( a \in \mathcal{A} \)) and \( tt^S = tt \),

(ii) \( (\neg ϕ)^S := \neg ϕ^S \),

(iii) \( (ϕ_1 \lor ϕ_2)^S := ϕ_1^S \lor ϕ_2^S \),

(iv) \( (ϕ_1 \land ϕ_2)^S := ϕ_1^S \land ϕ_2^S \),

(v) \( (ϕ_1 \rightarrow ϕ_2)^S := ϕ_1^S \rightarrow ϕ_2^S \),

(vi) \( \langle ρ \rangle ϕ^S := \langle ϕ \rangle^S \),

(vii) \( (\langle X \rangle ϕ)^S := (ϕ_1)^S \),

(viii) \( (\langle X \rangle ϕ)^S := (ϕ_2)^S \),

(ix) \( (ϕ_1 + ϕ_2)^S := ϕ_1^S + ϕ_2^S \),

(x) \( (ϕ_1 \cdot ϕ_2)^S := ϕ_1^S \cdot ϕ_2^S \),

(xi) \( (ϕ_1^S)^S := ϕ_1^S \).

Similarly, we can swap an LTL₁/LDL₁ formula \( ϕ \) into a PLTL₁/PLDL₁ formula \( ϕ^S \). The following lemma summarizes the relation between formulas and their swaps.

Lemma 1. If \( ϕ \) is an PLTL₁/PLDL₁ (resp. LTL₁/LDL₁) formula, its swap \( ϕ^S \) is an LTL₁/LDL₁ (resp. PLTL₁/PLDL₁) formula, of size \( |ϕ| \), and such that \( \pi = ϕ \iff π^R = ϕ^S \), i.e.,

\[ \text{L}_R(ϕ) = \text{L}(ϕ^S) \]

Now we are ready to show that using directly formulas leads to a computational advantage as they can be reduced to DFA in single exponential time (vs. double exponential time for LTL₁/LDL₁ formulas):

Theorem 2. For every PLTL₁/PLDL₁ formula \( ϕ \) there is an equivalent DFA \( \mathcal{A}_ϕ \) (computable in exponential time) of size at most \( 2^{O(|ϕ|)} \).

Proof. We swap the PLTL₁/PLDL₁ formula \( ϕ \) getting the LTL₁/LDL₁ \( ϕ^S \). Then we construct the DFA \( \mathcal{A}_ϕ^S \). By Lemma 1 and Theorem 1 we get that \( \mathcal{A}_ϕ \) has size \( 2^{O(|ϕ|)} \) and \( \text{L}(ϕ) = \text{L}(ϕ^S) \).

We can define the analogue of Algorithm 1 to translate PLTL₁/PLDL₁ formulas into DFA based on Theorem 2.
Algorithm 2: Translating PLTL$_f$/PLDL$_f$ to DFA
Given a PLTL$_f$/PLDL$_f$ formula $\phi$
1: Translate $\phi$ into the corresponding LTL$_f$/LDL$_f$ $\phi^S$ (lin) 
2: Compute AFA for $\phi^S$ (lin) 
3: Compute DFA from AFA for the reverse language (lexp)

Note that, Algorithm 2 returns the DFA corresponding to a PLTL$_f$/PLDL$_f$ formula in single EXPTIME (worst-case complexity) vs. 2EXPTIME of Algorithm 1 for the LTL$_f$/LDL$_f$ case. This implies that using past temporal formulas reduces the complexity of several problems, as we will see later.

6 PLTL$_f$/PLDL$_f$ and LTL$_f$/LDL$_f$

PLTL$_f$/PLDL$_f$ offers an exponential advantage over LTL$_f$/LDL$_f$ when building the corresponding DFA. However, here we show that they have the same expressive power, and, indeed, they can be translated one into the other. Unfortunately the translations appear to be quite expensive.

Expressive power of PLTL$_f$. We start by establishing that PLTL$_f$ and LTL$_f$ have the same expressive power by using first-order logic (FOL) as an intermediate logic. In this setting, FOL formulas are interpreted on finite traces viewed as labeled linear orders, i.e., formulas can use: variables $x$ that vary over positions and that can be quantified existentially and universally; the binary predicate $<$ denoting the order of positions; equality $=$ between positions; and unary (sometimes called monadic) predicates $P$ for the labels; see, e.g., [De Giacomo and Vardi, 2013] for formal definitions).

We start by observing that LTL$_f$ and PLTL$_f$ can be translated into FOL on finite traces by mimicking the semantics of these logics as FOL formulas, and can be done in linear-time:

Theorem 3. [De Giacomo and Vardi, 2013; Zhu et al., 2019] Both PLTL$_f$ and LTL$_f$ can be translated into FOL on finite traces in linear-time.

For the converse, it is known that FOL (on finite traces) can be translated into LTL$_f$ [Gabbay et al., 1980]. Here, we also show that FOL can also be translated into PLTL$_f$:

Theorem 4 (cf. [Kamp, 1968]). FOL (on finite traces) can be translated into both LTL$_f$ and PLTL$_f$.

Proof. Given an FOL formula $\phi$ replace $x < y$ by $y < x$ to get an FOL $\phi^S$ for the reverse language, i.e., $w \models \phi$ iff $w^R \models \phi^S$. Then, translate the FOL formula $\phi^S$ into an equivalent LTL$_f$ formula $\psi$ [Gabbay et al., 1980]. Then, the PLTL$_f$ formula $\psi^S$ is equivalent to the original FOL formula $\phi$. \qed

Putting these together, we immediately get:

Theorem 5. PLTL$_f$ and LTL$_f$ have the same expressive power.

Considering the results on LTL$_f$ in [De Giacomo and Vardi, 2013] we can now characterize the expressive power of PLTL$_f$.

Theorem 6. PLTL$_f$ has exactly the same expressive power as FOL on finite traces, i.e., star-free regular expressions.

Expressive power of PLDL$_f$. Next, we investigate the expressive power of PLDL$_f$.

Theorem 7. PLDL$_f$ can be translated into RE.

Proof. Apply Theorem 2 to get a DFA, and then apply Kleene’s Theorem to get an equivalent regular expression. \qed

The reverse direction also holds:

Theorem 8. RE can be translated into PLDL$_f$.

Proof. Given a regular expression $\rho$ compute the reverse regular expression $\rho$ and return $\langle \rho \rangle_{\text{start}}$. \qed

Since RE has the same expressive power as Monadic Second-order Logic (MSO) over finite traces (cf., [De Giacomo and Vardi, 2013]), we get the following characterization.

Theorem 9. PLDL$_f$ has the same expressive power as RE, and as MSO on finite traces.

Theorem 10. PLDL$_f$ has the same expressive power as LDL$_f$.

Translating between PLTL$_f$ and LTL$_f$. The above results give us a way to translate LTL$_f$ (resp. PLTL$_f$) into PLTL$_f$ (resp. LTL$_f$): first, translate LTL$_f$ into FOL; then translate FOL into PLTL$_f$. However, we remark that the transformation of an FOL formula into an LTL$_f$ formula is non-elementary (i.e., not bounded by any finite tower of exponentials) in the size of the FOL formula [Gabbay, 1987]. Hence, the above translation of LTL$_f$ (resp. PLTL$_f$) into PLTL$_f$ (resp. LDL$_f$) is non-elementary. In fact, we can do better by making use of the following result from the literature:

Theorem 11. [Maler and Pnueli, 1990] DFA accepting star-free regular languages can be translated into PLTL$_f$ formulas of size at most exponentially larger.

Now, we are ready to provide our translation, which give us the best known upper bound for the translation, though it remains open whether the bound is tight.

Theorem 12. For every PLTL$_f$ (resp. LTL$_f$) formula $\phi$ there exists an equivalent LTL$_f$ (resp. PLTL$_f$) formula whose size is at most triply exponential in the size of $\phi$, and which is computable in at most triply exponential time.

Proof. From a PLTL$_f$ formula $\phi$, get an equivalent DFA $A_\phi$ that may be exponentially larger than $\phi$ (Theorem 2), then reverse all transitions to get an NFA $A_\phi^R$ that accepts the reverse of the language of $A_\phi$, then determinize this NFA to get an equivalent DFA $A_\phi^{\text{lexp}}$. Note that $A_\phi^{\text{lexp}}$ may be exponentially larger than $A_\phi^R$. Now, apply Theorem 11 to transform this DFA into an equivalent PLTL$_f$ formula $\psi$. Finally, form the swap $\psi^S$ for the reverse language of $\psi$. Then, $\psi^S$ is the LTL$_f$ formula equivalent to the PLTL$_f$ formula $\phi$. Note that we reversed the language twice, and we incurred three exponential blowups.

Similarly, we can obtain a PLTL$_f$ formula from an LTL$_f$ one. From an LTL$_f$ formula $\phi$ build an equivalent DFA $A_\phi$, that may be double exponentially larger than $\phi$, and then apply Theorem 11 to get an equivalent PLTL$_f$ formula that may be single exponentially larger. \qed

Translating between PLDL$_f$ and LDL$_f$. Next we turn to PLDL$_f$ and LDL$_f$. Again the bound (and algorithm) in the theorem below is the best known upper bound for the translation. It is open whether the bound is tight.

Theorem 13. For every PLDL$_f$ (resp. LDL$_f$) formula $\phi$ there exists an equivalent LDL$_f$ (resp. PLDL$_f$) formula whose size is at most doubly exponential in the size of $\phi$, and which is computable in doubly exponential time.
Proof. From a PLDL$_f$ formula $\varphi$ build an equivalent DFA that may be exponentially larger (Theorem 2), then convert this using Kleene’s Theorem to a regular expression that may be exponentially larger, and then convert this to an LTL$_f$ formula with constant blowup [De Giacomo and Vardi, 2013]. The other case (from LTL$_f$/PLDL$_f$) follows by considering the swapped formulas. \qed

In light of the discussion in this section, we observe that while PLTL$_f$/PLDL$_f$ allows for exponentially smaller equivalent DFA compared to LTL$_f$/LDL$_f$, we cannot try to translate LTL$_f$/LDL$_f$ into PLTL$_f$/PLDL$_f$ to take advantage of this result, since the translation itself is too expensive. Hence, the properties of interest should be naturally expressible directly in PLTL$_f$/PLDL$_f$ to get the exponential improvement.

7 Impact of Adopting PLTL$_f$/PLDL$_f$

The exponential gain in transforming PLTL$_f$/PLDL$_f$ formulas into DFAs with respect to LTL$_f$/LDL$_f$ is reflected in an exponential gain in solving a variety of forms of sequential decision making involving temporal specifications. We start by focusing on Planning.

Planning in fully observable nondeterministic planning domains (FOND) for LTL$_f$/LDL$_f$ goals have been studied in [Camacho et al., 2017; De Giacomo and Rubin, 2018; Camacho et al., 2018]. A (rooted) fully observable nondeterministic domain is a tuple $D = (P, A, S, s_0, tr)$ where: (i) $P$ is a set of fluents (atomic propositions); (ii) $A$ is a set of actions (atomic symbols); (iii) $S = 2^P$ is the set of domain states; (iv) $s_0$ is the initial state (initial assignment to fluents); (v) $(s, a, s') \in tr$ represents action effects (including frame assumptions), and implicitly also actions preconditions. Since a domain is assumed to be represented compactly (e.g. in PDDL), we consider the size of the domain as the cardinality of $P$, i.e., logarithmic in the number of states (see e.g., [Geffner and Bonet, 2013]). In the case we are interested in the goal is expressed as an PLTL$_f$/PLDL$_f$ goal formula $\varphi_g$ over fluents $P$ holds. A plan $f$ is a strong solution to $D$ for goal $\varphi_g$ if every trace following the plan $f$ of $D$ is finite and satisfies $\varphi_g$. To find such a plan we adapt to PLTL$_f$/PLDL$_f$ the automata-based technique in [De Giacomo and Rubin, 2018].

Algorithm 3: FOND for PLTL$_f$/PLDL$_f$ goals

Given a FOND $D$ and a goal $\varphi_g$
1: Compute DFA $A_{\varphi_g}$ from goal $\varphi_g$ (1exp)
2: Compute product FOND $D_p$ over $D$ and $A_{\varphi_g}$ (lin)
3: Solve FOND $D_p$ for goal $G$ (poly in states of $D_p$)

The product between a FOND domain $D = (P, A, S, s_0, tr)$ and a goal DFA $A_{\varphi_g} = \langle 2^P, Q_g, q_0, q_{\delta_g}, F_g \rangle$ is the FOND domain $D_p = (P_p, A, S_p, s_{p,0}, tr_p)$ defined as follows: $P_p = P \cup Q_g$, where $Q_g$ are the extra states needed for the binary representation of the states in $Q_g$; state set $S_p = S \times Q_g$; initial state $s_{p,0} = (s_0, q_0, q_0)$; transitions $((s, a, (s', q')) \in tr_p$ iff $(s, a, s') \in tr$ and $q_\delta(q, s') = q'$. The goal to fulfill in $D_p$ is reaching the set of states $G = \{(s, q) | q \in F_g\}$.

Theorem 14. Solving FOND for PLTL$_f$/PLDL$_f$ goals is:
- EXPTIME-complete in the domain;
- EXPTIME-complete in the PLTL$_f$/PLDL$_f$ goals.

For the hardness in the domain, recall that the complexity for classical FOND planning, i.e., reachability goals, is already EXPTIME-hard [Rintanen, 2004]; and for the hardness in the goal, encode an entire planning problem for reachability goals into a PLTL$_f$ goal to be achieved in a trivial universal domain with a single state in which all actions have all possible effects. This as to be contrasted with the LTL$/LDL_f$ case, where FOND planning is EXPTIME-complete in the domain (compactly represented) and 2EXPTIME-complete in the goal [De Giacomo and Rubin, 2018]. The above construction can be adapted to handle (stochastically) fair domain [De Giacomo and Rubin, 2018; Aminof et al., 2020], with the same exponential advantage. Note that, if the domain is deterministic then the difference between LTL$_f$/LDL$_f$ and PLTL$_f$/PLDL$_f$ disappears because in the case of LTL$_f$/LDL$_f$ we can directly work with an NFA, since Step 3 reduces to solving simple reachability. Hence, in both cases the complexity becomes PSPACE in the domain and in the goal.

The crux of the above construction is that starting from the PLTL$_f$/PLDL$_f$ formula we build an exponential DFA, which is then combined through polynomial operation (the product) with the planning domain. An analogous line of reasoning can be exploited to show an exponential improvement in several other contexts. For example:

Solving MDPs with non-Markovian reward [Brafman et al., 1996; Thiébaux et al., 2006; Brafman et al., 2018] with PLTL$_f$/PLDL$_f$ rewards is EXPTIME-complete in the domain and EXPSPACE in the PLTL$_f$/PLDL$_f$ rewards, the latter is 2EXPTIME-complete for LTL$_f$/LDL$_f$ rewards [Brafman et al., 2018].

Reinforcement Learning where rewards are based on traces [De Giacomo et al., 2019; Camacho et al., 2019] with PLTL$_f$/PLDL$_f$ rewards also gain the exponential improvement.

Planning in non-Markovian domains [Brafman and De Giacomo, 2019a] expressed using PLTL$_f$/PLDL$_f$ is EXPSPACE-complete in the domain and EXPSPACE-complete in the PLTL$_f$/PLDL$_f$ goals., vs. 2EXPSPACE-complete both in the domain and in the goals/rewards in the case of LTL$_f$/LDL$_f$.

Solving non-Markovian decision processes [Brafman and De Giacomo, 2019b] expressed in PLTL$_f$/PLDL$_f$ is EXPSPACE-complete in the domain and EXPSPACE in the PLTL$_f$/PLDL$_f$ rewards. Again, this is an exponential improvement both in the domain and the rewards wrt the case of LTL$_f$/LDL$_f$.

8 Conclusion

We have reviewed PLTL$_f$ and its extension PLDL$_f$, which have an exponential advantage wrt LTL$_f$/LDL$_f$ in computing the corresponding DFA, which in turns positively impact several problems in AI. Though, to take advantage of this exponential improvement, the properties of interest must be directly expressed in PLTL$_f$/PLDL$_f$, because the translation between LTL$_f$/LDL$_f$ and PLTL$_f$/PLDL$_f$ dominates the exponential advantage of working with PLTL$_f$/PLDL$_f$. 

6
References

[Alechina et al., 2018] N. Alechina, B. Logan, and M. Dastani. Modeling norm specification and verification in multi-agent systems. *FLAP*, 5(2), 2018.

[Aminof et al., 2020] B. Aminof, G. De Giacomo, and S. Rubin. Stochastic fairness and language-theoretic fairness in planning on nondeterministic domains. In *ICAPS*, 2020.

[Bacchus and Kabanza, 1998] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Ann. of Math. and Artif. Intell.*, 22(1-2), 1998.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2), 2000.

[Bacchus et al., 1996] F. Bacchus, C. Boutilier, and A. Grove. Rewarding behaviors. In *AAAI/IAAI*, 1996.

[Bienvenu et al., 2006] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *KR*, 2006.

[Brafman and De Giacomo, 2019a] R. Brafman and G. De Giacomo. Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In *IJCAI*, 2019.

[Brafman and De Giacomo, 2019b] R. Brafman and G. De Giacomo. Regular decision processes: a model for non-markovian domains. In *IJCAI*, 2019.

[Brafman et al., 2018] R. Brafman, G. De Giacomo, and F. Patrizi. LTLf/LDLf non-Markovian rewards. In *AAAI*, 2018.

[Calvanese et al., 2002] D. Calvanese, G. De Giacomo, and M. Vardi. Reasoning about actions and planning in LTL action theories. In *KR*, 2002.

[Camacho et al., 2017] A. Camacho, E. Triantafillou, C. Muise, J. Baier, and S. McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*, 2017.

[Camacho et al., 2018] A. Camacho, J. Baier, C. Muise, and S. McIlraith. Finite LTL synthesis as planning. In *ICAPS*, 2018.

[Camacho et al., 2019] A. Camacho, R. Toro Icarte, T. Klassen, R. Valenzano, and S. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, 2019.

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

[De Giacomo and Rubin, 2018] G. De Giacomo and S. Rubin. Automata-theoretic foundations of FOND planning for LTLf and LDLf goals. In *IJCAI*, 2018.

[De Giacomo and Vardi, 1999] G. De Giacomo and M. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 1999.

[De Giacomo and Vardi, 2013] G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 2013.

[De Giacomo and Vardi, 2015] G. De Giacomo and M. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 2015.

[De Giacomo et al., 2019] G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *ICAPS*, 2019.

[Fagin et al., 1995] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[Fisher and Wooldridge, 2005] M. Fisher and M. Wooldridge. Temporal reasoning in agent-based systems. In *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1. Elsevier, 2005.

[Gabaldon, 2011] A. Gabaldon. Non-Markovian control in the situation calculus. *Artif. Intell.*, 175(1), 2011.

[Gabbay et al., 1980] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *POPL*, 1980.

[Gabbay, 1987] D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, 1987.

[Geffner and Bonet, 2013] H. Geffner and B. Bonet. A Coincise Introduction to Models and Methods for Automated Planning. Morgan&Claypool, 2013.

[Harel et al., 2000] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[Hopcroft and Ullman, 1979] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Kamp, 1968] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Ucla, 1968.

[Knobout et al., 2016] M. Knobout, M. Dastani, and J.J. Meyer. A dynamic logic of norm change. In *ECAI*. IOS Press, 2016.

[Lichtenstein et al., 1985] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *LNCS 163*. Springer, 1985.

[Maler and Pnueli, 1990] O. Maler and A. Pnueli. Tight bounds on the complexity of cascaded decomposition of automata. In *FOCS*, 1990.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[Rintanen, 2004] J. Rintanen. Complexity of planning with partial observability. In *ICAPS*, 2004.

[Thiébaux et al., 2006] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza. Decision-theoretic planning with non-markovian rewards. *J. Artif. Intell. Res.*, 25, 2006.

[Vardi, 1996] M. Vardi. An automata-theoretic approach to linear temporal logic. In *LNCS 1043*. Springer, 1996.

[Zhu et al., 2019] S. Zhu, G. Pu, and M. Vardi. First-order vs. second-order encodings for LTLf -to-automata translation. In *TAMC*, 2019.