Pure-Past Linear Temporal and Dynamic Logic on Finite Traces

Giuseppe De Giacomo[1] , **Antonio Di Stasio**[1] , **Francesco Fuggitti**[1,2] and **Sasha Rubin**[3]

[1]Università degli Studi di Roma "La Sapienza", Roma, Italy
[2]York University, Toronto, ON, Canada
[3]University of Sydney, Sydney, NSW, Australia
{degiacomo, distasio, fuggitti}@diag.uniroma1.it, sasha.rubin@sydney.edu.au

Abstract

We review PLTL_f and PLDL_f, the pure-past versions of the well-known logics on finite traces LTL_f and LDL_f, respectively. PLTL_f and PLDL_f are logics about the past, and so scan the trace backwards from the end towards the beginning. Because of this, we can exploit a foundational result on reverse languages to get an exponential improvement, over LTL_f/LDL_f, for 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 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 have been attracted by the clarity and ease 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], to express 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 Propositional Dynamic Logic (PDL) [Harel *et al.*, 2000], have been investigated [De Giacomo and Vardi, 2013], and have found application in several contexts. 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 can then be suitably employed in different contexts, as mentioned above.

LTL_f/LDL_f, like LTL originally, expresses temporal properties in a "pure-future fashion", i.e., referring only to the present and to 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, to say that we have accomplished our task and that since we were decontaminated we have been in clean areas, we write: $TaskDone \wedge (InCleanArea\ \mathcal{S}\ Decontaminated)$. The use of past temporal logics has been advocated for non-Markovian models in reasoning about actions [Gabaldon, 2011], for non-Markovian rewards in MDPs [Bacchus *et al.*, 1996], and for normative properties in multi-agent systems [Fisher and Wooldridge, 2005; Knobbout *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 been introduced in the literature as a technical means to get results for LTL and LTL_f [Maler and Pnueli, 1990; Zhu *et al.*, 2019]. 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, like LTL_f/LDL_f, also PLTL_f/PLDL_f can be translated into an *Alternating Finite-state Automaton* (AFA) in polynomial time, but, in the case of 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 problems involving temporal logics on finite traces in several contexts, including planning in non-deterministic domains (FOND) [Camacho *et al.*, 2017; De Giacomo and Rubin, 2018], reactive synthesis [De Giacomo and Vardi, 2015; Camacho *et al.*, 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].

Interestingly, PLTL$_f$ (resp., PLDL$_f$) and LTL$_f$ (resp., LDL$_f$) have the same expressive power. Indeed, we provide an algorithm to transform a PLTL$_f$ (resp., PLDL$_f$) formula into an LTL$_f$ (resp., LDL$_f$) formula. However, the transformation can be triple-exponentially (resp., double-exponentially) larger in the worst-case, and these are the best-known bounds.

Hence, it is not computationally sensible to start from LTL$_f$/LDL$_f$ 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 $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where (i) $\Sigma = 2^{\mathcal{P}}$ is a finite *input alphabet*; (ii) Q is a finite set of *states*; (iii) $q_0 \in Q$ is the *initial state*; (iv) $F \in 2^Q$ is the set of *accepting states*; (v) $\delta : Q \times \Sigma \to B^+(Q)$ is the *transition function*, where $B^+(Q)$ is the set of positive Boolean formulas over Q (i.e., built from the states using $\wedge$, $\vee$ and the constants $true$ and $false$). For instance, $\delta(q_1, a) = q_2 \vee (q_3 \wedge q_4)$ means that for the automaton to accept the input $a\tau$, from state q_1, it should either accept the input τ 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 φ.

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

The *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, 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 $\mathcal{A}$ is defined by introducing the function $Acc : \Sigma^* \to 2^Q$, where $q \in Acc(\tau)$ is read "input τ is accepted from state q", inductively given as follows:

1. $Acc(\epsilon) = F$,
2. $q \in Acc(a\tau)$ iff $V \models \delta(q, a)$ for some $V \subseteq Acc(\tau)$.

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

[1]A *run-tree* of $\mathcal{A}$ on input τ is a tree labeled by states such that i) 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 $q \in Acc(\tau)$ iff there is a run-tree on input τ whose root is labeled by q.

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 a DFA to a 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 $\mathcal{P}$ of atomic propositions, LTL$_f$ formulas φ are defined by:

$$\varphi \quad ::= \quad a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \circ\varphi \mid \varphi\,\mathcal{U}\,\varphi$$

where a denotes an atomic proposition in $\mathcal{P}$, $\circ$ is the *next* operator, and $\mathcal{U}$ is the *until* operator. We use abbreviations for other Boolean connectives, as well as the following: *eventually* as $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$; *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$; *weak next* $\bullet\varphi \equiv \neg\circ\neg\varphi$ (note that, on finite traces, $\neg\circ\varphi$ is not equivalent to $\circ\neg\varphi$); and *last (time point of the trace)* $last \equiv \bullet false$.

Formulas of LTL$_f$ are interpreted on *finite traces* $\tau = \tau_0\tau_1\cdots\tau_{n-1}$ where τ_i at instant i is a propositional interpretation over the alphabet $2^{\mathcal{P}}$. We denote by $length(\tau)$ the length n of τ. Given τ, an LTL$_f$ formula φ and an instant i, we define when φ *holds* at i, written $\tau, i \models \varphi$, by induction, as follows:

- $\tau, i \models a$ iff $a \in \tau_i$ (for $a \in \mathcal{P}$);
- $\tau, i \models \neg\varphi$ iff $\tau, i \not\models \varphi$;
- $\tau, i \models \varphi_1 \wedge \varphi_2$ iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \circ\varphi$ iff $i < length(\tau) - 1$ and $\tau, i+1 \models \varphi$;
- $\tau, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff for some $j,\ i \le j < length(\tau)\ \ \tau, j \models \varphi_2$, and for all $k,\ i \le k < j\ \ \tau, k \models \varphi_1$.

We write $\tau \models \varphi$, if $\tau, 0 \models \varphi$ and say that τ *satisfies* φ.

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$_f$. The syntax of LDL$_f$ is defined by:

$$\begin{array}{lcl} \varphi & ::= & tt \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\varrho\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho + \varrho \mid \varrho;\varrho \mid \varrho^* \end{array}$$

where ϕ denotes propositional formulas over $\mathcal{P}$ (we use the usual abbreviations, e.g., the Boolean constant $true$ is defined as $a \vee \neg a$, for some fixed $a \in \mathcal{P}$) and tt stands for logical true (not to be confused with the Boolean constant $true$). Expressions of the form ϱ are regular expressions (RE) over propositional formulas ϕ and the test construct $\varphi?$ typical of PDL. We abbreviate $[\varrho]\varphi \equiv \neg\langle\varrho\rangle\neg\varphi$ as in PDL, $ff \equiv \neg tt$ for false, and $\phi \equiv \langle\phi\rangle tt$ to denote the occurrence of the propositional formula ϕ. We also use $end \equiv [true]ff$ to express that the trace has ended.

Intuitively, $\langle\varrho\rangle\varphi$ states that, from the current instant in the trace, there exists an execution satisfying the RE ϱ such that its last instant satisfies φ, while $[\varrho]\varphi$ states that, from the current instant, all executions satisfying the RE ϱ are such that their last instant satisfies φ. Test constructs put into the execution path checks for satisfaction of additional LDL_f formulas.

Given a trace $\tau = \tau_0\tau_1\cdots\tau_{n-1}$ we denote by $\tau_{i,j}$ the sub-trace $\tau_i \ldots \tau_j$ if $j < length(\tau)$, $\tau_i \ldots \tau_{n-1}$ if $j \geq length(\tau)$. Note that if $i \geq length(\tau)$ then $\tau_{i,j}$ denotes the empty trace. Given a finite trace τ, an LDL_f formula φ, and an instant i, we define when φ *holds* at i, written $\tau, i \models \varphi$, by (mutual) induction, as follows:

- $\tau, i \models tt$;
- $\tau, i \models \neg\varphi$ iff $\tau, i \not\models \varphi$;
- $\tau, i \models \varphi_1 \wedge \varphi_2$ iff $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \langle\varrho\rangle\varphi$ iff there is a j such that $i \leq j$ and $\tau_{i,j} \in \mathcal{R}(\varrho)$ and $\tau, j \models \varphi$,

where the relation $\tau_{i,j} \in \mathcal{R}(\varrho)$ is inductively defined as:

- $\tau_{i,j} \in \mathcal{R}(\phi)$ if $j = i+1$, $i < length(\tau)$, and $\tau_i \models \phi$;
- $\tau_{i,j} \in \mathcal{R}(\varphi?)$ if $j = i$ and $\tau, i \models \varphi$;
- $\tau_{i,j} \in \mathcal{R}(\varrho_1 + \varrho_2)$ if $\tau_{i,j} \in \mathcal{R}(\varrho_1)$ or $\tau_{i,j} \in \mathcal{R}(\varrho_2)$;
- $\tau_{i,j} \in \mathcal{R}(\varrho_1; \varrho_2)$ if there exists $i \leq k \leq j$ such that $\tau_{i,k} \in \mathcal{R}(\varrho_1)$ and $\tau_{k,j} \in \mathcal{R}(\varrho_2)$;
- $\tau_{i,j} \in \mathcal{R}(\varrho^*)$ if $j = i$ or there exists k such that $\tau_{i,k} \in \mathcal{R}(\varrho)$ and $\tau_{k,j} \in \mathcal{R}(\varrho^*)$.

Note that if $i \geq length(\tau)$, the above definitions still apply.

Again, we say that a trace τ *satisfies* an LDL_f formula φ, written $\tau \models \varphi$, if $\tau, 0 \models \varphi$.

LTL_f/LDL_f to AFA and DFA. For every LTL_f/LDL_f formula φ there is an equivalent AFA $\mathcal{A}_\varphi$ accepting exactly the traces satisfying φ, which is linear in the size of φ [De Giacomo and Vardi, 2013]. Moreover, every AFA can be translated into an equivalent 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 translating an LTL_f/LDL_f formula into an equivalent DFA [De Giacomo and Vardi, 2013].

Algorithm 1: Translating LTL_f/LDL_f to DFA

Given an LTL_f/LDL_f formula φ

1: Compute an AFA equivalent to φ *(lin)*

2: Compute an NFA equivalent to the AFA *(1exp)*

3: Determinize the NFA obtaining an equivalent DFA *(1exp)*

3 PLTL_f and PLDL_f

We study the *pure-past* version of LTL_f, denoted as PLTL_f, and the *pure-past* version of LDL_f, denoted as PLDL_f. These are logics on finite traces that refer *only* to the past. PLTL_f and PLDL_f have a natural interpretation on finite traces: they are satisfied if they hold in the last instant of the trace.

PLTL_f. Given a set $\mathcal{P}$ of propositional symbols, PLTL_f is defined by:

$$\varphi \quad ::= \quad a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \ominus\varphi \mid \varphi\, \mathcal{S}\, \varphi$$

where $a \in \mathcal{P}$, $\ominus$ is the *before* operator and $\mathcal{S}$ is the *since* operator. Similarly to LTL_f and LDL_f, we define the following common abbreviations: the *once* operator $\diamondminus\varphi \equiv true\, \mathcal{S}\, \varphi$ and the *historically* operator $\boxminus\varphi \equiv \neg\diamondminus\neg\varphi$.

We define the satisfaction relation $\tau, i \models \varphi$, stating that φ holds at instant i, as follows. For atomic propositions and Boolean operators it is as for LTL_f. For past operators:

- $\tau, i \models \ominus\varphi$ iff $i \geq 1$ and $\tau, i-1 \models \varphi$;
- $\tau, i \models \varphi_1\, \mathcal{S}\, \varphi_2$ iff there exists k, with $0 \leq k \leq i$ 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 PLTL_f formula φ is *true* in τ, denoted $\tau \models \varphi$, if $\tau, length(\tau) - 1 \models \varphi$.

Example 1. *The property "we are now at location p_{23} and we have passed through location p_{12}" can be expressed in PLTL_f as $p_{23} \wedge \diamondminus p_{12}$. It is also expressible in LTL_f, although with a more complex formula, i.e., $\Diamond(p_{12} \wedge \Diamond(p_{23} \wedge last))$.*

Example 2. *The property "every time you took the bus, you bought a new ticket beforehand" can be expressed in PLTL_f as $\boxminus(takeB \Rightarrow \ominus(\neg takeB\, \mathcal{S}\, buyT))$, while it can be expressed in LTL_f as $(buyT\, \mathcal{R}\, takeB) \wedge \Box(takeB \Rightarrow (buyT \vee \bigcirc(buyT\, \mathcal{R}\, \neg takeB)))$ [Cimatti et al., 2004].*

PLDL_f. PLDL_f is the extension of PLTL_f with regular expressions in the eventualities. The syntax of PLDL_f is similar to the syntax of LDL_f except that we use a backward diamond operator. Intuitively, $\langle\!\langle\varrho\rangle\!\rangle\varphi$ states that there exists a point in the past, reachable (going backwards) through the regular expression ϱ from the current instant, where φ holds. Formally, the syntax of PLDL_f is defined by:

$$\begin{array}{lcl} \varphi & ::= & tt \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle\varrho\rangle\!\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho + \varrho \mid \varrho; \varrho \mid \varrho^* \end{array}$$

Define the satisfaction relation as for LDL_f, except that:

- $\tau, i \models \langle\!\langle\varrho\rangle\!\rangle\varphi$ iff there exists j, $0 \leq j \leq i$ such that $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho)$ and $\tau, j \models \varphi$,

where the relation $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho)$ is inductively defined as:

- $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\phi)$ if $j = i-1$, $i \geq 1$, and $\tau_i \models \phi$;
- $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varphi?)$ if $j = i$ and $\tau, i \models \varphi$;
- $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho_1 + \varrho_2)$ if $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho_1)$ or $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho_2)$;
- $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho_1; \varrho_2)$ if there exists $j \leq k \leq i$ such that $\tau_{k,i} \in \mathcal{R}_{\text{past}}(\varrho_1)$ and $\tau_{j,k} \in \mathcal{R}_{\text{past}}(\varrho_2)$;
- $\tau_{j,i} \in \mathcal{R}_{\text{past}}(\varrho^*)$ if $j = i$ or there exists $j \leq k \leq i$ such that $\tau_{k,i} \in \mathcal{R}_{\text{past}}(\varrho)$ and $\tau_{j,k} \in \mathcal{R}_{\text{past}}(\varrho^*)$.

We say that τ *satisfies* a PLDL_f formula φ, written $\tau \models \varphi$, if $\tau, length(\tau) - 1 \models \varphi$. As before, we use the abbreviation $[\![\varrho]\!]\varphi \equiv \neg\langle\!\langle\varrho\rangle\!\rangle\neg\varphi$. Moreover, we define $start \equiv [\![true]\!]ff$ to express the fact that the trace has just started.

Example 3. *The property "every time, if the cargo-ship departed (cs), then beforehand there was an alternation of grab and unload (unl) of containers" can be expressed in PLDL_f as $[\![true^*]\!](\langle\!\langle cs\rangle\!\rangle tt \Rightarrow \langle\!\langle(unl; grab)^*; (unl; grab)\rangle\!\rangle start)$, whereas it can be expressed in LDL_f as $\langle(\neg cs + (grab \wedge \neg cs); (unl; (grab \wedge \neg cs))^*; (cs \wedge unl)); \neg cs^*\rangle end$.*

As we will see later, all PLTL_f/PLDL_f formulas are translatable into LTL_f/LDL_f and vice versa, however the translation can be quite involved.

Just as for $\text{LTL}_f/\text{LDL}_f$, one can build a DFA accepting the traces satisfying φ. The same is true for $\text{PLTL}_f/\text{PLDL}_f$ formulas φ. However, since $\text{PLTL}_f/\text{PLDL}_f$ 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 $\text{LTL}_f/\text{LDL}_f$). The crux of this result lays on the possibility of obtaining from an AFA a DFA for the reverse language in single exponential time.

4 Reverse Languages and AFA

The *reverse* of a string $\tau = \tau_0\tau_1\dots\tau_{n-1}$ is the string $\tau^R = \tau_{n-1}\dots\tau_1\tau_0$, and the reverse of a language $\mathcal{L}$ is the language $\mathcal{L}^R = \{\tau^R : \tau \in \mathcal{L}\}$. Notably, while the minimal DFA equivalent to an AFA can be double-exponentially larger, the minimal DFA for the reverse language is at most single-exponentially larger [Chandra *et al.*, 1981], and can be easily built as shown below. For an AFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, define the DFA $\mathcal{A}^R = (\Sigma, S, s_{init}, T, F')$ that recognizes the reverse language where:

- $S = 2^Q$, $s_{init} = F$;
- for $v \in S, a \in \Sigma$, define $T(V, a)$ to be the set of all q such that $V \models \delta(q, a)$;
- let $V \in F'$ iff $q_0 \in V$.

Note that the size of $\mathcal{A}^R$ is $2^{O(|\mathcal{A}|)}$. Moreover, $\mathcal{A}^R$ can be computed in exponential time.

Theorem 1. *[Chandra* et al.*, 1981] The* DFA $\mathcal{A}^R$ *recognizes the reverse of the language of the* AFA $\mathcal{A}$.

Proof. Introduce a generalization of Acc defined in Section 2: $q \in Fwd(\tau, X)$ which is intuitively read as "the automaton reads τ forward from state q and results in a state in the set X". Formally, define $Fwd : \Sigma^* \times 2^Q \to 2^Q$ inductively:

(i) $Fwd(\epsilon, X) = X$,

(ii) $Fwd(a\tau, X) = \{q : Fwd(\tau, X) \models \delta(q, a)\}$.

Note that Acc is definable in terms of Fwd, i.e., $q \in Acc(\tau)$ iff $q \in Fwd(\tau, F)$. Intuitively, Fwd processes the input word in the forward direction. Also, for processing the input τ in the backward direction, define $Bck : \Sigma^* \times 2^Q \to 2^Q$ inductively:

(iii) $Bck(\epsilon, X) = X$,

(iv) $Bck(\tau a, X) = Bck(\tau, \{s : X \models \delta(s, a)\})$.

By (ii) and (iv) we get the following property corresponding to the shift from $(\tau a)\tau'$ to $\tau(a\tau')$: $Bck(\tau a, Fwd(\tau', X)) = Bck(\tau, Fwd(a\tau', X))$. Then, an induction on the length of τ shows $Bck(\tau, Fwd(\tau', X)) = Fwd(\tau\tau', X)$ for all τ'. Indeed, for the base case $\tau = \epsilon$, note that $Bck(\epsilon, Fwd(\tau', X)) = Fwd(\tau', X)$ by (iii), and for the general case, say $\tau = \tau'' a$, note that $Bck(\tau'' a, Fwd(\tau', X)) = Bck(\tau'', Fwd(a\tau', X))$ by the shifting property above, which equals $Fwd(\tau''(a\tau'), X)$ by the inductive hypothesis, which equals $Fwd((\tau'' a)\tau', X)$ as required.

We return to the proof of Theorem 1. We prove, by induction on τ, that $q_0 \in Fwd(\tau, V)$ iff $\mathcal{A}^R$ accepts τ^R from state V. If $\tau = \epsilon$ then both sides are equivalent to $q_0 \in V$. Consider $\tau = \tau' a$. Then $Fwd(\tau' a, V) = Bck(\tau' a, V) = Bck(\tau', \{s : V \models \delta(s, a)\}) = Bck(\tau', T(V, a)) = Fwd(\tau', T(V, a))$. Thus, $q_0 \in Fwd(\tau' a, V) = Fwd(\tau', T(V, a))$ iff $\mathcal{A}^R$ accepts $(\tau')^R$ from state $T(V, a)$ (by induction), iff $\mathcal{A}^R$ accepts $a(\tau')^R = (\tau' a)^R$ from state V, as required. So, $\mathcal{A}$ accepts τ iff $\mathcal{A}^R$ accepts τ^R. □

5 From $\text{PLTL}_f/\text{PLDL}_f$ to DFA

We take advantage of the single exponential reduction of an AFA to a DFA for the reverse language to get a DFA for $\text{PLTL}_f/\text{PLDL}_f$ formulas, which is single exponential in the size of the formula. To do so, we introduce the syntactic notion of *swap*, which, given a $\text{PLTL}_f/\text{PLDL}_f$ formula, produces an $\text{LTL}_f/\text{LDL}_f$ formula by *syntactically* replacing each past operator with its corresponding future operator. Intuitively, $\ominus$ corresponds to $\circ$, $\mathcal{S}$ corresponds $\mathcal{U}$, $\langle\!\langle \varrho \rangle\!\rangle$ corresponds to $\langle \varrho^{\mathtt{sw}} \rangle$, and $[\![\varrho]\!]$ corresponds to $[\varrho^{\mathtt{sw}}]$, where $\varrho^{\mathtt{sw}}$ is the regular expression ϱ, with all formulas in test constructs replaced by the corresponding swapped formulas. Formally, we define $\varphi^{\mathtt{sw}}$ by induction: (i) $a^{\mathtt{sw}} = a$ (for all $a \in \mathcal{P}$) and $tt^{\mathtt{sw}} = tt$; (ii) $(\neg\varphi)^{\mathtt{sw}} = \neg\varphi^{\mathtt{sw}}$ and $(\varphi_1 \wedge \varphi_2)^{\mathtt{sw}} = \varphi_1^{\mathtt{sw}} \wedge \varphi_2^{\mathtt{sw}}$; (iii) $(\ominus\varphi)^{\mathtt{sw}} = \circ\varphi^{\mathtt{sw}}$, (iv) $(\varphi_1 \,\mathcal{S}\, \varphi_2)^{\mathtt{sw}} = \varphi_1^{\mathtt{sw}} \,\mathcal{U}\, \varphi_2^{\mathtt{sw}}$; (v) $(\langle\!\langle \varrho \rangle\!\rangle\varphi)^{\mathtt{sw}} = \langle \varrho^{\mathtt{sw}} \rangle \varphi^{\mathtt{sw}}$ and $([\![\varrho]\!]\varphi)^{\mathtt{sw}} = [\varrho^{\mathtt{sw}}]\varphi^{\mathtt{sw}}$; (vi) $\phi^{\mathtt{sw}} = \phi$, and $(\varphi?)^{\mathtt{sw}} = (\varphi^{\mathtt{sw}})?$, and $(\rho_1 + \rho_2)^{\mathtt{sw}} = \rho_1^{\mathtt{sw}} + \rho_2^{\mathtt{sw}}$; (vii) $(\rho_1; \rho_2)^{\mathtt{sw}} = (\rho_1^{\mathtt{sw}}; \rho_2^{\mathtt{sw}})$, and $(\rho^*)^{\mathtt{sw}} = (\rho^{\mathtt{sw}})^*$. Similarly, we can swap an $\text{LTL}_f/\text{LDL}_f$ formula φ into a $\text{PLTL}_f/\text{PLDL}_f$ formula $\varphi^{\mathtt{sw}}$.

The following lemma summarizes the relation between formulas and their swaps.

Lemma 1. *If φ is a* $\text{PLTL}_f/\text{PLDL}_f$ *(resp.,* $\text{LTL}_f/\text{LDL}_f$*) formula, its swap $\varphi^{\mathtt{sw}}$ is an* $\text{LTL}_f/\text{LDL}_f$ *(resp.,* $\text{PLTL}_f/\text{PLDL}_f$*) formula of size $|\varphi|$ such that $\tau \models \varphi$ iff $\tau^R \models \varphi^{\mathtt{sw}}$, i.e., $\mathcal{L}^R(\varphi) = \mathcal{L}(\varphi^{\mathtt{sw}})$.*

We present two examples that illustrate the syntactic (vs. semantic) relationship between a formula and its swap.

Example 4. *Consider the* PLTL_f *formula "$inRoom \wedge roomDecontaminated \wedge \diamondminus(getPermit)$" — a variant of the example in the introduction. Its swapped* LTL_f *formula is "$inRoom \wedge roomDecontaminated \wedge \Diamond(getPermit)$". Although the two formulas are syntactically similar, they have different meanings. The former says that the robot is in a decontaminated room and it acquired the permit to enter the room beforehand. The latter says that the robot is in a decontaminated room and later (!) it will get the permit to enter.*

Example 5. *Consider the* LTL_f *formula "$batteryCharged \wedge \Diamond(useNotebook)$" and its swapped* PLTL_f *formula "$batteryCharged \wedge \diamondminus(useNotebook)$". While the first says that the battery is now charged and you can eventually use the notebook, the* PLTL_f *formula says that you used the notebook in the past, but the battery is charged now.*

Now, we are ready to show that transforming $\text{PLTL}_f/\text{PLDL}_f$ formulas into DFA can be done in exponential time (vs. double exponential time for $\text{LTL}_f/\text{LDL}_f$ formulas):

Theorem 2. *For every* $\text{PLTL}_f/\text{PLDL}_f$ *formula φ there is an equivalent* DFA $\mathcal{A}_\varphi$ *whose size is at most $2^{O(|\varphi|)}$ in the size of φ, and which is computable in at most exponential time.*

Proof. Swap the $\text{PLTL}_f/\text{PLDL}_f$ formula φ getting the $\text{LTL}_f/\text{LDL}_f$ $\varphi^{\mathtt{sw}}$, then construct the AFA $A_{\varphi^{\mathtt{sw}}}$, and, finally, build the DFA $\mathcal{A}_\varphi = \mathcal{A}^R_{\varphi^{\mathtt{sw}}}$. By Lemma 1 and Theorem 1 we get that $\mathcal{A}_\varphi$ has size $2^{O|\varphi|}$ and $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. □

Hence, we can define the analogue of Algorithm 1 to translate PLTL_f/PLDL_f formulas into DFA based on Theorem 2.

Algorithm 2: Translating PLTL_f/PLDL_f to DFA
Given a PLTL_f/PLDL_f formula φ
1: Swap φ into the corresponding LTL_f/LDL_f $\varphi^{\mathtt{sw}}$ *(lin)*
2: Compute AFA for $\varphi^{\mathtt{sw}}$ *(lin)*
3: Compute DFA from AFA for the reverse language *(1exp)*

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 are 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 instants and that can be quantified existentially and universally; the binary predicate $<$ denoting the order of instants; equality $=$ between instants; 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 use this fact to 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 φ replace $x < y$ by $y < x$ to get an FOL $\varphi^{\mathtt{sw}}$ for the reverse language, i.e., $w \models \varphi$ iff $w^R \models \varphi^{\mathtt{sw}}$. Then, translate the FOL formula $\varphi^{\mathtt{sw}}$ into an equivalent LTL_f formula ψ [Gabbay *et al.*, 1980]. Then, the PLTL_f formula $\psi^{\mathtt{sw}}$ is equivalent to the original FOL formula φ. $\square$

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. *RE is as least as expressive as PLDL_f.*

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

The reverse direction also holds:

Theorem 8. *PLDL_f is as least as expressive as RE.*

Proof. Given a regular expression ϱ compute the reverse regular expression ϱ and return $\langle\!\langle \varrho \rangle\!\rangle start$. $\square$

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 characterizations.

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, in general, can be 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., LTL_f) is not trivial. 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 gives 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 φ there exists an equivalent LTL_f (resp., PLTL_f) formula whose size is at most triply exponential in the size of φ, and which is computable in at most triply exponential time.*

Proof. Given a PLTL_f formula φ, build an equivalent DFA $\mathcal{A}_\varphi$ by Theorem 2. Note that the DFA may be exponentially larger than φ. Reverse all transitions to get an NFA $\mathcal{A}^R_\varphi$ that accepts the reverse of the language of $\mathcal{A}_\varphi$, then determinize this NFA to get an equivalent DFA $\mathcal{A}'^R_\varphi$. Note that $\mathcal{A}'^R_\varphi$ may be exponentially larger than $\mathcal{A}^R_\varphi$. Now, apply Theorem 11 to transform this DFA into an equivalent PLTL_f formula ψ. Finally, form the swap $\psi^{\mathtt{sw}}$ for the reverse language of ψ. Then, $\psi^{\mathtt{sw}}$ is the LTL_f formula equivalent to the PLTL_f formula φ. Note that we reversed the language twice, and we incurred in three exponential blowups.

Similarly, we can obtain a PLTL_f formula from an LTL_f one. From an LTL_f formula φ, build an equivalent DFA $\mathcal{A}_\varphi$ that may be double-exponentially larger than φ, and, then, apply Theorem 11 to get an equivalent PLTL_f formula which may be single-exponentially larger. $\square$

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* φ *there exists an equivalent* LDL_f *(resp.,* PLDL_f*) formula whose size is at most doubly exponential in the size of* φ*, and which is computable in doubly exponential time.*

Proof. From a PLDL_f formula φ build, an equivalent DFA that may be exponentially larger (Theorem 2), then, using Kleene's Theorem, convert this to a regular expression that may be exponentially larger, and, finally, convert this to an LDL_f formula with constant blow-up [De Giacomo and Vardi, 2013]. The other case (from LDL_f to PLDL_f) follows by considering the swapped formulas. □

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

7 Impact of Adopting $\text{PLTL}_f/\text{PLDL}_f$

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

Planning in fully observable nondeterministic planning domains (FOND) for $\text{LTL}_f/\text{LDL}_f$ goals has 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 $\mathcal{D} = \langle P, A, S, s_0, tr\rangle$ 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]). We are interested in the case where the goal is given as a $\text{PLTL}_f/\text{PLDL}_f$ *goal* formula φ_g over fluents P. A plan f is a *strong solution to* $\mathcal{D}$ *for goal* φ_g if every trace following the plan f of $\mathcal{D}$ is finite and satisfies φ_g. To find such a plan we use the automata-based technique in [De Giacomo and Rubin, 2018], but exploit the fact that $\text{PLTL}_f/\text{PLDL}_f$ goals give us a single exponential DFA.

Theorem 14. *Solving FOND for* $\text{PLTL}_f/\text{PLDL}_f$ *goals is EXPTIME-complete in the domain and EXPTIME-complete in the* $\text{PLTL}_f/\text{PLDL}_f$ *goals.*

Contrast this result with the $\text{LTL}_f/\text{LDL}_f$ case, where FOND planning is EXPTIME-complete in the domain (compactly represented) and 2EXPTIME-complete in the goal [De Giacomo and Rubin, 2018]. As mentioned in Section 6, the exponential gain is only achieved for properties expressed directly in $\text{PLTL}_f/\text{PLDL}_f$. If we first express the specification in $\text{LTL}_f/\text{LDL}_f$ and then translate it into $\text{PLTL}_f/\text{PLDL}_f$, we lose the advantage due to a blow-up in the translation. Thus, our approach yields an improvement for problems that can *natively* be specified in $\text{PLTL}_f/\text{PLDL}_f$, without resorting to $\text{LTL}_f/\text{LDL}_f$ at all. The above construction can be adapted to handle (stochastically) fair domains [De Giacomo and Rubin, 2018; Aminof *et al.*, 2020] with the same exponential advantage.

Note that if the domain is deterministic the difference between $\text{LTL}_f/\text{LDL}_f$ and $\text{PLTL}_f/\text{PLDL}_f$ disappears because in the former case we can directly work with an NFA, since it is sufficient to solve simple reachability, i.e., nonemptiness (cf. [De Giacomo and Rubin, 2018]). 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 $\text{PLTL}_f/\text{PLDL}_f$ formula we build an exponential DFA, which is then combined through a 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 as we will show in what follows.

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

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

Planning in non-Markovian domains [Brafman and De Giacomo, 2019a], with both the non-Markovian domain and the goal expressed in $\text{PLTL}_f/\text{PLDL}_f$ is EXPTIME-complete in the domain and in the goal, vs. 2EXPTIME-complete in the domain and in the goal in the case these are expressed in $\text{LTL}_f/\text{LDL}_f$.

Solving non-Markovian decision processes [Brafman and De Giacomo, 2019b], with both the system dynamics and the rewards expressed using $\text{PLTL}_f/\text{PLDL}_f$, is EXPTIME-complete in the domain and in the rewards specification. Again, this is an exponential improvement both in the domain and the rewards wrt the case of $\text{LTL}_f/\text{LDL}_f$.

8 Conclusion

We reviewed PLTL_f and its extension PLDL_f, which have an exponential advantage over $\text{LTL}_f/\text{LDL}_f$ when computing the corresponding DFA, which, in turn, positively impacts several problems in AI. However, to take advantage of this exponential improvement, the properties of interest must be directly expressed in $\text{PLTL}_f/\text{PLDL}_f$ because the translation between $\text{LTL}_f/\text{LDL}_f$ and $\text{PLTL}_f/\text{PLDL}_f$ dominates the exponential advantage of working with $\text{PLTL}_f/\text{PLDL}_f$.

Acknowledgments

Work partially supported by the European Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (No. 834228).

References

[Alechina *et al.*, 2018] N. Alechina, B. Logan, and M. Dastani. Modeling norm specification and verification in multiagent 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.

[Cimatti *et al.*, 2004] A. Cimatti, M. Roveri, and D. Sheridan. Bounded verification of past ltl. In *FMCAD*, 2004.

[De Giacomo and Rubin, 2018] G. De Giacomo and S. Rubin. Automata-theoretic foundations of FOND planning for LTLf and LDL_f 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 *H. of Temporal Reasoning in AI*. 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 Concise 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.

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

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

[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.