Linear Temporal Logic and Linear Dynamic Logic on Finite Traces

Giuseppe De Giacomo
Sapienza Università di Roma
Roma, Italy
degiacomo@dis.uniroma1.it

Moshe Y. Vardi
Rice University,
Houston, TX, USA
vardi@cs.rice.edu

Abstract

In this paper we look into the assumption of interpreting LTL over finite traces. In particular we show that LTL_f, i.e., LTL under this assumption, is less expressive than what might appear at first sight, and that at essentially no computational cost one can make a significant increase in expressiveness while maintaining the same intuitiveness of LTL_f interpreted over finite traces. Indeed, we propose a logic, LDL_f for *Linear Dynamic Logic over finite traces*, which borrows the syntax from Propositional Dynamic Logic (PDL), but is interpreted over finite traces. Satisfiability, validity and logical implication (as well as model checking) for LDL_f are PSPACE-complete as for LTL_f (and LTL).

1 Introduction

Several research areas of AI have been attracted by the simplicity and naturalness of Linear Time Logic (LTL) [33] for temporal specification of the course of actions of an agent or a system of agents [17]. In particular in reasoning about actions and planning, LTL is often used as a specification mechanism for temporally extended goals [2; 13; 11; 31; 18], for constraints on plans [2; 3; 20], and for expressing preferences and soft constraints [5; 6; 38].

Quite often, especially in the context of temporal constraints and preferences, LTL formulas are used to express properties or constraints on *finite* traces of actions/states; in fact, this can be done even if the standard semantics of LTL is defined on infinite traces [33]. Similarly, in the area of Business Process Specification and Verification [43; 32], variants of LTL are used to specify processes declaratively, but these variants are interpreted over finite traces. Yet, there has been little discussion in the AI literature about the differences arising from interpreting LTL over finite or infinite traces.

In this paper we look into case where LTL is interpreted over finite traces. In particular we show that LTL, in this case, is less expressive than what might appear at first sight, and that at essentially no cost one can make a significant increase in expressiveness while maintaining the same intuitiveness of LTL interpreted over finite traces.

Specifically, we recall that LTL interpreted over finite traces has the expressive power of First Order Logic (FOL) over finite ordered traces and that of star-free regular expressions [14; 30; 34; 46]. We also notice that Monadic Second Order Logic (MSO) over finite ordered traces is expressively equivalent to regular expressions and finite-state automata [9; 16; 42]. In other words, regular expressions and finite-state automata properly subsume LTL over finite traces.

This observation raises the question of why one restricts oneself to LTL over finite traces instead of adopting a more expressive formalism such as regular expressions or finite state automata. We believe that one key obstacle is that regular expressions and finite-state automata are perceived as too procedural and possibly low level to be an attractive specification formalisms. We propose here an extension of LTL over finite traces that has the same expressive power as MSO over finite traces, while retaining the declarative nature and intuitive appeal of LTL. Our logic, is called LDL_f for *Linear Dynamic Logic over finite traces*. It is an adaptation of LDL, introduced in [44], which is interpreted over infinite traces. Both LDL and LDL_f borrows the syntax from Propositional Dynamic Logic (PDL) [19], but are interpreted over traces.

We show how to immediately capture an LTL formula as an equivalent LDL_f formula, as well as how to capture a regular expression or finite-state automata-based specification as an LDL_f formula. We then show that LDL_f shares the same computational characteristics of LTL [45]. In particular, satisfiability, validity and logical implication (as well as model checking) are PSPACE-complete (with potential exponentiality depending only on the formula in the case of model checking). To do so we resort to a polynomial translation of LDL_f formulas into alternating automata on finite traces [8; 12; 27], whose emptiness problem is known to be PSPACE-complete [12]. The reduction actually gives us a practical algorithm for reasoning in LDL_f. Such an algorithm can be implemented using symbolic representation, see, e.g.,[7], and thus promises to be quite scalable in practice, though we leave this for further research.

2 Linear Time Logic on Finite Traces (LTL_f)

Linear Temporal Logic (LTL) over infinite traces was originally proposed in Computer Science as a specification language for concurrent programs [33]. The variant of LTL interpreted over finite traces that we consider is that of [5; 20; 32; 43; 46], called here LTL_f. Such a logic uses the same syntax as that of the original LTL. Namely, *formulas* of LTL_f are built from a set $\mathcal{P}$ of propositional symbols and are closed under the boolean connectives, the unary temporal operator $\circ$ (*next-time*) and the binary temporal operator $\mathcal{U}$ (*until*):

$$\varphi ::= A \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (\circ\varphi) \mid (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

with $A \in \mathcal{P}$.

Intuitively, $\circ\varphi$ says that φ holds at the *next* instant, $\varphi_1 \,\mathcal{U}\, \varphi_2$ says that at some future instant φ_1 will hold and *until* that point φ_2 holds. Common abbreviations are also used, including the ones listed below.

- Standard boolean abbreviations, such as $true$, $false$, $\vee$ (or) and $\rightarrow$ (implies).
- $Last$, which stands for $\neg\circ true$, and denotes the last instant of the sequence. Notice that in LTL over infinite traces $Last$, which in that case is equivalent to $\circ false$ is indeed always false. When interpreted on finite traces however it becomes true exactly at the last instant of the sequence.
- $\Diamond\varphi$ which stands for $true \,\mathcal{U}\, \varphi$, and says that φ will *eventually* hold before the last instant (included).
- $\Box\varphi$, which stands for $\neg\Diamond\neg\varphi$, and says that from the current instant till the last instant φ will *always* hold.

The semantics of LTL_f is given in terms of LT_f-interpretations, i.e., interpretations over a *finite traces* denoting a finite sequence of consecutive instants of time. LT_f-interpretations are represented here as finite words π over the alphabet of $2^{\mathcal{P}}$, i.e., as alphabet we have all the possible propositional interpretations of the propositional symbols in $\mathcal{P}$. We use the following notation. We denote the *length* of a trace π as $length(\pi)$. We denote the *positions*, i.e. instants, on the trace as $\pi(i)$ with $0 \leq i \leq last$, where $last = length(\pi) - 1$ is the last element of the trace. We denote by $\pi(i, j)$ the *segment* (i.e., the subword) obtained from π starting from position i and terminating in positon j, $0 \leq i \leq j \leq last$.

Given an LT_f-interpretation π, we inductively define when an LTL_f formula φ *is true* at an instant i (for $0 \leq i \leq last$), in symbols $\pi, i \models \varphi$, as follows:

- $\pi, i \models A$, for $A \in \mathcal{P}$ iff $A \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$.
- $\pi, i \models \circ\varphi$ iff $i < last$ and $\pi, i{+}1 \models \varphi$.
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff for some j such that $i \leq j \leq last$, we have that $\pi, j \models \varphi_2$ and for all k, $i \leq k < j$, we have that $\pi, k \models \varphi_1$.

A formula φ *is true* in π, in notation $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula φ is *satisfiable* if it is true in some LT_f-interpretation, and is *valid*, if it is true in every LT_f-interpretation. A formula φ logically implies a formula φ', in notation $\varphi \models \varphi'$ if for every LT_f-interpretation π we have that $\pi \models \varphi$ implies $\pi \models \varphi'$. Notice that satiability, validity and logical implication are all immediately mutually reducible to each other.

Theorem 1. [37] *Satisfiability, validity, and logical implication for* LTL_f *formulas are PSPACE-complete.*

Proof. PSPACE membership follows from PSPACE completeness of LTL on infinite traces [37]. Indeed, it is easy to reduce LTL_f satisfiability into LTL (on infinite traces) satisfiability as follows: (*i*) introduce a proposition "$Tail$"; (*ii*) require that $Tail$ holds at 0 ($Tail$); (*iii*) require that $Tail$ stays true until it fails and then stay failed ($Tail \,\mathcal{U}\, \Box\neg Tail$); (*iv*) translate the LTL_f formula into an LTL formulas as follows:

- $t(P) \mapsto P$
- $t(\neg\varphi) \mapsto \neg t(\varphi)$
- $t(\varphi_1 \wedge \varphi_2) \mapsto t(\varphi_1) \wedge t(\varphi_2)$
- $t(\circ\varphi) \mapsto \circ(Tail \wedge t(\varphi))$
- $t(\varphi_1 \,\mathcal{U}\, \varphi_2) \mapsto (\varphi_1)\mathcal{U}(Tail \wedge t(\varphi))$

PSPACE-hardness can be obtained adapting the original hardness proof for LTL on infinite trace in [37]. Here, however, we show it by observing that we can easily reduce (propositional) STRIPS planning, which is PSPACE-complete [10] into LTL_f satisfiability. The basic idea is to capture in LTL_f runs over the planning domain (the plan itself is a *good* run). We can do this as follows. For each operator/action $A \in Act$ with precondition φ and effects $\bigwedge_{F \in Add(A)} F \wedge \bigwedge_{F \in Del(A)} \neg F$ we generate the following LTL_f formulas: *(i)* $\Box(\circ A \rightarrow \varphi)$: that is, always if next action A as occurred (denoted by a proposition A) then now φ must be true; *(ii)* $\Box(\circ A \rightarrow \circ(\bigwedge_{F \in Add(A)} F \wedge \bigwedge_{F \in Del(A)} \neg F))$, that is, when A as occurs, its effects are true; *(iii)* $\Box(\circ A \rightarrow \bigwedge_{F \notin Add(A) \cup Del(A)} (F \equiv \circ F))$, that is, everything that is not in the add or delete list of A remains unchanged, this is the so called the frame axiom [29]. We then say that at every step one and only one action is executed: $\Box((\bigvee_{A \in Act} A) \wedge (\bigwedge_{A_i, A_j \in Act, A_i \neq A_j} A_i \rightarrow \neg A_j))$. We encode the initial situation, described with a set of propositions $Init$ that are initially true, as the formula (that holds at the beginning of the sequence): $\bigwedge_{F \in Init} F \wedge \bigwedge_{F \notin Init} \neg F$. Finally, given a goal φ_g we require it to eventually hold: $\Diamond\varphi_g$. Then a plan exists iff the conjunction of all above formula is satisfiable. Indeed if it satisfiable there exists a sequence where eventually φ_g is true and such sequence is a run over the planning domain. $\Box$

Notice that in the proof above we encoded STRIPS effects in LTL_f, but it is equally easy to encode successor state axioms of the situation calculus (in the propositional case and instantiated to single actions) [36]. For the successor state axiom $F(do(A, s)) \equiv \varphi^+(s) \vee (F(s) \wedge \neg\varphi^-(s))$ we have:

$$\Box(\circ A \rightarrow (\circ F \equiv \varphi^+ \vee F \wedge \neg\varphi^-).$$

However, precondition axioms $Poss(A, s) \equiv \varphi_A(s)$ can only be captured in the part that says that if A happens then its precondition must be true:

$$\Box(\circ A \rightarrow \varphi_A).$$

The part that says that if the precondition φ_A holds that action A is *possible* cannot be expressed in linear time formalisms since they talk about the runs that actually happen not the one that are possible. See, e.g., the discussion in [11].

While, as hinted above, LTL_f is able to capture the runs of an arbitray transition system by expressing formulas about the current state and the next, when we consider more sophisticated temporal properties, LTL_f on finite traces presents us with some surprises. To see this, let us look at some classical LTL formulas and their meaning on finite traces.

- "Safety": $\Box\varphi$ means that always *till the end of the trace* φ holds.

- "Liveness": $\Diamond\varphi$ means that eventually *before the end of the trace* φ holds. By the way, the term "liveness" is not fully appropriate, since often the class of the liveness properties is mathematically characterized exactly as that of those properties that cannot be checked within any finite length run. Refer, e.g., to Chapter 3 of [4] for details.
- "Response": $\Box\Diamond\varphi$ means that for any point in the trace there is a point later in the trace where φ holds. But this is equivalent to say that the *last point in the trace satisfies* φ, i.e., it is equivalent to $\Diamond(Last \wedge \varphi)$. Notice that this meaning is completely different from the meaning on infinite traces and cannot be considered a "fairness" property as "response" is in the infinite case.
- "Persistence": $\Diamond\Box\varphi$ means that there exists a point in the trace such that from then on till the end of the trace φ holds. But again this is equivalent to say that the *last point in the trace satisfies* φ, i.e., it is equivalent to $\Diamond(Last \wedge \varphi)$.

In other words, no direct nesting of eventually and always connectives is meaningful in LTL$_f$. In contrast, in LTL of infinite traces alternation of eventually and always have different meaning up to three level of nesting, see, e.g., Chapter 5 of [4] for details. Obviously, if we nest eventually and always indirectly, through boolean connectives, we do get interesting properties. For example, $\Box(\psi \rightarrow \Diamond\varphi)$ does have an interesting meaning also for finite traces: always, before the end of the trace, if ψ holds then later φ holds.

3 LTL$_f$ to FOL

Next we show how to translate LTL$_f$ into *first-order logic* (FOL) over finite linear order sequences[1]. Specifically, we consider a first-order language that is formed by the two binary predicates $succ$ and $<$ (which we use in the usual infix notation) plus equality, a unary predicate for each symbol in $\mathcal{P}$ and two constants 0 and $last$. Then we restrict our interest to *finite linear ordered* FOL *interpretation*, which are FOL interpretations of the form $\mathcal{I} = (\Delta^I, \cdot^{\mathcal{I}})$, where the domain is $\Delta^I = \{0, \ldots, n\}$ with $n \in \mathbb{N}$, and the interpretation function $\cdot^{\mathcal{I}}$ interprets binary predicates and constants in a fixed way:

- $succ^{\mathcal{I}} = \{(i, i+1) \mid i \in \{0, \ldots, n-1\}\}$,
- $<^{\mathcal{I}} = \{(i, j) \mid i, j \in \{0, \ldots, n\}$ and $i < j\}$,
- $=^{\mathcal{I}} = \{(i, i) \mid i \in \{0, \ldots, n\}\}$,
- $0^{\mathcal{I}} = 0$ and $last^{\mathcal{I}} = n$.

In fact, $succ$, $=$, 0 and $last$ can all be defined in terms of $<$. Specifically:

- $succ(x, y) \doteq (x < y) \wedge \neg\exists z. x < z < y$;
- $x = y \doteq \forall z. x < z \equiv y < z$;
- 0 can be defined as that x such that $\neg\exists y. succ(y, x)$, and $last$ as that x such that $\neg\exists y. succ(x, y)$.

For convenience we keep these symbols in the language. Also we use the usual abbreviation $x \leq y$ for $x < y \vee x = y$.

[1] More precisely *monadic first-order logic on finite linearly ordered domains*, sometimes denoted as FO[$<$].

In spite of the obvious notational differences, it is easy to see that finite linear ordered FOL interpretations and LT$_f$-interpretations are isomorphic. Indeed, given an LT$_f$-interpretation π we define the corresponding finite FOL interpretation $\mathcal{I} = (\Delta^I, \cdot^{\mathcal{I}})$ as follows: $\Delta^{\mathcal{I}} = \{0, \ldots, last\}$ (with $last = length(\pi) - 1$); with the obvious predefined predicates and constants interpretation, and, for each $A \in \mathcal{P}$, its interpretation is $A^{\mathcal{I}} = \{i \mid A \in \pi(i)\}$. Conversely, given a finite linear ordered FOL interpretation $\mathcal{I} = (\Delta^I, \cdot^{\mathcal{I}})$, with $\Delta^I = \{0, \ldots, n\}$, we define the corresponding LT$_f$-interpretation π as follows: $length(\pi) = n + 1$; and for each position $0 \leq i \leq last$, with $last = n$, we have $\pi(i) = \{A \mid i \in A^{\mathcal{I}}\}$.

We can then use a translation function $fol(\varphi, x)$ that given an LTL$_f$ formula φ and a variable x returns a corresponding FOL formula open in x. We define $fol()$ by induction on the structure of the LTL$_f$ formula:

- $fol(A, x) = A(x)$
- $fol(\neg\varphi, x) = \neg fol(\varphi, x)$
- $fol(\varphi \wedge \varphi', x) = fol(\varphi, x) \wedge fol(\varphi', x)$
- $fol(\circ\varphi, x) = \exists y. succ(x, y) \wedge fol(\varphi, y)$
- $fol(\varphi \, \mathcal{U} \, \varphi', x) = \exists y. x \leq y \leq last \wedge fol(\varphi', y) \wedge \forall z. x \leq z < y \rightarrow fol(\varphi, z)$

Theorem 2. *Given an* LT$_f$*-interpretation* π *and a corresponding finite linear ordered* FOL *interpretation* $\mathcal{I}$*, we have*

$$\pi, i \models \varphi \quad \textit{iff} \quad \mathcal{I}, [x/i] \models fol(\varphi, x)$$

where $[x/i]$ *stands for a variable assignment that assigns to the free variable* x *of* $fol(\varphi, x)$ *the value* i.

Proof. By induction on the LTL$_f$ formula. □

In fact also the converse reduction holds, indeed we have:

Theorem 3 ([21][2])**.** LTL$_f$ *has exactly the same expressive power of* FOL.

4 Regular Temporal Specifications (RE$_f$)

We now introduce regular languages, concretely represented as regular expressions or finite state automata [24; 26], as a form of temporal specification over finite traces. In particular we focus on regular expressions.

We consider as alphabet the set propositional interpretations $2^{\mathcal{P}}$ over the propositional symbols $\mathcal{P}$. Then RE$_f$ expressions are defined as follows: $\varrho ::= \emptyset \mid \mathcal{P} \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^*$, where $\mathcal{P} \in 2^{\mathcal{P}}$ and $\emptyset$ denotes the empty language. We denote by $\mathcal{L}(\varrho)$ the language recognized by a RE$_f$ expression ϱ ($\mathcal{L}(\emptyset)) = \emptyset$). In fact, it is convenient to introduce some syntactic sugar and redefine RE$_f$ as follows:

$$\varrho ::= \phi \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^*$$

where ϕ is a propositional formula that is an abbreviation for the union of all the propositional interpretations that satisfy ϕ, that is $\phi = \sum_{\mathcal{P} \models \phi} \mathcal{P}$ ($\emptyset$ is now abbreviated by $false$).

[2] Specifically, this result is a direct consequence of Theorem 2.2 in [21] on "discrete complete models", which include finite sequences. That theorem strengthen an analogous one in [25] by avoiding the use of past operators.

Notice that we interpret these expressions (with or without abbreviations) over the same kind of LT_f-interpretations used for LTL_f. Namely, we say that a RE_f expression is satisfied by an LT_f-interpretation π if $\pi \in \mathcal{L}(\varrho)$, we say that ϱ is true at instant i if $\pi(i, last) \in \mathcal{L}(\varrho)$, we say that ϱ is satisfied beween i, j if $\pi(i, j) \in \mathcal{L}(\varrho)$.

Here are some interesting properties that can be expressed using RE_f as a temporal property specification mechanism.

- "Safety": φ^*, which is equivalent to $\Box\phi$, and means that always, *until the end of the trace*, φ holds.
- "Liveness": $true^*; \varphi; true^*$, which is equivalent to the LTL_f formula $\Diamond\varphi$, and means that eventually *before the end of the trace* φ holds.
- "Conditional response": $true^*; (\neg\psi + true^*; \varphi)$, which is equivalent to $\Box(\psi \rightarrow \Diamond\varphi)$, and means that always before the end of the trace if ψ holds then later φ holds.
- "Ordered occurrence": $true^*; \varphi_1; true^*; \varphi_2; true^*$ that says φ_1 and φ_2 will both happen in order.
- "Alternating sequence": $(\psi; \varphi)^*$ that means that ψ and φ, not necessarily disjoint, alternate for the whole sequence starting with ψ and ending with φ.
- "Pair sequence": $(true; true)^*$ that means that the sequence is of even length.
- "Eventually on an even path φ": $(true; true)^*; \varphi; true^*$, i.e., we can constrain the path fulfilling the eventuality to satisfy some structural (regular) properties, in particular in this case that of $(true; true)^*$.

The latter three formulas cannot be expressed in LTL_f. More generally, the capability of requiring regular structural properties on paths, is exactly what is missing from LTL_f, as noted in [47].

Next, we consider *monadic second-order logic* MSO over bounded ordered sequences, see e.g., Chapter 2 of [26]. This is a strict extension of the FOL language introduced above, where we add the possibility of writing formulas of the form $\forall X.\varphi$ and $\exists X.\varphi$ where X is a monadic (i.e., unary) predicate variable and φ may include atoms whose predicate is such variable. Binary predicates and constants remain exactly those introduced above for FOL. The following classical result clarifies the relationship between RE_f and MSO, see e.g., [41] or Chapter 2 of [26].

Theorem 4 ([9; 16; 42])**.** *RE_f has exactly the same expressive power of* MSO.

Notice that MSO is strictly more expressive that FOL on finite ordered sequences.

Theorem 5 ([40])**.** *The expressive power of* FOL *over finite ordered sequences is strictly less than that of* MSO.

Recalling Theorem 3, this immediately implies that:

Theorem 6. *RE_f is strictly more expressive than LTL_f.*

In fact this theorem can be refined by isolating which sort of RE_f expressions correspond to LTL_f. These are the so-called *star-free regular expressions* (aka, counter-free regular languages) [30], which are the regular expressions obtained as follows:

$$\varrho ::= \phi \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \overline{\varrho}$$

where $\overline{\varrho}$ stands for the complement of ϱ, i.e., $\mathcal{L}(\overline{\varrho}) = (2^{\mathcal{P}})^*/\mathcal{L}(\varrho)$. Star-free regular expressions are strictly less espressive then RE_f since they do not allow for unrestrictedly expressing properties involving the Keene star $*$, which appears implicitly only to generate the universal language used in complementation.

Note, however, that several RE_f expressions involving $*$ can be rewritten by using complementation instead, including, it turns out, all the ones that correspond to LTL_f properties.

Here are some examples of RE_f expressions, which are indeed star-free.

- $(2^{\mathcal{P}})^* = true^*$ is in fact star-free, as it can be expressed as $\overline{false}$
- $true^*; \phi; true^*$ is star-free, as $true^*$ is star-free.
- ϕ^* for a propositional ϕ is also star-free, as it is equivalent to $\overline{true^*; \neg\phi; true^*}$.

A classical result on star-free regular expression is that:

Theorem 7 ([30])**.** *Star-free RE_f have exactly the same expressive power of* FOL.

Hence, by Theorem 3, we get the following result, see [14; 34; 46].

Theorem 8. *LTL_f has exactly the same expressive power of star-free RE_f.*

5 Linear-time Dynamic Logic (LDL_f)

As we have seen above, LTL_f is strictly less expressive that RE_f. On the other hand, RE_f is often consider too low level as a formalism for expressing temporal specifications. For example, RE_f expressions miss a direct construct for negation, for conjunction, and so forth (to see these limiting factors, one can try to encode in RE_f the STRIPS domain or the successor state axioms coded in LTL_f in Section 2). So it is natural to look for a formalism that merges the declarativeness and convenience of LTL_f with the expressive power of RE_f. This need is considered compelling also from a practical point of view. Indeed, industrial linear time specification languages, such as Intel *ForSpec* [1] and the standard *PSL* (Property Specification Language) [15], enhance LTL (on infinite strings) with forms of specifications based on regular expressions.

It may be tempting to simply add directly complementation and intersection to RE_f, but it is known that such extensions result in very high complexity; in particular, the nonemptiness problem (corresponding to satisfiability in our setting) for star-free regular expressions in *nonelementary*, which means that the complexity cannot be bounded by any fixed-heigh stack of exponentials [39].

Here we follow another approach and propose a temporal logic that merges LTL_f with RE_f in a very natural way. The logic is called LDL_f, *Linear Dynamic Logic of Finite Traces*, and adopts exactly the syntax of the well-know logic of programs PDL, *Propositional Dynamic Logic*, [19; 22; 23], but whose semantics is given in terms of finite traces. This logic is an adaptation of LDL, introduced in [44], which, like LTL, is interpreted over infinite traces.

Formally, LDL_f formulas are built as follows:

$$\begin{array}{lll}\varphi & ::= & A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\rho\rangle\varphi \\ \rho & ::= & \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1;\rho_2 \mid \rho^*\end{array}$$

where A denotes an atomic proposition in $\mathcal{P}$; ϕ denotes a propositional formulas over the atomic propositions in $\mathcal{P}$; ρ denotes path expressions, which are RE_f expressions over propositional formulas ϕ, with the addition of the test construct $\varphi?$ typical of PDL; and φ stand for LDL_f formulas built by applying boolean connectives and the modal connectives $\langle\rho\rangle\varphi$. Tests are used to insert into the execution path checks for satisfaction of additional LDL_f formulas. We use the usual boolean abbreviations as well as the abbreviation $[\rho]\varphi$ for $\neg\langle\rho\rangle\neg\varphi$.

Intuitively, $\langle\rho\rangle\varphi$ states that, from the current instant, there exists an execution satisfying the RE_f expression ρ such that its last instant satisfies φ. While $[\rho]\varphi$ states that, from the current instant, all executions satisfying the RE_f expression ρ are such that their last instant satisfies φ.

As for the semantics of LDL_f, for an LT_f-interpretation π, we inductively define when an LDL_f formula φ *is true* at an instant $i \in \{0, \ldots, \mathit{last}\}$, in symbols $\pi, i \models \varphi$, as follows:

- $\pi, i \models A$, for $A \in \mathcal{P}$ iff $A \in \pi(i)$
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$
- $\pi, i \models \varphi \wedge \varphi'$ iff $\pi, i \models \varphi$ and $\pi, i \models \varphi'$
- $\pi, i \models \langle\rho\rangle\varphi$ iff for some j such that $i \leq j \leq \mathit{last}$, we have that $(i, j) \in \mathcal{R}(\rho, \pi)$ and $\pi, j \models \varphi$

where the relation $\mathcal{R}(\rho, s)$ is defined inductively as follows:

- $\mathcal{R}(\phi, s) = \{(i, i+1) \mid \pi(i) \models \phi\}$ (ϕ propositional)
- $\mathcal{R}(\varphi?, s) = \{(i, i) \mid \pi, i \models \varphi\}$
- $\mathcal{R}(\rho_1 + \rho_2, s) = \mathcal{R}(\rho_1, s) \cup \mathcal{R}(\rho_2, s)$
- $\mathcal{R}(\rho_1; \rho_2, s) = \{(i, j) \mid$ exists k such that $(i, k) \in \mathcal{R}(\rho_1, s)$ and $(k, j) \in \mathcal{R}(\rho_2, s)\}$
- $\mathcal{R}(\rho^*, s) = \{(i, i)\} \cup \{(i, j) \mid$ exists k such that $(i, k) \in \mathcal{R}(\rho, s)$ and $(k, j) \in \mathcal{R}(\rho^*, s)\}$

Theorem 9. *LTL_f can be translated into LDL_f in linear time.*

Proof. We prove the theorem constructively, by exhibiting a translation function f form LTL_f to LDL_f

- $f(A) = A$
- $f(\neg\varphi) = \neg f(\varphi)$
- $f(\varphi_1 \wedge \varphi_2) = f(\varphi_1) \wedge f(\varphi_2)$
- $f(\circ\varphi) = \langle \mathit{true}\rangle f(\varphi)$
- $f(\varphi_1 \,\mathcal{U}\, \varphi_2) = \langle f(\varphi_1)^*\rangle f(\varphi_2)$

It is easy to see that for every LT_f-interpretation π, we have $\pi, i \models \varphi$ iff $\pi, i \models f(\varphi)$. □

Theorem 10. *RE_f can be translated into LDL_f in linear time.*

Proof. We prove the theorem constructively, by exhibiting a translation function g form RE_f to LDL_f (here Last stands for $[\mathit{true}]\mathit{false}$):

$$g(\varrho) = \langle\varrho\rangle \mathit{Last}.$$

It is easy to see that $\pi, i \models \varrho$ iff $\pi(i, \mathit{last}) \in \mathcal{L}(\varrho)$ iff for some $i \leq j \leq \mathit{last}$, we have that $(i, j) \in \mathcal{R}(\varrho, \pi)$ and $\pi, j \models \mathit{Last}$ iff $\pi, i \models \langle\varrho\rangle \mathit{Last}$. □

The reverse direction also hold:

Theorem 11. *LDL_f can be translated into RE_f.*

It is possible to translate LDL_f directly into RE_f, via structural induction, but the direct translation is non-elementary, in general, since each occurrence of negation requires an exponential complementation construction. Below we demonstrate an elementary (doubly exponential) translation that proceeds via alternating automata.

Theorems 10, 11, and 4, allow us to characterize the expressive power of LDL_f.

Theorem 12. *LDL_f has exactly the same expressive power of* MSO.

For convenience we define an equivalent semantics for LDL_f, which we call *doubly-inductive semantics*. Its main characteristic is that it looks only at the current instant and at the next, and this will come handy in the next section. Specifically, for an LT_f-interpretation π, we inductively define when an LDL_f formula φ *is true* at an instant $i \in \{0, \ldots, \mathit{last}\}$, in symbols $\pi, i \models \varphi$, as follows:

- $\pi, i \models A$, for $A \in \mathcal{P}$ iff $A \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$
- $\pi, i \models \varphi \wedge \varphi'$ iff $\pi, i \models \varphi$ and $\pi, i \models \varphi'$
- $\pi, i \models \langle\phi\rangle\varphi$ iff $i < \mathit{last}$ and $\pi(i) \models \phi$ and $\pi, i+1 \models \varphi$ (ϕ propositional)
- $\pi, i \models \langle\psi?\rangle\varphi$ iff $\pi, i \models \psi \wedge \pi, i \models \varphi$
- $\pi, i \models \langle\rho_1 + \rho_2\rangle\varphi$ iff $\pi, i \models \langle\rho_1\rangle\varphi \vee \langle\rho_2\rangle\varphi$
- $\pi, i \models \langle\rho_1; \rho_2\rangle\varphi$ iff $\pi, i \models \langle\rho_1\rangle\langle\rho_2\rangle\varphi$
- $\pi, i \models \langle\rho^*\rangle\varphi$ iff $\pi, i \models \varphi$, or $i < \mathit{last}$ and $\pi, i \models \langle\rho\rangle\langle\rho^*\rangle\varphi$ and ρ is not *test-only*.

We say that ρ is *test-only* if it is a RE_f expression whose atoms are only tests $\psi?$.

Theorem 13. *The two semantics of LDL_f are equivalent.*

Proof. By mutual induction on the structure of the LDL_f formulas and the length of the LT_f-interpretation. □

6 LDL_f to AFW

Next we show how to reason in LDL_f. We do so by resorting to a direct translation of LDL_f formulas to alternating automata on words (AFW) [8; 12; 27]. We follow here the notation of [45]. Formally, an AFW on the alphabet $2^{\mathcal{P}}$ is a tuple $\mathcal{A} = (2^{\mathcal{P}}, Q, q_0, \delta, F)$, where Q is a finite nonempty set of states, q_0 is the initial state, F is a set of accepting states, and δ is a transition function $\delta : Q \times 2^{\mathcal{P}} \rightarrow B^+(Q)$, where $B^+(Q)$ is a set of positive boolean formulas whose atoms are states of Q. Given an input word $a_0, a_1, \ldots a_{n-1}$, a run of an AFW is a *tree* (rather than a sequence) labelled by states of the AFW such that *(i)* the root is labelled by q_0; *(ii)* if a node x at level i is labelled by a state q and $\delta(q, a_i) = \Theta$, then either Θ is true or some $P \subseteq Q$ satisfies Θ and x has a child for each element in P; *(iii)* the run is accepting if all leaves at depth n are labeled by states in F. Thus, a branch in an accepting run has to hit the true transition or hit an accepting state after reading all the input word $a_0, a_1, \ldots, a_{n-1}$.

Theorem 14 ([8; 12; 27]). AFW *are exactly as expressive as* RE_f.

It should be noted that while the translation from RE_f to AFW is linear, the translation from AFW to RE_f is doubly exponential. In particular every AFW can be translated into a standard nondeterministic finite automaton (NFA) that is exponentially larger than the AFW. Such a translation can be done on-the-fly while checking for nonemptiness of the NFA which, in turn, can be done in NLOGSPACE. Hence, we get the following complexity characterization for nonemptiness (the existence of a word that leads to acceptance) of AFW's.

Theorem 15 ([12]). *Nonemptiness for* AFW *is PSPACE-complete.*

We now show that we can associate with each LDL_f formula φ an AFW $\mathcal{A}_\varphi$ that accept exactly the traces that make φ true. The key idea in building the AFW $\mathcal{A}_\varphi$ is to use "subformulas" as states of the automaton and generate suitable transitions that mimic the doubly-inductive-semantics of such formulas. Actually we need a generalization of the notion of subformulas, which is known as the Fisher-Ladner closure, first introduced in the context of PDL [19]. The Fisher-Ladner closure CL_φ of an LDL_f formula φ is a set of LDL_f formulas inductively defined as follows:

$$
\begin{array}{l}
\varphi \in CL_\varphi\\
\neg\psi \in CL_\varphi \text{ if } \psi \in CL_\varphi \text{ and } \psi \text{ not of the form } \neg\psi'\\
\varphi_1 \wedge \varphi_2 \in CL_\varphi \text{ implies } \varphi_1, \varphi_2 \in CL_\varphi\\
\langle\rho\rangle\varphi \in CL_\varphi \text{ implies } \varphi \in CL_\varphi\\
\langle\phi\rangle\varphi \in CL_\varphi \text{ implies } \phi \in CL_\varphi \quad (\phi \text{ is propositional})\\
\langle\psi?\rangle\varphi \in CL_\varphi \text{ implies } \psi \in CL_\varphi\\
\langle\rho_1;\rho_2\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho_1\rangle\langle\rho_2\rangle\varphi \in CL_\varphi\\
\langle\rho_1+\rho_2\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho_1\rangle\varphi, \langle\rho_2\rangle\varphi \in CL_\varphi\\
\langle\rho^*\rangle\varphi \in CL_\varphi \text{ implies } \langle\rho\rangle\langle\rho^*\rangle\varphi \in CL_\varphi
\end{array}
$$

Observe that the cardinality of CL_φ is linear in the size of φ.

In order to proceed with the construction of the AFW $\mathcal{A}_\varphi$, we put LDL_f formulas φ in negation normal form $nnf(\varphi)$ by exploiting abbreviations and pushing negation inside as much as possible, leaving negations only in front of propositional symbols. Note that computing $nnf(\varphi)$ can be done in linear time. So, in the following, we restrict our attention to LDL_f formulas in negation normal form, i.e., LDL_f formulas formed according to the following syntax:

$$
\begin{array}{lcl}
\varphi & ::= & A \mid \neg A \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid \langle\rho\rangle\varphi \mid [\rho]\varphi\\
\rho & ::= & \phi \mid \varphi? \mid (\rho_1 + \rho_2) \mid (\rho_1;\rho_2) \mid (\rho^*).
\end{array}
$$

We assume that all the formulas in CL_φ are in negation normal form as well. Also, for convenience, we assume to have a special proposition *Last* which denotes the last element of the trace. Note that *Last* can be defined as: $Last \equiv [true]false$.

Then, we define the AFW $\mathcal{A}_\varphi$ associated with an LDL_f formula φ as $\mathcal{A}_\varphi = (2^{\mathcal{P}}, CL_\varphi, \varphi, \delta, \{\})$ where *(i)* $2^{\mathcal{P}}$ is the alphabet, *(ii)* CL_φ is the the state set, *(iii)* φ is the initial state *(iv)* δ is the transition function defined as follows:

$$
\begin{array}{lcl}
\delta(A,\Pi) & = & true \text{ if } A \in \Pi\\
\delta(A,\Pi) & = & false \text{ if } A \notin \Pi\\
\delta(\varphi_1 \wedge \varphi_2,\Pi) & = & \delta(\varphi_1,\Pi) \wedge \delta(\varphi_2,\Pi)\\
\delta(\varphi_1 \vee \varphi_2,\Pi) & = & \delta(\varphi_1,\Pi) \vee \delta(\varphi_2,\Pi)\\
\delta(\langle\phi\rangle\varphi,\Pi) & = & \begin{cases} \varphi \text{ if } \Pi \models \phi & (\phi \text{ propositional})\\ false \text{ if } \Pi \not\models \phi \end{cases}\\
\delta(\langle\psi?\rangle\varphi,\Pi) & = & \delta(\psi,\Pi) \wedge \delta(\varphi,\Pi)\\
\delta(\langle\rho_1+\rho_2\rangle\varphi,\Pi) & = & \delta(\langle\rho_1\rangle\varphi,\Pi) \vee \delta(\langle\rho_2\rangle\varphi,\Pi)\\
\delta(\langle\rho_1;\rho_2\rangle\varphi,\Pi) & = & \delta(\langle\rho_1\rangle\langle\rho_2\rangle\varphi,\Pi)\\
\delta(\langle\rho^*\rangle\varphi,\Pi) & = & \begin{cases} \delta(\varphi,\Pi) & \text{if } \rho \text{ is test-only}\\ \delta(\varphi,\Pi) \vee \delta(\langle\rho\rangle\langle\rho^*\rangle\varphi,\Pi) & \text{o/w} \end{cases}\\
\delta([\phi]\varphi,\Pi) & = & \begin{cases} \varphi \text{ if } \Pi \models \phi & (\phi \text{ propositional})\\ true \text{ if } \Pi \not\models \phi \end{cases}\\
\delta([\psi?]\varphi,\Pi) & = & \delta(nnf(\neg\psi),\Pi) \vee \delta(\varphi,\Pi)\\
\delta([\rho_1+\rho_2]\varphi,\Pi) & = & \delta([\rho_1]\varphi,\Pi) \wedge \delta([\rho_2]\varphi,\Pi)\\
\delta([\rho_1;\rho_2]\varphi,\Pi) & = & \delta([\rho_1][\rho_2]\varphi,\Pi)\\
\delta([\rho^*]\varphi,\Pi) & = & \begin{cases} \delta(\varphi,\Pi) & \text{if } \rho \text{ is test-only}\\ \delta(\varphi,\Pi) \wedge \delta([\rho][\rho^*]\varphi,\Pi) & \text{o/w} \end{cases}
\end{array}
$$

Theorem 16. *The state-size of $\mathcal{A}_\varphi$ is linear in the size of φ.*

Proof. Immediate, by inspecting the construction of $\mathcal{A}_\varphi$. □

Theorem 17. *Let φ be an LDL_f formula and $\mathcal{A}_\varphi$ the corresponding* AFW. *Then for every LT_f interpretation π we have that $\pi \models \varphi$ iff $\mathcal{A}_\varphi$ accepts π.*

Proof. By induction on the length of the LT_f interpretation π. We exploit the fact that the runs of $\mathcal{A}_\varphi$ over π follow closely the doubly-inductive semantics of the LDL formula φ. □

By Theorems 11 and 14 , we get LDL_f is exactly as expressive as RE_f and hence as MSO (cf. Theorem 12). Note that as the translation from LDL_f to AFW is linear, and the translation from AFW to RE_f is doubly exponential, the translation from LDL_f to RE_f is doubly exponential.

From Theorems 15 and 17 we finally get a complexity characterization of reasoning in LDL_f.

Theorem 18. *Satisfiability, validity, and logical implication for LDL_f formulas are PSPACE-complete.*

Proof. By Theorem 17, satisfiability of an LDL_f formula (to which validity and logical implication can be reduced) correspond to checking nonemptiness of the corresponding AFW, hence, by Theorems 15 and 16, we get the claim. □

7 Conclusion

In this paper we have analyzed LTL_f over finite traces, and devised a new logic LDL_f, which shares the naturalness and same computational properties of LTL_f, while being substantially more powerful. Although we do not detail it here, in LDL_f it is also possible to capture finite executions of programs expressed (in propositional variant, e.g., on finite object domains, of) high-level AI programming languages such as GOLOG [28], which also are used to constraint finite sequences [6]. We have focused on satisfiability, validity and logical implication, but analogous results are immediate for model checking as well: both LTL_f and LDL_f are PSPACE-complete with potential exponentiality depending only on the formula and not on the transition system be checked. As future work, we plan focus on automated synthesis [35], related to advanced forms of Planning in AI. Notice that, determinization, which is notoriously difficult step for synthesis in the infinite trace setting, becomes doable in practice in the finite trace setting. So, in principle, we can develop effective tools for unrestricted synthesis for LDL_f.

References

[1] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The forspec temporal logic: A new temporal property-specification language. In *TACAS*, 2002.

[2] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *AAAI*, 1996.

[3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000.

[4] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

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

[6] M. Bienvenu, C. Fritz, and S. A. McIlraith. Specifying and computing preferred plans. *Artif. Intell.*, 175(7-8):1308–1345, 2011.

[7] R. Bloem, A. Cimatti, I. Pill, M. Roveri, and S. Semprini. Symbolic implementation of alternating automata. *Implementation and Application of Automata*, LNCS 4094, 2006.

[8] J. A. Brzozowski and E. L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theor. Comput. Sci.*, 10:19–35, 1980.

[9] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik. Grund. Math.*, 6:66–92, 1960.

[10] T. Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

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

[12] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM (JACM)*, 28(1), Jan. 1981.

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

[14] V. Diekert and P. Gastin. First-order definable languages. In *Logic and automata: history and perspectives*. Amsterdam University Press, 2008.

[15] C. Eisner and D. Fisman. *A practical introduction to PSL*. Springer-Verlag New York Inc, 2006.

[16] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[17] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[18] P. Felli, G. De Giacomo, and A. Lomuscio. Synthesizing agent protocols from LTL specifications against multiple partially-observable environments. In *KR*, 2012.

[19] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18:194–211, 1979.

[20] A. Gabaldon. Precondition control and the progression algorithm. In *KR*, 2004.

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

[22] D. Harel. Dynamic logic. In *Handbook of Philosophical Logic*, D. Reidel Publishing Company, 1984.

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

[24] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

[25] H. Kamp. *On tense logic and the theory of order*. PhD thesis, UCLA, 1968.

[26] B. Khoussainov and A. Nerode. *Automata theory and its applications*. Birkhauser, 2001.

[27] E. L. Leiss. Succint representation of regular languages by boolean automata. *Theor. Comput. Sci.*, 13:323–330, 1981.

[28] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31:59–84, 1997.

[29] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of aritificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[30] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

[31] F. Patrizi, N. Lipovetzky, G. D. Giacomo, and H. Geffner. Computing infinite plans for LTL goals using a classical planner. In *IJCAI*, 2011.

[32] M. Pešić, D. Bošnački, and W. van der Aalst. Enacting declarative languages using LTL: avoiding errors and improving performance. *Model Checking Software*, 2010.

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

[34] A. Pnueli, O. Lichtenstein, and L. Zuck. The Glory of The Past. *Logics of Programs: Brooklyn, June 17-19, 1985*, 193:194, 1985.

[35] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

[36] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[37] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985.

[38] S. Sohrabi, J. A. Baier, and S. A. McIlraith. Preferred explanations: Theory and generation via planning. In *AAAI*, 2011.

[39] L. J. Stockmeyer and A. Meyer. Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM*, 49(6):753–784, 2002.

[40] W. Thomas. Star-free regular sets of ω-sequences. *Information and Control*, 42(2):148–156, 1979.

[41] W. Thomas. Languages, Automata, and Logic. *Handbook of formal languages: beyond words*, 1997.

[42] B. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:101–131, 1962. Russian; English translation in: AMS Transl. 59 (1966), 23-55.

[43] W. M. P. van der Aalst, M. Pešić, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, Mar. 2009.

[44] M. Vardi. The rise and fall of linear time logic In *2nd Int'l Symp. on Games, Automata, Logics and Formal Verification*, 2011.

[45] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, 1996.

[46] T. Wilke. Classifying discrete temporal properties. *STACS*,1999.

[47] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.