Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness

Giuseppe De Giacomo
Riccardo De Masellis
Dip. di Ing. Informatica, Automatica e Gestionale
Sapienza Università di Roma, Italy
{degiacomo, demassellis}@dis.uniroma1.it

Marco Montali
KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy
montali@inf.unibz.it

Abstract
In this paper we study when an LTL formula on finite traces (LTL_f formula) is insensitive to infiniteness, that is, it can be correctly handled as a formula on infinite traces under the assumption that at a certain point the infinite trace starts repeating an end event forever, trivializing all other propositions to false. This intuition has been put forward and (wrongly) assumed to hold in general in the literature. We define a necessary and sufficient condition to characterize whether an LTL_f formula is insensitive to infiniteness, which can be automatically checked by any LTL reasoner. Then, we show that typical LTL_f specification patterns used in process and service modeling in CS, as well as trajectory constraints in Planning and transition-based LTL_f specifications of action domains in KR, are indeed very often insensitive to infiniteness. This may help to explain why the assumption of interpreting LTL on finite and on infinite traces has been (wrongly) blurring. Possibly because of this blurring, virtually all literature detours to Büchi automata. We give a simple direct algorithm for computing such NFA.

1 Introduction
LTL on finite traces, here called LTL_f as in (De Giacomo and Vardi 2013), has been extensively used in AI. For example, it is at the base of trajectory constraints in Planning in PDDL 3.0 (Bacchus and Kanbanza 2000; Gerevini et al. 2009), the de-facto standard formalism for representing planning problems. Notably, LTL_f is recently gaining momentum in CS as a declarative way to specify (terminating) services and processes (Pesc and van der Aalst 2006; Montali et al. 2010; Sun, Xu, and Su 2012). We will collectively refer here to this literature as the DECLARE approach, after the main system in that area (Pesc, Schonenberg, and van der Aalst 2007).

The presence of a big body of work in LTL on infinite traces (Gabbay et al. 1980; Vardi 1996; Holzmann 1995), leads researchers to “hack” it for dealing with finite traces as well, “blurring” the distinction between the two settings. For example, both the declarative patterns for processes and services, widely adopted in the DECLARE approach (Pesc and van der Aalst 2006; Montali et al. 2010) are directly inspired by a catalogue of temporal logic patterns developed for LTL on infinite traces (Dwyer, Avrunin, and Corbett 1999).

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

as the trajectory constraints in PDDL 3.0 are. As another example, in (Edelkamp 2006) it is proposed to directly use Büchi automata, capturing LTL on infinite traces, for LTL_f, saying: “[...we can cast the Büchi automaton as an NFA (nondeterministic finite automaton, ed.), which accepts a word (i.e., trace ed.) if it terminates in a final state.” Then in (Gerevini et al. 2009) this is taken up, saying: “Since PDDL 3.0 constraints are normally evaluated over finite trajectories, the Büchi acceptance condition, that “an accepting state is visited infinitely often”, reduces to the standard acceptance condition that the automaton is in an accepting state at the end of the trajectory.” (Notice: this is incorrect if one simply leaves as accepting states those of the Büchi automaton.)

In (van der Aalst and Pesic 2006) the authors gave a quite appealing, but unfortunately incorrect in general, intuition for the blurring: “[...] we use the original algorithm for the generation of (Büchi, ed.) automata, but we slightly change the DecSerFlow (i.e., DECLARE, ed.) model before creating the automaton. To be able to check if a finite trace is accepting, we add one “invisible” activity and one “invisible” constraint to every DecSerFlow model and then construct the automaton. With this we specify that each execution of the model will eventually end. We introduce an “invisible” activity e, which represents the ending activity in the model. We use this activity to specify that the service will end - the termination constraint. This constraint has the LTL formula $\Diamond e \land \Box (e \rightarrow \varphi(e))$. In DECLARE it is assumed that only one activity can happen (i.e., only a proposition is true) at every time point, so the presence of the “e(nd)” activity above implies that all other propositions trivialize to false.

In fact, the two variants of LTL on finite and infinite traces are quite different, as discussed, e.g., in (Bauer and McIlraith 2006; De Giacomo and Vardi 2013). Why, can the research community live up with this blurring between finite and infinite traces? We help to answer this question in this paper by showing that the intuition in (van der Aalst and Pesic 2006) reported above is surprisingly correct over several widely used formulas. Specifically, we define the notion of insensitivity to infiniteness for an LTL_f formula, which captures exactly the intuition in (van der Aalst and Pesic 2006)\footnote{This is an alternative approach to Bauer and Haslum 2010, where correctness conditions are considered in extending finite traces by repeating at infinitum the propositional assignment in the last element of the finite trace. While this choice is suitable when...}.\footnote{This is an alternative approach to Bauer and Haslum 2010, where correctness conditions are considered in extending finite traces by repeating at infinitum the propositional assignment in the last element of the finite trace. While this choice is suitable when...}
We, then, define a necessary and sufficient condition, which can be automatically checked by any LTL reasoner, to verify whether an LTL formula is insensitive to infiniteness. Using such a condition, we show that all LTL formulas corresponding to the \texttt{DECLARE} patterns but one, are indeed insensitive to infiniteness. We also show that virtually all transition-based specifications of action domains expressed in LTL are insensitive to infiniteness, and that most PDDL 3.0 trajectory constraints can be easily adjusted to meet this property. Possibly because of the blurring between finite and infinite traces, virtually all literature in AI and CS detours to Büchi automata for building the NFA that accepts the traces satisfying an LTL formula (Giannakopoulou and Havelund 2001; Edelkamp 2006; Bauer and McIraith 2006; Bauer, Katoen, and Guldstrand Larsen 2008; Bauer and Haslum 2010; Westergaard 2011). As a further contribution, we give a simple direct algorithm for computing such NFA.

2 LTL$_f$: LTL on Finite Traces

LTL$_f$ (De Giacomo and Vardi 2013) uses the same syntax of the original LTL (Pnueli 1977). Formulas of LTL$_f$ are built from a set $P$ of propositional symbols and are closed under the boolean connectives, the unary temporal operator $\circ$ (next-time) and the binary temporal operator $U$ (until):

\[ \varphi ::= a | \neg \varphi | \varphi_1 \land \varphi_2 | \circ \varphi | \varphi_1 U \varphi_2 \quad \text{with } a \in P \]

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

- Standard boolean abbreviations, such as $true$, $false$, $\vee$, $\rightarrow$.
- $last = \neg true$ denotes the last instant of the sequence.

Over infinite traces it corresponds to $\lozenge false$ and is indeed always false, while in LTL$_f$ it becomes true at the last instance of the sequence.

- $\lozenge \varphi = \neg \circ false$ is interpreted as a weak next, stating that if $last$ does not hold then $\varphi$ must hold in the next state.
- $\lozenge \varphi = true U \varphi$ says that $\varphi$ will eventually hold before the last instant (included).
- $\square \varphi = \neg \lozenge \varphi$ says that from the current instant till the last instant $\varphi$ will always hold.
- $\varphi_1 R \varphi_2 = \neg (\neg \varphi_1 U \neg \varphi_2)$ means that $\varphi_1$ releases $\varphi_2$, i.e., either $\varphi_2$ must hold forever, or until $\varphi_1$ also holds.
- $\varphi_1 W \varphi_2 = (\varphi_1 U \varphi_2 \lor \square \varphi_1)$ is interpreted as a weak until, and means that $\varphi_1$ holds until $\varphi_2$ or forever.

The semantics of LTL$_f$ is given in terms of finite traces denoting a finite sequence of consecutive instants of time, i.e., finite words $\pi$ over the alphabet of $2^P$, containing all possible interpretations of the propositional symbols in $P$. Given a finite trace $\pi$, we inductively define when an LTL$_f$ formula $\varphi$ is true at an instant $i$ (for $0 \leq i \leq n$), written $\pi, i \models \varphi$, as:

- $\pi, i \models a$, for $a \in P$ iff $a \in \pi(i)$.
- $\pi, i \models \neg \varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi_1 \land \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$.
- $\pi, i \models \lozenge \varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi_1 U \varphi_2$ iff for some $j$ s.t. $i \leq j \leq n$, we have $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$, we have $\pi, k \not\models \varphi_1$.

Propositions are used to describe the domain states, ours is more natural when propositions denote actions.

A formula $\varphi$ is true in $\pi$, in notation $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula $\varphi$ is satisfiable if it is true in some finite trace, and it is valid if it is true in every finite trace. A formula $\varphi$ logically implies a formula $\varphi'$, written $\varphi \models \varphi'$, if for every finite trace $\pi$ we have that $\pi \models \varphi$ implies $\pi \models \varphi'$. Notice that satisfiability, validity and logical implication are all mutually reducible to each other: for example $\varphi$ is valid iff $\neg \varphi$ is unsatisfiable. Similarly, $\varphi \models \varphi'$ iff $\varphi \land \neg \varphi'$ is unsatisfiable.

**Theorem 1.** (De Giacomo and Vardi 2013) Satisfiability (hence validity and logical implication) for LTL$_f$ formulas is PSPACE-complete.

We observe that LTL on infinite traces and LTL$_f$ are quite different. E.g., the formula

$$\Diamond a \land \Box (\Diamond b \land \Box (b \rightarrow \Diamond a) \land \Box (\neg a \lor \neg b))$$

is unsatisfiable in LTL$_f$ but is satisfiable in LTL. In other words, in a finite trace setting $\Diamond a \land \Box (\Diamond b \land \Box (b \rightarrow \Diamond a) \land \Box (\neg a \lor \neg b))$ implies that eventually both $a$ and $b$ are going to be simultaneously true. Interestingly, the NFA for (1) on finite traces and the Büchi automaton for the same formula on infinite traces are radically different. In fact, the NFA recognizes nothing (cf. Figure 1b), while the Büchi automaton is shown in Figure 1a. Certainly, one cannot consider such Büchi automaton as a correct NFA for the formula on finite traces by simply considering the accepting states as final.

3 Insensitivity to Infiniteness

Often the distinction between interpreting LTL formulas over finite vs. infinite traces is blurred via some hacking. In this section we want to tackle this issue in a precise way.

One can reduce LTL$_f$ into LTL (on infinite traces), while preserving all standard reasoning task, such as satisfiability, validity, etc. In particular, given an LTL$_f$ formula $\varphi$, we can construct a corresponding LTL formula, as follows: (i) introduce a fresh proposition “end” to denote that the trace is ended (note that $\text{end} \not\in P$, and that $last$ is true just before the first occurrence of $\text{end}$); (ii) require that $\text{end}$ eventually holds ($\Diamond \text{end}$); (iii) require that once $\text{end}$ becomes true it stays true forever ($\Box (\text{end} \rightarrow \Diamond \text{end})$); (iv) require that when $\text{end}$ is true, all other propositions are reset to false ($\Box (\text{end} \rightarrow \bigwedge_{a \in P} \neg a)$); (v) translate the LTL$_f$ formula into an LTL formula as follows:

$$f(a) \rightarrow a \quad f(\varphi_1 U \varphi_2) \rightarrow f(\varphi_1) \land f(\varphi_2) \land \neg \text{end}$$
$$f(\neg \varphi) \rightarrow \neg f(\varphi) \quad f(\Box \varphi) \rightarrow \Diamond f(\varphi) \land \neg \text{end}$$
$$f(\varphi_1 \land \varphi_2) \rightarrow f(\varphi_1) \land f(\varphi_2) \quad f(\Diamond \varphi) \rightarrow \Box (f(\varphi) \lor \text{end})$$
$$f(\varphi) \rightarrow f(\varphi) \land \Diamond (f(\varphi) \land \text{end}) \quad f(\Diamond \varphi) \rightarrow \Diamond (f(\varphi) \lor \text{end})$$

**Theorem 2.** Let $\pi$, be an infinite trace. Then

$$\pi \models \Diamond \text{end} \land \Box (\Diamond \text{end} \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} \neg a))$$

iff it is the form $\pi = \pi_1 \{\text{end}\}^\omega$, where $\text{end}$ is always false in $\pi_1$.  

![Figure 1: Automata for formula (1)](image-url)
Proof (sketch). The only if direction is immediate. For the if direction, suffice it to observe that if \( \pi_i \) satisfies \( \diamond \text{end} \land \Box (\text{end} \rightarrow \diamond \text{end}) \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} -a) \) then there must be a first instant in which \( \text{end} \) becomes true, hence \( \pi_i \) must have the form \( \pi_f \{ \text{end} \}^\omega \) where \( \text{end} \) is always false in \( \pi_f \).

**Theorem 3.** Let \( \varphi \) be an \( LTL_f \) formula and \( \pi_i = \pi_f \{ \text{end} \}^\omega \) an infinite trace where \( \text{end} \) is always false in \( \pi_f \). Then

\[
\pi_f \models \varphi \iff \pi_f \{ \text{end} \}^\omega \models f(\varphi).
\]

Proof (sketch). Both directions can be shown by induction on the structure of the formula \( \varphi \).

We now exploit the formal notions behind the above two theorems to define the notion of insensitivity to infiniteness, capturing the intuition discussed in the introduction.

**Definition 1.** An \( LTL_f \) formula \( \varphi \) is insensitive to infiniteness if for every (infinite) trace \( \pi_i = \pi_f \{ \text{end} \}^\omega \) where \( \text{end} \) is always false in \( \pi_f \), we have that

\[
\pi_f \models \varphi \iff \pi_f \{ \text{end} \}^\omega \models \varphi.
\]

\( LTL_f \) formulas that are insensitive to infiniteness can be translated into \( LTL \) by simply adding the conditions on \( \text{end} \) without applying the translation function \( f(\cdot) \). Notice that if an \( LTL_f \) formula is insensitive to infiniteness, we can essentially blur the distinction between finite and infinite traces by simply asserting in the infinite case that there exists an \( \text{end} \) of the significant part and that once such \( \text{end} \) is reached every proposition is trivially reset to false in the infinite trace. Next theorem gives us necessary and sufficient conditions for an \( LTL_f \) formula to be insensitive to infiniteness.

**Theorem 4.** An \( LTL_f \) \( \varphi \) is insensitive to infiniteness if and only if the following \( LTL_f \) valid:

\[
(\diamond \text{end} \land \Box (\text{end} \rightarrow \Diamond \text{end}) \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} -a)) \rightarrow (\varphi \equiv f(\varphi))
\]

Proof. (If direction.) By Theorem 3 we know that for every infinite trace satisfying the premise of the implication must have the form \( \pi_i = \pi_f \{ \text{end} \}^\omega \) where \( \text{end} \) is always false in \( \pi_f \). While by Theorem 2 \( \pi_f \models \varphi \) if and only if \( \pi_f \{ \text{end} \}^\omega \models f(\varphi) \). But then by the consequent of the implication we have that \( \pi_f \{ \text{end} \}^\omega \models \varphi \), hence \( \varphi \) is insensitive to infiniteness.

(Only if direction.) Since \( \varphi \) is insensitive to infiniteness, we have that for every (infinite) trace \( \pi_i = \pi_f \{ \text{end} \}^\omega \) where \( \text{end} \) is always false in \( \pi_f \); \( \pi_f \models \varphi \) if and only if \( \pi_f \{ \text{end} \}^\omega \models f(\varphi) \). On the other hand, by Theorem 3 we have that \( \pi_f \models \varphi \) if and only if \( \pi_f \{ \text{end} \}^\omega \models f(\varphi) \). By combining the two above equivalences we get \( \pi_f \{ \text{end} \}^\omega \models f(\varphi) \), which in turn implies \( \pi_f \{ \text{end} \}^\omega \models \varphi \equiv f(\varphi) \). Now by Theorem 2 we have that an infinite trace has the form \( \pi_i = \pi_f \{ \text{end} \}^\omega \) if and only if \( \pi_i \models \text{Diamond}(\text{end} \land \Box (\text{end} \rightarrow \Diamond \text{end}) \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} -a)) \). Hence, we get \( \pi_i \models (\Diamond (\text{end} \rightarrow \Diamond \text{end}) \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} -a)) \) implies \( \pi_i \models \varphi \equiv f(\varphi) \), which is the claim.

This theorem is quite interesting since it gives us a technique to check an \( LTL_f \) formula for insensitivity to the infiniteness: we simply need to check the standard \( LTL \) formula \( \Diamond (\text{end} \land \Box (\text{end} \rightarrow \Diamond \text{end}) \land \Box (\text{end} \rightarrow \bigwedge_{a \in P} -a)) \rightarrow (\varphi \equiv f(\varphi)) \) for validity, or its negation for unsatisfiability, which can be done by checking for emptiness the corresponding Büchi automata, see e.g., [Vardi 1996]. For example, one can check that the \( LTL_f \) formula \( (1) \) is insensitive to infiniteness.

We close the section by showing that the class of \( LTL_f \) formulas that are insensitive to infiniteness is closed under Boolean operations.

**Theorem 5.** Let \( \varphi_1 \) and \( \varphi_2 \) be two \( LTL_f \) formulas that are insensitive to infiniteness. Then the \( LTL_f \) formulas \( \lnot \varphi_1 \) and \( \varphi_1 \land \varphi_2 \) are also insensitive to infiniteness.

Proof. By induction on the structure of the formula, considering the definition of insensitive to infiniteness.

\[\Boxend\]
Table 1: Declare patterns and their insensitivity to infiniteness

<table>
<thead>
<tr>
<th>NAME</th>
<th>NOTATION</th>
<th>LTL$_f$ FORMALIZATION</th>
<th>DESCRIPTION</th>
<th>INSENSITIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existence</td>
<td>a</td>
<td>$\Diamond a$</td>
<td>$a$ must be executed at least once</td>
<td>Y</td>
</tr>
<tr>
<td>Absence 2</td>
<td>a</td>
<td>$\neg \Diamond (a \land \Diamond a)$</td>
<td>$a$ can be executed at most once</td>
<td>Y</td>
</tr>
<tr>
<td>Choice</td>
<td>a -- b</td>
<td>$\Diamond a \lor \Diamond b$</td>
<td>$a$ or $b$ must be executed</td>
<td>Y</td>
</tr>
<tr>
<td>Exclusive Choice</td>
<td>a -- b</td>
<td>$(\Diamond a \lor \Diamond b) \land \neg (\Diamond a \land \Diamond b)$</td>
<td>Either $a$ or $b$ must be executed, but not both</td>
<td>Y</td>
</tr>
<tr>
<td>Resp. existence</td>
<td>a -- b</td>
<td>$\Diamond a \rightarrow \Diamond b$</td>
<td>If $a$ is executed, then $b$ must be executed as well</td>
<td>Y</td>
</tr>
<tr>
<td>Coexistence</td>
<td>a -- b</td>
<td>$(\Diamond a \rightarrow \Diamond b) \land (\Diamond b \rightarrow \Diamond a)$</td>
<td>Either $a$ and $b$ are both executed, or none of them is executed</td>
<td>Y</td>
</tr>
<tr>
<td>Response</td>
<td>a -- b</td>
<td>$\Box (a \rightarrow \Diamond b)$</td>
<td>Every time $a$ is executed, $b$ must be executed afterwards</td>
<td>Y</td>
</tr>
<tr>
<td>Precedence</td>
<td>a -- b</td>
<td>$a \lor \Diamond b$</td>
<td>$b$ can be executed only if $a$ has been executed before</td>
<td>Y</td>
</tr>
<tr>
<td>Succession</td>
<td>a -- b</td>
<td>$\Box (a \rightarrow \Diamond b) \land (\Diamond b \lor \Diamond a)$</td>
<td>$b$ must be executed after $a$, and $a$ must precede $b$</td>
<td>Y</td>
</tr>
<tr>
<td>Alt. Response</td>
<td>a -- b</td>
<td>$\Box(a \rightarrow (\neg \Diamond f b))$</td>
<td>Every $a$ may be followed by $b$, without any other $a$ between $b$</td>
<td>Y</td>
</tr>
<tr>
<td>Alt. Precedence</td>
<td>a -- b</td>
<td>$\Box (\neg b \lor a) \land (\Box (\neg b \lor a))$</td>
<td>Every $b$ must be preceded by $a$, without any other $b$ between $a$</td>
<td>Y</td>
</tr>
<tr>
<td>Alt. Succession</td>
<td>a -- b</td>
<td>$\Box (\neg b \lor a) \land (\Box (\neg b \lor a))$</td>
<td>Combination of alternate response and alternate precedence</td>
<td>Y</td>
</tr>
<tr>
<td>Chain Response</td>
<td>a -- b</td>
<td>$\Box (a \equiv \Diamond b)$</td>
<td>If $a$ is executed then $b$ must be executed next</td>
<td>Y</td>
</tr>
<tr>
<td>Chain Precedence</td>
<td>a -- b</td>
<td>$\Box (\Diamond b \rightarrow a)$</td>
<td>Task $b$ can be executed only immediately after $a$</td>
<td>Y</td>
</tr>
<tr>
<td>Chain Succession</td>
<td>a -- b</td>
<td>$\Box (a \equiv \Diamond b)$</td>
<td>Tasks $a$ and $b$ must be executed next to each other</td>
<td>Y</td>
</tr>
<tr>
<td>Not Coexistence</td>
<td>a -- b</td>
<td>$\neg (\Diamond a \land \Diamond b)$</td>
<td>Only one among tasks $a$ and $b$ can be executed, but not both</td>
<td>Y</td>
</tr>
<tr>
<td>Neg. Succession</td>
<td>a -- b</td>
<td>$\Box (a \rightarrow \neg b)$</td>
<td>Task $a$ cannot be followed by $b$, and $b$ cannot be preceded by $a$</td>
<td>N</td>
</tr>
<tr>
<td>Neg. Chain Succession</td>
<td>a -- b</td>
<td>$\Box(a \equiv \neg b)$</td>
<td>Tasks $a$ and $b$ cannot be executed next to each other</td>
<td>N</td>
</tr>
</tbody>
</table>

The theorem can be proven automatically, making use of an LTL reasoner on infinite traces. Specifically, each DECLARE pattern can be grounded on a concrete set of tasks (propositions), and then, by Theorem 4, we simply need to check the validity of the corresponding formula. In fact, we encoded each validity check in the model checker NuSMV following the approach of satisfiability via model checking (Rozier and Vardi 2007). E.g., the following NuSMV specification checks whether response is insensitive to infiniteness:

```
MODULE main
VAR a: boolean; b: boolean; other: boolean; end: boolean;
LTLSPEC
G\end (a \rightarrow X (\neg b \land \Box a))
-> ( (G (a \rightarrow X (F(b)))) \landend )

NuSMV confirmed that all patterns but the negation chain succession are insensitive to infiniteness. This is true both making or not the DECLARE assumption, and independently whether $P$ only contains the propositions explicitly mentioned in the pattern, or also further ones.

Let us discuss the negation chain succession, which is not insensitive to infiniteness. On infinite traces, $\Box(a \equiv \neg b)$ retains the meaning specified in Table 1. On finite traces, it also forbids $a$ to be the last-executed task in the finite trace, since it requires $a$ to be followed by another task that is different.

2 The full list of specifications is available here: http://www.inf.unibz.it/ montali/AAAI14
from b. E.g., we have that $\{a\}\{\text{end}\}^\omega \models \Box(a \equiv \neg b)$, but $\{a\} \not\models (a \equiv \neg b)$. This is not foreseen in the informal description present in all papers about DECLARE, and shows the subtlety of directly adopting formulas originally devised in the infinite-trace setting to the one of finite traces. In fact, the same meaning is retained only for those formulas that are insensitive to infiniteness. Notice that the correct way of formalizing the intended meaning of negation chain succession on finite traces is $\Box(a \equiv \neg b)$ (that is, $\Box(a \equiv \neg c)$). This is equivalent to the other formulation in the infinite-trace setting, and actually it is insensitive to infiniteness.

Notice that there are several other DECLARE constraints, beyond standard patterns, that are not insensitive to infiniteness, such as $\Box a$. Over infinite traces, $\Box a$ states that a must be executed forever, whereas, on finite traces, it obviously stops requiring $a$ when the trace ends.

5 Action Domains and Trajectories

We often characterize an action domain by the set of actions, each represented as a sequence of situations (Reiter 2001). To do so, we typically introduce a set of atomic facts, called fluents, whose truth value changes as the system evolves from one situation to the next because of actions. Since LTL/PLTL do not provide a direct notion of action, we use propositions to denote them, as in (Calvanese, De Giacomo, and Vardi 2002). Hence, we partition $P$ into fluents $F$ and actions $A$, adding structural constraint (analogous to the DECLARE assumption) such as $\Box(\forall a \in A \, a) \land \Box(\forall a \in A \, a \land a \land a \land a \land a)$, to specify that one action must be performed to get to a new situation, and that a single action at a time can be performed. Then, the initial situation is described by a propositional formula $\varphi_{\text{init}}$ involving only fluents, while effects can be modelled as:

$$\Box(\varphi \rightarrow \Box(a \rightarrow \psi))$$

(2)

where $a \in A$, while $\psi$ and $\varphi$ are arbitrary propositional formulas involving only fluents. Such a formula states that performing action $a$ under the conditions denoted by $\varphi$ brings about the conditions denoted by $\psi$.

Alternatively, we can formalize effects through Reiter’s successor state axioms (Reiter 2001) (which also provide a solution to the frame problem), as in (Calvanese, De Giacomo, and Vardi 2002) De Giacomo and Vardi 2013), by translating the (instantiated) successor state axiom $F(d(o(a, s))) \equiv \varphi^+(s) \lor (F(s) \land \neg \varphi^-(s))$ into the LTL$_f$ formula:

$$\Box(o(a \rightarrow \Box(F \equiv \varphi^+ \lor (F \land \neg \varphi^-))).$$

(3)

In general, to specify effects we need special LTL$_f$ formulas that talk only about the current state and the next state to capture how the domain does a transition from the current to the next state. Such formulas are called transition formula, and are inductively built as follows:

$$\varphi ::= \psi \lor \Box \varphi \lor \varphi \land \varphi \land \varphi$$

where $\psi$ is propositional.

For such formulas we can state a notable result: under the assumption that at every step at least one proposition is true, every specification based on transition formulas is insensitive to infiniteness. More precisely:

$^1$A formula like $\Box(\varphi \rightarrow \Box(a \rightarrow \varphi))$ corresponds to a frame axiom expressing that $\varphi$ does not change when performing $a$.

Theorem 7. Let $\varphi$ be an LTL$_f$ transition formula and $P$ any non-empty subset of $P$. Then all LTL$_f$ formulas of the form $\Box(\biglor_{a \in P} a \rightarrow \varphi)$ are insensitive to infiniteness.

Proof. Suppose not. Then there exists a finite trace $\pi_f$ and a formula $\Box(\biglor_{a \in P} a \rightarrow \varphi)$ such that $\pi_f \models \Box(\biglor_{a \in P} P \rightarrow \varphi)$, but $\pi_f \{\text{end}\}^\omega \not\models \Box(\biglor_{a \in P} P \rightarrow \varphi)$. Hence, $\pi_f \{\text{end}\}^\omega \models \Box(\biglor_{a \in P} P \land \neg \varphi)$. That is there exist a point $i$ in the trace $\pi_f \{\text{end}\}^\omega$ such that $\pi_f \{\text{end}\}^\omega, i \models \Box(\biglor_{a \in P} P \land \neg \varphi)$. Now observe that $i$ can only be in $\{\text{end}\}^\omega$ since in the $\{\text{end}\}^\omega$ part $\Box(\biglor_{a \in P} P$ is false. But then $\pi_f \not\models \Box(\biglor_{a \in P} P \rightarrow \varphi)$ contradicting the assumption.

By applying the above theorem we can immediately show that $\Box(\varphi)$ (for the latter, noting that it is equivalent to $\Box(o(a \rightarrow (\varphi \land \Box))$) are insensitive to infiniteness.

Also PDDL action effects (McDermott et al. 1998) can be encoded in LTL$_f$, and show to be insensitive to infiniteness using the above theorem. Here, however, we focus on PDDL

3.0 trajectory constraints (Gerevini et al. 2009):

$$(\text{at end } \varphi) ::= \text{last } \land \varphi$$

(always $\varphi) ::= \Box \varphi$$

(sometimes $\varphi) ::= \Box \varphi$$

(within $n \varphi) ::= \biglor_{0 \leq i \leq n} \Box \varphi$$

(hold-after $n \varphi) ::= \Box \varphi$$

(hold-before $n \varphi) ::= \biglor_{0 \leq i \leq n} \Box \varphi$$

(sometime-before $n \varphi) ::= \biglor_{0 \leq i \leq n} \Box \varphi$$

(always-within $n \varphi) ::= \biglor_{0 \leq i \leq n} \Box \varphi$$

(never-instantiation $n \varphi) ::= \biglor_{0 \leq i \leq n} \Box \varphi$$

where $\varphi$ is a propositional formula on fluents, called goal formula. Most trajectory constraints are (variants) of DECLARE patterns, and we can ask if they are insensitive to infiniteness using Theorem 7. Moreover, the following general result holds. Let a goal formula be guarded when it is equivalent to $\biglor_{F \in F} F \land \varphi$ with $\varphi$ any propositional formula. Then:

Theorem 8. All trajectory constraints involving only guarded goal formulas, except from (always $\varphi$, are insensitive to infiniteness.

6 Reasoning in LTL$_f$ through NFAs

We can associate with each LTL$_f$ formula $\varphi$ an (exponential) NFA $A_{\varphi}$ that accepts exactly the traces that make $\varphi$ true. Various techniques for building such NFAs have been proposed in the literature, but they all require a detour to automata on infinite traces first. In (Bauer, Leucker, and Schallhart 2007) NFAs are used to check the compliance of an evolving trace to a formula expressed in LTL. The automaton construction is grounded on the one in (Lichtenstein, Pnueli, and Zuck 1985), which, by introducing past operators, focuses on finite traces. The procedure builds an NFA that recognizes both finite and infinite traces satisfying the formula. Such an automaton is indeed very similar to a generalized Büchi automaton (cf. the Büchi automaton construction for LTL formulas in (Baier, Katoen, and Geldesbrand Larsen 2008)).
As explained in (Westergaard 2011), the DECLARE environment uses the automaton construction in (Giannakopoulou and Havelund 2001), which applies the traditional Büchi automaton construction in (Gerth et al. 1995), and then suitably defines which states have to be considered as final. The language, however, does not include the next operator. Inspired by (Giannakopoulou and Havelund 2001), also the approach in (Bauer and McIraith 2006) relies on the procedure in (Gerth et al. 1995) to build the NFA, but it implements the full LTL\textsubscript{f} semantics by dealing also with the next operator.

Here, we provide a simple direct algorithm for computing the NFA corresponding to an LTL\textsubscript{f} formula. The correctness of the algorithm is based on the fact that (i) we can associate with each LTL\textsubscript{f} formula \( \varphi \) a polynomial alternating automaton on words (AFW) \( A_\varphi \) that accept exactly the traces that make \( \varphi \) true (De Giacomo and Vardi 2013), and (ii) every AFW can be transformed into an NFA, see, e.g., (De Giacomo and Vardi 2013). However, to formulate the algorithm we do not need these notions, but we can work directly on the LTL\textsubscript{f} formula. We assume our formula to be in negation normal form, by exploiting abbreviations and pushing negation inside as much as possible, leaving it only in front of propositions (any LTL\textsubscript{f} formula can be transformed into negation normal form in linear time). We also assume \( \mathcal{P} \) to include a special proposition \( \text{last} \) which denotes the last element of the trace. Note that \( \text{last} \) can be defined as \( \text{last} \equiv \bullet \text{false} \). Then we define an auxiliary function \( \delta \) that takes an LTL\textsubscript{f} formula \( \psi \) (in negation normal form) and a propositional interpretation \( \Pi \) for \( \mathcal{P} \) (including \( \text{last} \)), returning a positive boolean formula whose atoms are (quoted) \( \psi \) subformulas.

\[
\begin{align*}
\delta(&\bullet \varphi, \Pi) = \text{true} \quad \text{if} \ a \in \Pi \\
\delta(&\text{false}, \Pi) = \text{false} \quad \text{if} \ a \notin \Pi \\
\delta(&\neg \varphi, \Pi) = \text{false} \quad \text{if} \ a \notin \Pi \\
\delta(&\varphi_1 \land \varphi_2, \Pi) = \delta(\varphi_1, \Pi) \land \delta(\varphi_2, \Pi) \\
\delta(&\varphi_1 \lor \varphi_2, \Pi) = \delta(\varphi_1, \Pi) \lor \delta(\varphi_2, \Pi) \\
\delta(&\neg \varphi, \Pi) = \neg \delta(\varphi, \Pi) \\
\delta(&\bullet \varphi, \Pi) = \delta(\varphi, \Pi) \\
\delta(&\varphi_1 \land \varphi_2, \Pi) = \delta(\varphi_1, \Pi) \land \delta(\varphi_2, \Pi) \\
\delta(&\text{false}, \Pi) = \text{false} \\
\delta(&\varphi_1 \lor \varphi_2, \Pi) = \delta(\varphi_1, \Pi) \lor \delta(\varphi_2, \Pi) \\
\delta(&\varphi, \Pi) = \varphi \\
\delta(\Pi) = \Pi \\
\delta(\text{false}, \Pi) = \text{false} \\
\delta(\varphi_1 \land \varphi_2, \Pi) = \delta(\varphi_1, \Pi) \land \delta(\varphi_2, \Pi) \\
\delta(\Pi) = \Pi
\end{align*}
\]

Using function \( \delta \) we can build the NFA \( A_\varphi \) of an LTL\textsubscript{f} formula \( \varphi \) in a forward fashion. States of \( A_\varphi \) are sets of atoms (recall that each atom is quoted \( \psi \) subformulas) to be interpreted as a conjunction; the empty conjunction \( \emptyset \) stands for \text{true}.

1: algorithm LTL\textsubscript{f}/2NFA() 2: input LTL\textsubscript{f} formula \( \varphi \) 3: output NFA \( A_\varphi = (2^\mathbb{P}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\}) \) 4: \( s_0 \leftarrow \{\varphi\} \) \( \triangleright \) single initial state 5: \( s_f \leftarrow \emptyset \) \( \triangleright \) single final state 6: \( \mathcal{S} \leftarrow \{s_0, s_f\}, \varrho \leftarrow \emptyset \) 7: while (\( \mathcal{S} \) or \( \varrho \) change) do 8: if (\( q \in \mathcal{S} \) and \( q = \bigwedge (\psi \in S_q) \delta(\psi, \Pi) \)) then 9: \( \mathcal{S} \leftarrow \mathcal{S} \cup \{q\} \) \( \triangleright \) update set of states 10: \( \varrho \leftarrow \varrho \cup \{(q, \Pi, q')\} \) \( \triangleright \) update transition relation

where \( q' \) is a set of quoted subformulas of \( \varphi \) that denotes a minimal interpretation such that \( q' \models \bigwedge (\psi \in S_q) \delta(\psi, \Pi) \), but only the minimal ones. Notice that trivially we have \((0, a, \emptyset) \in \varrho \) for every \( a \in \Sigma \).

The algorithm LTL\textsubscript{f}/2NFA terminates in at most exponential number of steps, and generates a set of states \( \mathcal{S} \) whose size is at most exponential in the size of the formula \( \varphi \).

**Theorem 9.** Let \( \varphi \) be an LTL\textsubscript{f} formula and \( A_\varphi \) the NFA constructed as above. Then \( \varphi \models \pi \) iff \( \pi \in L(A_\varphi) \) for every finite trace \( \pi \).

**Proof (sketch).** Given a specific formula \( \varphi \), \( \delta \) grounded on the subformulas of \( \varphi \) becomes the transition function of the AFW, with initial state \( "\varphi" \) and no final states, corresponding to \( \varphi \) (De Giacomo and Vardi 2013). Then LTL\textsubscript{f}/2NFA essentially transforms the AFW into a NFA.

Notice that above we have assumed to have a special proposition \( \text{last} \in \mathcal{P} \). If we want to remove such an assumption, we can easily transform the obtained automaton \( A_\varphi = (2^\mathbb{P}, \mathcal{S}, \{\varphi\}, \varrho, \{\emptyset\}) \) into the new automaton \( A'_\varphi = (2^{2^\mathbb{P} \setminus \{\text{last}\}}, \mathcal{S} \cup \{\text{ended}\}, \{\varphi\}, \varrho, \{\emptyset, \text{ended}\}) \) where: \( (q, \Pi', q') \in \varrho \) iff \( (q, \Pi, q') \in \varrho \) or \( (q, \Pi) \in \{\text{last}\} \), \( true \in \varrho \) and \( q' = \text{ended} \).

It is easy to see that the NFA obtained can be built on-the-fly while checking for nonemptiness, hence we have:

**Theorem 10.** Satisfiability of an LTL\textsubscript{f} formula can be checked in PSPACE by nonemptiness of \( A_\varphi \) (or \( A'_\varphi \)).

Considering that validity and logical implications can be linearly reduced to satisfiability in LTL\textsubscript{f} (see Theorem 1), we can conclude the proposed construction is optimal wrt computational complexity for reasoning on LTL\textsubscript{f}.

We conclude this section by observing that the obtained NFA (or in fact any correct NFA for LTL\textsubscript{f} in the literature, e.g., (Bauer and McIraith 2006), one can easily check when the NFA obtained via the approach in (Edelkamp 2006) mentioned in the introduction, i.e., using directly the Büchi automaton for the formula, but by substituting the Büchi acceptance condition with the NFA one, is indeed correct, by simply checking language equivalence.

### 7 Conclusions

While the blurring between infinite and finite traces has been of help as a jump start, we should now sharpen our focus on LTL on finite traces (LTL\textsubscript{f}). This paper does it in two ways: by showing notable cases where the blurring does not harm (witnessed by insensitivity to infiniteness); and by proposing a direct route to develop algorithms for finite traces (as witnessed by the algorithm LTL\textsubscript{f}/2NFA). Along the latter line, we note that LTL\textsubscript{f}/2NFA can easily be extended to deal with the more powerful LDL\textsubscript{f} (De Giacomo and Vardi 2013). In future work, we plan to investigate runtime monitoring (Bauer, Leucker, and Schallhart 2007) by using LTL\textsubscript{f} and LDL\textsubscript{f} monitors.

**Acknowledgments.** This research has been partially supported by the EU project Optique (FP7-IP-318338), and by the Sapienza Award 2013 “SPIRITLETs: SPIRITLET–based Smart spaces”.


References


