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 $\Box(a \equiv \bullet\neg b)$ (that is, $\Box(a \equiv \neg\circ b)$). 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 allowed evolutions, 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/LTL_f 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 $\mathcal{P}$ into fluents $\mathcal{F}$ and actions $\mathcal{A}$, adding structural constraint (analogous to the DECLARE assumption) such as $\Box(\bigvee_{a\in\mathcal{A}} a) \wedge \Box(\bigwedge_{a\in\mathcal{A}}(a \rightarrow \bigwedge_{b\in\mathcal{A},b\neq a} \neg b))$, 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 φ_{init} involving only fluents, while effects can be modelled as:

$$\Box(\varphi \rightarrow \circ(a \rightarrow \psi)) \qquad (2)$$

where $a \in \mathcal{A}$, while ψ and φ are arbitrary propositional formulas involving only fluents. Such a formula states that performing action a under the conditions denoted by φ brings about the conditions denoted by ψ.[3] 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 $F(do(a,s)) \equiv \varphi^+(s) \vee (F(s) \wedge \neg\varphi^-(s))$ into the LTL_f formula:

$$\Box(\circ a \rightarrow (\circ F \equiv \varphi^+ \vee F \wedge \neg\varphi^-)). \qquad (3)$$

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:

$$\varphi ::= \phi \mid \circ\phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2, \quad \text{where } \phi \text{ is propositional.}$$

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:

[3]A formula like $\Box(\varphi \rightarrow \circ(a \rightarrow \varphi))$ corresponds to a frame axiom expressing that φ does not change when performing a.

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

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

By applying the above theorem we can immediately show that (3) and (2) (for the latter, noting that it is equivalent to $\Box(\circ a \rightarrow (\varphi \rightarrow \circ\psi))$) 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):

$$
\begin{aligned}
(\texttt{at end}\,\phi) &::= last \wedge \phi\\
(\texttt{always}\,\phi) &::= \Box\phi\\
(\texttt{sometime}\,\phi) &::= \Diamond\phi\\
(\texttt{within}\,n\,\phi) &::= \textstyle\bigvee_{0\leq i\leq n} \underbrace{\circ\cdots\circ}_{i}\phi\\
(\texttt{hold-after}\,n\,\phi) &::= \underbrace{\circ\cdots\circ}_{n+1}\Diamond\phi\\
(\texttt{hold-during}\,n_1\,n_2\,\phi) &::= \underbrace{\circ\cdots\circ}_{n_1}(\textstyle\bigwedge_{0\leq i\leq n_2} \underbrace{\circ\cdots\circ}_{i}\phi)\\
(\texttt{at-most-once}\,\phi) &::= \Box(\phi \rightarrow \phi\,\mathcal{W}\,\neg\phi)\\
(\texttt{sometime-after}\,\phi_1\,\phi_2) &::= \Box(\phi_1 \rightarrow \Diamond\phi_2)\\
(\texttt{sometime-before}\,\phi_1\,\phi_2) &::= (\neg\phi_1 \wedge \neg\phi_2)\,\mathcal{W}(\neg\phi_1 \wedge \phi_2)\\
(\texttt{always-within}\,n\,\phi_1\,\phi_2) &::= \Box(\phi_1 \rightarrow \textstyle\bigvee_{0\leq i\leq n} \underbrace{\circ\cdots\circ}_{i}\phi_2)
\end{aligned}
$$

where ϕ 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 4. Moreover, the following general result holds. Let a *goal formula* be *guarded* when it is equivalent to $(\bigvee_{F\in\mathcal{F}} F) \wedge \phi$ with ϕ any propositional formula. Then:

Theorem 8. *All trajectory constraints involving only guarded goal formulas, except from* (`always` φ)*, are insensitive to infiniteness.*

6 Reasoning in LTL_f through NFAs

We can associate with each LTL_f formula φ an (exponential) NFA A_φ that accepts exactly the traces that make φ 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 Guldstrand 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 (Baier and McIlraith 2006) relies on the procedure in (Gerth et al. 1995) to build the NFA, but it implements the full LTL_f semantics by dealing also with the next operator.

Here, we provide a simple direct algorithm for computing the NFA corresponding to an LTL_f formula. The correctness of the algorithm is based on the fact that (*i*) we can associate with each LTL_f formula φ a polynomial *alternating automaton on words* (AFW) $\mathcal{A}_\varphi$ that accept exactly the traces that make φ 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_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_f formula can be transformed into negation normal form in linear time). We also assume $\mathcal{P}$ to include a special proposition $last$ which denotes the last element of the trace. Note that $last$ can be defined as $last \equiv \bullet \mathit{false}$. Then we define an auxiliary function δ that takes an LTL_f formula ψ (in negation normal form) and a propositional interpretation Π for $\mathcal{P}$ (including $last$), returning a positive boolean formula whose atoms are (quoted) ψ subformulas.

$$
\begin{array}{ll}
\delta(\texttt{"}a\texttt{"}, \Pi) & = \mathit{true} \text{ if } a \in \Pi \\
\delta(\texttt{"}a\texttt{"}, \Pi) & = \mathit{false} \text{ if } a \notin \Pi \\
\delta(\texttt{"}\neg a\texttt{"}, \Pi) & = \mathit{false} \text{ if } a \in \Pi \\
\delta(\texttt{"}\neg a\texttt{"}, \Pi) & = \mathit{true} \text{ if } a \notin \Pi \\
\delta(\texttt{"}\varphi_1 \wedge \varphi_2\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \wedge \delta(\texttt{"}\varphi_2\texttt{"}, \Pi) \\
\delta(\texttt{"}\varphi_1 \vee \varphi_2\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \vee \delta(\texttt{"}\varphi_2\texttt{"}, \Pi) \\
\delta(\texttt{"}\circ\varphi\texttt{"}, \Pi) & = \begin{cases} \texttt{"}\varphi\texttt{"} & \text{if } last \notin \Pi \\ \mathit{false} & \text{if } last \in \Pi \end{cases} \\
\delta(\texttt{"}\Diamond\varphi\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi\texttt{"}, \Pi) \vee \delta(\texttt{"}\circ\Diamond\varphi\texttt{"}, \Pi) \\
\delta(\texttt{"}\varphi_1 \mathcal{U} \varphi_2\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi_2\texttt{"}, \Pi) \vee \\
 & \quad (\delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \wedge \delta(\texttt{"}\circ(\varphi_1 \mathcal{U} \varphi_2)\texttt{"}, \Pi)) \\
\delta(\texttt{"}\bullet\varphi\texttt{"}, \Pi) & = \begin{cases} \texttt{"}\varphi\texttt{"} & \text{if } last \notin \Pi \\ \mathit{true} & \text{if } last \in \Pi \end{cases} \\
\delta(\texttt{"}\Box\varphi\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi\texttt{"}, \Pi) \wedge \delta(\texttt{"}\bullet\Box\varphi\texttt{"}, \Pi) \\
\delta(\texttt{"}\varphi_1 \mathcal{R} \varphi_2\texttt{"}, \Pi) & = \delta(\texttt{"}\varphi_2\texttt{"}, \Pi) \wedge (\delta(\texttt{"}\varphi_1\texttt{"}, \Pi) \vee \\
 & \quad \delta(\texttt{"}\bullet(\varphi_1 \mathcal{R} \varphi_2)\texttt{"}, \Pi))
\end{array}
$$

Using function δ we can build the NFA A_φ of an LTL_f formula φ in a forward fashion. States of A_φ are sets of atoms (recall that each atom is quoted φ subformulas) to be interpreted as a conjunction; the empty conjunction $\emptyset$ stands for true:

algorithm $\text{LTL}_f2\text{NFA}()$
input LTL_f formula φ
output NFA $A_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\})$
$s_0 \leftarrow \{\texttt{"}\varphi\texttt{"}\}$ ▷ single initial state
$s_f \leftarrow \emptyset$ ▷ single final state
$\mathcal{S} \leftarrow \{s_0, s_f\}$, $\varrho \leftarrow \emptyset$
while ($\mathcal{S}$ or ϱ change) **do**
  if ($q \in \mathcal{S}$ and $q' \models \bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$) **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{q'\}$ ▷ update set of states
    $\varrho \leftarrow \varrho \cup \{(q, \Pi, q')\}$ ▷ update transition relation

where q' is a set of quoted subformulas of φ that denotes a minimal interpretation such that $q' \models \bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$. (Note: we do not need to get all q such that $q' \models \bigwedge_{(\texttt{"}\psi\texttt{"} \in q)} \delta(\texttt{"}\psi\texttt{"}, \Pi)$, but only the minimal ones.) Notice that trivially we have $(\emptyset, a, \emptyset) \in \varrho$ for every $a \in \Sigma$.

The algorithm LTL_f2NFA 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 φ.

Theorem 9. *Let φ be an LTL_f formula and A_φ the NFA constructed as above. Then $\pi \models \varphi$ iff $\pi \in L(A_\varphi)$ for every finite trace π.*

Proof (sketch). Given a specific formula φ, δ grounded on the subformulas of φ becomes the transition function of the AFW, with initial state $\texttt{"}\varphi\texttt{"}$ and no final states, corresponding to φ (De Giacomo and Vardi 2013). Then LTL_f2NFA essentially transforms the AFW into a NFA. □

Notice that above we have assumed to have a special proposition $last \in \mathcal{P}$. If we want to remove such an assumption, we can easily transform the obtained automaton $A_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{\texttt{"}\varphi\texttt{"}\}, \varrho, \{\emptyset\})$ into the new automaton

$$A'_\varphi = (2^{\mathcal{P} - \{last\}}, \mathcal{S} \cup \{\mathit{ended}\}, \{\texttt{"}\varphi\texttt{"}\}, \varrho', \{\emptyset, \mathit{ended}\})$$

where: $(q, \Pi', q') \in \varrho'$ iff $(q, \Pi', q') \in \varrho$, or $(q, \Pi' \cup \{last\}, \mathit{true}) \in \varrho$ and $q' = \mathit{ended}$.

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_f formula can be checked in PSPACE by nonemptiness of A_φ (or A'_φ).*

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

We conclude this section by observing that using the obtained NFA (or in fact any correct NFA for LTL_f in the literature, e.g., (Baier and McIlraith 2006)), one can easily check when the NFA obtained via the approach in (Edelkamp 2006; Gerevini et al. 2009) 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_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_f2NFA). Along the latter line, we note that LTL_f2NFA can easily be extended to deal with the more powerful LDL_f (De Giacomo and Vardi 2013). In future work, we plan to investigate runtime monitoring (Bauer, Leucker, and Schallhart 2007) by using LTL_f and LDL_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

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

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proc. of the 21th AAAI Conf. on Artificial Intelligence (AAAI)*, 788–795.

Baier, C.; Katoen, J.-P.; and Guldstrand Larsen, K. 2008. *Principles of Model Checking*. The MIT Press.

Bauer, A., and Haslum, P. 2010. LTL goal specifications revisited. In *Proc. of the 19th Eu. Conf. on Artificial Intelligence (ECAI)*, 881–886. IOS Press.

Bauer, A.; Leucker, M.; and Schallhart, C. 2007. The good, the bad, and the ugly, but how ugly is ugly? In *Proc. of the 7th Int. Ws. on Runtime Verification (RV)*, 126–138.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, 593–602. Morgan Kaufmann.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*. IJCAI/AAAI.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1999. Patterns in property specifications for finite-state verification. In *Proc. of the 1999 Int. Conf. on Software Engineering (ICSE)*, 411–420. ACM Press.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *Proc. of the 16th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 374–377. AAAI.

Gabbay, D.; Pnueli, A.; Shelah, S.; and Stavi, J. 1980. On the temporal analysis of fairness. In *Proc. of the 7th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POLP)*, 163–173.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Gerth, R.; Peled, D.; Vardi, M. Y.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. of the 15th IFIP Int. Symp. on Protocol Specification, Testing and Verification (PSTV)*, 3–18. IFIP.

Giannakopoulou, D., and Havelund, K. 2001. Automata-based verification of temporal properties on running programs. In *Proc. of the 16th IEEE Int. Conf. on Automated Software Engineering (ASE)*, 412–416. IEEE Computer Society.

Holzmann, G. J. 1995. Tutorial: Proving properties of concurrent system with spin. In *Proc. of the 6th Int. Conf. on Concurrency Theory (CONCUR)*, LNCS, 453–455. Springer.

Lichtenstein, O.; Pnueli, A.; and Zuck, L. D. 1985. The glory of the past. In *Logic of Programs*, 196–218.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl – the planning domain definition language – version 1.2. Technical report, TR-98-003, Yale Center for Computational Vision and Contro.

Montali, M.; Pesic, M.; van der Aalst, W. M. P.; Chesani, F.; Mello, P.; and Storari, S. 2010. Declarative specification and verification of service choreographies. *ACM Transactions on the Web* 4(1).

Montali, M. 2010. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *LNBIP*. Springer.

Pesic, M., and van der Aalst, W. M. P. 2006. A declarative approach for flexible business processes management. In *Proc. of the BPM 2006 Workshops*, LNCS, 169–180. Springer.

Pesic, M.; Schonenberg, H.; and van der Aalst, W. M. P. 2007. Declare: Full support for loosely-structured processes. In *Proc. of the 11th IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC)*, 287–300. IEEE Computer Society.

Pesic, M. 2008. *Constraint-Based Workflow Management Systems: Shifting Controls to Users*. Ph.D. Dissertation, Beta Research School for Operations Management and Logistics, Eindhoven.

Pnueli, A. 1977. The temporal logic of programs. In *Proc. of the 18th Ann. Symp. on Foundations of Computer Science (FOCS)*, 46–57. IEEE Computer Society.

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

Rozier, K. Y., and Vardi, M. Y. 2007. LTL satisfiability checking. In *Proc. of SPIN - Model Checking Software*, LNCS, 149–167. Springer.

Sun, Y.; Xu, W.; and Su, J. 2012. Declarative choreographies for artifacts. In *Proc. of the 10th Int. Conf. on Service-Oriented Computing (ICSOC)*, LNCS, 420–434. Springer.

van der Aalst, W. M. P., and Pesic, M. 2006. Decserflow: Towards a truly declarative service flow language. In *Proc. of the 3rd Ws. on Web Services and Formal Methods (WS-FM2006)*, LNCS, 1–23. Springer.

Vardi, M. Y. 1996. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS. Springer. 238–266.

Westergaard, M. 2011. Better algorithms for analyzing and enacting declarative workflow languages using LTL. In *Proc. of the 9th Int. Conf. on Business Process Management (BPM)*, LNCS, 83–98. Springer.