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,demasellis}@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) blurred. Possibly because of this blurring, virtually all literature detours to Büchi automata for constructing the NFA that accepts the traces satisfying an LTL$_f$ formula. As a further contribution, 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 for Planning in PDDL 3.0 (Bacchus and Kabanza 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 (Pesic 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 (Pesic, 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 (Pesic 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), 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* $\Diamond e \wedge \Box(e \rightarrow \circ e))$.*"* 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 (Baier and McIlraith 2006; De Giacomo and Vardi 2013). So, *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).[1]

[1] 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_f formula is insensitive to infiniteness. Using such a condition, we show that all LTL_f formulas corresponding to the DECLARE patterns but one, are indeed insensitive to infiniteness. We also show that virtually all transition-based specifications of action domains expressed in LTL_f 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_f formula (Giannakopoulou and Havelund 2001; Edelkamp 2006; Baier and McIlraith 2006; Baier, 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 $\mathcal{P}$ 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*):

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \circ\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \quad \text{with } a \in \mathcal{P}$$

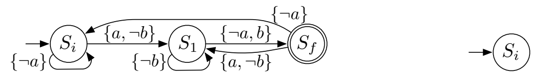
Intuitively, $\circ\varphi$ says that φ holds at the *next* instant, $\varphi_1 \,\mathcal{U}\, \varphi_2$ says that at some future instant φ_2 will hold and *until* that point φ_1 holds. Common abbreviations are also used, including the ones listed below.

- Standard boolean abbreviations, such as $true$, $false$, $\vee$, $\rightarrow$.
- $last = \neg\circ true$ denotes the last instant of the sequence. Over infinite traces it corresponds to $\circ false$ and is indeed always false, while in LTL_f it becomes true at the last instance of the sequence.
- $\bullet\varphi = \neg\circ\neg\varphi$ is interpreted as a *weak next*, stating that if $last$ does not hold then φ must hold in the next state.
- $\Diamond\varphi = true \,\mathcal{U}\, \varphi$ says that φ will *eventually* hold before the last instant (included).
- $\Box\varphi = \neg\Diamond\neg\varphi$ says that from the current instant till the last instant φ will *always* hold.
- $\varphi_1 \,\mathcal{R}\, \varphi_2 = \neg(\neg\varphi_1 \,\mathcal{U}\, \neg\varphi_2)$ means that φ_1 *releases* φ_2, i.e., either φ_2 must hold forever, or until φ_1 also holds.
- $\varphi_1 \,\mathcal{W}\, \varphi_2 = (\varphi_1 \,\mathcal{U}\, \varphi_2 \vee \Box\varphi_1)$ is interpreted as a *weak until*, and means that φ_1 holds until φ_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 π over the alphabet of $2^{\mathcal{P}}$, containing all possible interpretations of the propositional symbols in $\mathcal{P}$. Given a finite trace π, we inductively define when an LTL_f formula φ *is true* at an instant i (for $0 \leq i \leq n$), written $\pi, i \models \varphi$, as:

- $\pi, i \models a$, for $a \in \mathcal{P}$ 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 $i < n$ and $\pi, i{+}1 \models \varphi$.
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ 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 $\pi, k \models \varphi_1$.

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



(a) Büchi automaton (b) NFA

Figure 1: Automata for formula (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 finite trace, and it is *valid* if it is true in every finite trace. A formula φ logically implies a formula φ', written $\varphi \models \varphi'$, if for every finite trace π 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 φ is valid iff $\neg\varphi$ is unsatisfiable. Similarly, $\varphi \models \varphi'$ iff $\varphi \wedge \neg\varphi'$ 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

$$\Diamond a \wedge \Box(a \rightarrow \Diamond b) \wedge \Box(b \rightarrow \Diamond a) \wedge \Box(\neg a \vee \neg b) \qquad (1)$$

is unsatisfiable in LTL_f but is satisfiable in LTL. In other words, in a finite trace setting $\Diamond a \wedge \Box(a \rightarrow \Diamond b) \wedge \Box(b \rightarrow \Diamond a)$ 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 φ we can construct a corresponding LTL formula, as follows: (*i*) introduce a fresh proposition "end" to denote that the trace is ended (note that $end \notin \mathcal{P}$, and that $last$ is true just before the first occurrence of end); (*ii*) require that end eventually holds ($\Diamond end$); (*iii*) require that once end becomes true it stays true forever ($\Box(end \rightarrow \circ end)$); (*iv*) require that when end is true, all other propositions are reset to false ($\Box(end \rightarrow \bigwedge_{a \in \mathcal{P}} \neg a)$); (*v*) translate the LTL_f formula into an LTL formula as follows:

$$\begin{array}{rlrl}
f(a) & \mapsto a & f(\varphi_1 \,\mathcal{U}\, \varphi_2) & \mapsto f(\varphi_1) \;\mathcal{U}\; (f(\varphi_2) \wedge \neg end) \\
f(\neg\varphi) & \mapsto \neg f(\varphi) & f(\Diamond\varphi) & \mapsto \Diamond(f(\varphi) \wedge \neg end) \\
f(\varphi_1 \wedge \varphi_2) & \mapsto f(\varphi_1) \wedge f(\varphi_2) & f(\bullet\varphi) & \mapsto \circ(f(\varphi) \vee end) \\
f(\circ\varphi) & \mapsto \circ(f(\varphi) \wedge \neg end) & f(\Box\varphi) & \mapsto \Box(f(\varphi) \vee end)
\end{array}$$

Theorem 2. *Let π_i be an infinite trace. Then*

$$\pi_i \models \Diamond end \wedge \Box(end \rightarrow \circ end) \wedge \Box(end \rightarrow \bigwedge_{a \in \mathcal{P}} \neg a)$$

iff it is has the form $\pi_i = \pi_f\{end\}^\omega$, where end is always false in π_f.

Proof (sketch). The only if direction is immediate. For the if direction, suffice it to observe that if π_i satisfies $\Diamond end \wedge \Box(end \rightarrow \circ end) \wedge \Box(end \rightarrow \bigwedge_{a \in \mathcal{P}} \neg a)$ then there must be a first instant in which end becomes true, hence π_i must have the form $\pi_f\{end\}^\omega$ where end is always false in π_f. $\Box$

Theorem 3. *Let φ be an* LTL$_f$ *formula and $\pi_i = \pi_f\{end\}^\omega$ an infinite trace where end is always false in π_f. Then*

$$\pi_f \models \varphi \textit{ iff } \pi_f\{end\}^\omega \models f(\varphi).$$

Proof (sketch). Both direction can be shown by induction on the structure of the formula φ. $\Box$

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 φ is* insensitive to infiniteness *if for every (infinite) trace $\pi_i = \pi_f\{end\}^\omega$ where end is always false in π_f, we have that*

$$\pi_f \models \varphi \textit{ iff } \pi_f\{end\}^\omega \models \varphi$$

LTL$_f$ formulas that are insensitive to infiniteness can be translated into LTL by simply adding the conditions on 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 end of the significant part and that once such 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$_f$ *φ is insensitive to infiniteness if and only if the following* LTL *formula is valid:*

$$(\Diamond end \wedge \Box(end \rightarrow \circ end) \wedge \Box(end \rightarrow \bigwedge_{a \in \mathcal{P}} \neg a)) \rightarrow (\varphi \equiv f(\varphi))$$

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

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

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 $\Diamond end \wedge \Box(end \rightarrow \circ end) \wedge \Box(end \rightarrow \bigwedge_{a \in \mathcal{P}} \neg a)) \rightarrow (\varphi \equiv f(\varphi)$ 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 φ_1 and φ_2 be two* LTL$_f$ *formulas that are insensitive to infiniteness. Then the* LTL$_f$ *formulas $\neg\varphi_i$ ($i = 1, 2$) and $\varphi_1 \wedge \varphi_2$ are also insensitive to infiniteness.*

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

4 DECLARE Process Modeling

DECLARE is a language and framework for the declarative, constraint-based modelling of processes and services. It started from seminal works on ConDec (Pesic and van der Aalst 2006) and DecSerFlow (van der Aalst and Pesic 2006; Montali et al. 2010). A thorough treatment of constraint-based processes can be found in (Pesic 2008; Montali 2010).

The DECLARE framework provides a set $\mathcal{P}$ of propositions representing atomic tasks (i.e., actions), which are units of work in the process. Notice that properties of states are not represented. DECLARE assumes that, at each point in time, one and only one task is executed, and that the process eventually terminates. Following the second assumption, LTL$_f$ is used to specify DECLARE processes, whereas the first assumption is captured by the following LTL$_f$ formula, assumed as an implicit constraint: $\xi_{\mathcal{P}} = \Box(\bigvee_{a \in \mathcal{P}} a) \wedge \Box(\bigwedge_{a,b \in \mathcal{P}, a \neq b} a \rightarrow \neg b)$, which we call the DECLARE *assumption*.

A DECLARE model is a set $\mathcal{C}$ of LTL$_f$ constraints over $\mathcal{P}$, used to restrict the allowed execution traces. Among all possible LTL$_f$ constraints, some specific *patterns* have been singled out as particularly meaningful for expressing DECLARE processes, taking inspiration from (Dwyer, Avrunin, and Corbett 1999). As shown in Table 1, patterns are grouped into four families: *(i) existence* (unary) constraints, stating that the target task must/cannot be executed (a certain amount of times); *(ii) choice* (binary) constraints, modeling choice of execution; *(iii) relation* (binary) constraints, modeling that whenever the source task is executed, then the target task must also be executed (possibly with additional requirements); *(iv) negation* (binary) constraints, modeling that whenever the source task is executed, then the target task cannot be executed (possibly with additional restrictions).

Observe that the set of finite traces that satisfies the constraints $\mathcal{C}$ together with the DECLARE assumption $\xi_{\mathcal{P}}$ can be captured by a single deterministic process, obtained by:

1. generating the corresponding NFA (exponential step);
2. transforming it into a DFA- *deterministic finite-state automaton* (exponential step);
3. trimming the resulting DFA by removing every state from which no final state is reachable (polynomial step).

The obtained DFA is indeed a process in the sense that at every step, depending only on the history (i.e., the current state), it exposes the set of tasks that are legally executable and eventually lead to a final state (assuming fairness of the execution, which disallows remaining forever in a loop). In

Table 1: Declare patterns and their insensitivity to infiniteness

	NAME	NOTATION	LTL$_f$ FORMALIZATION	DESCRIPTION	INSENSITIVE
EXISTENCE	**Existence**	1..* a	$\Diamond a$	a must be executed at least once	Y
	Absence 2	0..1 a	$\neg\Diamond(a \wedge \Diamond a)$	a can be executed at most once	Y
CHOICE	**Choice**	a b	$\Diamond a \vee \Diamond b$	a or b must be executed	Y
	Exclusive Choice	a b	$(\Diamond a \vee \Diamond b) \wedge \neg(\Diamond a \wedge \Diamond b)$	Either a or b must be executed, but not both	Y
RELATION	**Resp. existence**	a b	$\Diamond a \rightarrow \Diamond b$	If a is executed, then b must be executed as well	Y
	Coexistence	a b	$(\Diamond a \rightarrow \Diamond b) \wedge (\Diamond b \rightarrow \Diamond a)$	Either a and b are both executed, or none of them is executed	Y
	Response	a b	$\Box(a \rightarrow \Diamond b)$	Every time a is executed, b must be executed afterwards	Y
	Precedence	a b	$\neg b \,\mathcal{W}\, a$	b can be executed only if a has been executed before	Y
	Succession	a b	$\Box(a \rightarrow \Diamond b) \wedge (\neg b \,\mathcal{W}\, a)$	b must be executed after a, and a must precede b	Y
	Alt. Response	a b	$\Box(a \rightarrow \circ(\neg a \,\mathcal{U}\, b))$	Every a must be followed by b, without any other a inbetween	Y
	Alt. Precedence	a b	$(\neg b \,\mathcal{W}\, a) \wedge \Box(b \rightarrow \circ(\neg b \,\mathcal{W}\, a))$	Every b must be preceded by a, without any other b inbetween	Y
	Alt. Succession	a b	$\Box(a \rightarrow \circ(\neg a \,\mathcal{U}\, b)) \wedge (\neg b \,\mathcal{W}\, a) \wedge \Box(b \rightarrow \circ(\neg b \,\mathcal{W}\, a))$	Combination of alternate response and alternate precedence	Y
	Chain Response	a b	$\Box(a \rightarrow \circ b)$	If a is executed then b must be executed next	Y
	Chain Precedence	a b	$\Box(\circ b \rightarrow a)$	Task b can be executed only immediately after a	Y
	Chain Succession	a b	$\Box(a \equiv \circ b)$	Tasks a and b must be executed next to each other	Y
NEGATION	**Not Coexistence**	a b	$\neg(\Diamond a \wedge \Diamond b)$	Only one among tasks a and b can be executed, but not both	Y
	Neg. Succession	a b	$\Box(a \rightarrow \neg\Diamond b)$	Task a cannot be followed by b, and b cannot be preceded by a	Y
	Neg. Chain Succession	a b	$\Box(a \equiv \circ\neg b)$	Tasks a and b cannot be executed next to each other	N

the DECLARE implementation, this process is maintained implicit, and traces are generated incrementally, in accordance with the constraints. This requires an engine that, at each state, infers which tasks are legal (cf. *enactment* below).

Three fundamental reasoning services are of interest in DECLARE: *(i) Consistency*, which checks whether the model is executable, i.e., there exists at least one (finite) trace π_f over $\mathcal{P}$ such that $\pi_f \models \mathcal{C} \wedge \xi_{\mathcal{P}}$ where, with a little abuse of notation, we use $\mathcal{C}$ to denote the LTL$_f$ formula $\bigwedge_{C \in \mathcal{C}} C$. *(ii) Detection of dead tasks*, which checks whether the model contains tasks that can never be executed; a task $a \in \mathcal{P}$ is *dead* if, for every (finite) trace π_f over $\mathcal{P}$: $\pi_f \models (\mathcal{C} \wedge \xi_{\mathcal{P}}) \rightarrow \Box\neg a$. *(iii) Enactment*, which drives the execution of the model, inferring which tasks are currently legal, which constraints are currently pending (i.e., require to do something), and whether the execution can be currently ended or not; specifically, given a (finite) trace π_1 denoting the execution so far:

- Task $a \in \mathcal{P}$ is *legal in* π_1 if there exist a (finite, possibly empty) trace π_2 s.t.: $\pi_1 a \pi_2 \models \mathcal{C} \wedge \xi_{\mathcal{P}}$.
- Constraint $C \in \mathcal{C}$ is *pending in* π_1 if $\pi_1 \not\models C$.
- The execution can be *ended in* π_1 if $\pi_1 \models \mathcal{C} \wedge \xi_{\mathcal{P}}$.

All these services reduce to standard reasoning in LTL$_f$.

It turns out that all DECLARE patterns but one are insensitive to infiniteness (see Table 1).

Theorem 6. *All the* DECLARE *patterns, with the exception of* negation chain succession*, are insensitive to infiniteness, independently from the* DECLARE *assumption.*

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[2], 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
(F(end) & G(end -> X(end)) & G(end -> (!b & !a)))
 -> ( (G(a -> X(F(b)))) <->
      (G((a -> (X((F(b & !end))) & !end)) | end)) )
```

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 on whether $\mathcal{P}$ 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, $\Box(a \equiv \circ\neg b)$ 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 $\{a\}\{end\}^\omega \models \Box(a \equiv \circ\neg b)$, but $\{a\} \not\models \Box(a \equiv \circ\neg b)$. 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 $\Box(a \equiv \bullet\neg b)$ (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:

$$\Box(\varphi \rightarrow \circ(a \rightarrow \psi)) \tag{2}$$

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:

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

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):

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

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 ($\texttt{always}\ \varphi$), 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 LTLf2NFA()
input LTLf formula φ
output NFA A_φ = (2^P, S, {s_0}, ϱ, {s_f})
s_0 ← {"φ"}                                   ▷ single initial state
s_f ← ∅                                       ▷ single final state
S ← {s_0, s_f}, ϱ ← ∅
while (S or ϱ change) do
    if (q ∈ S and q' ⊨ ⋀_("ψ"∈q) δ("ψ", Π)) then
        S ← S ∪ {q'}                          ▷ update set of states
        ϱ ← ϱ ∪ {(q, Π, 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 $\mathrm{LTL}_f 2\mathrm{NFA}$ 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 $\mathrm{LTL}_f 2\mathrm{NFA}$ 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 $\mathrm{LTL}_f 2\mathrm{NFA}$). Along the latter line, we note that $\mathrm{LTL}_f 2\mathrm{NFA}$ 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 and Kabanza 2000] Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artif. Intell.* 116(1-2):123–191.

[Baier and McIlraith 2006] 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, Katoen, and Guldstrand Larsen 2008] Baier, C.; Katoen, J.-P.; and Guldstrand Larsen, K. 2008. *Principles of Model Checking*. The MIT Press.

[Bauer and Haslum 2010] 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, Leucker, and Schallhart 2007] 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, De Giacomo, and Vardi 2002] 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 and Vardi 2013] 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, Avrunin, and Corbett 1999] 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 2006] 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 et al. 1980] 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 et al. 2009] 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 et al. 1995] 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 and Havelund 2001] 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 1995] 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, Pnueli, and Zuck 1985] Lichtenstein, O.; Pnueli, A.; and Zuck, L. D. 1985. The glory of the past. In *Logic of Programs*, 196–218.

[McDermott et al. 1998] 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 et al. 2010] 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 2010] Montali, M. 2010. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *LNBIP*. Springer.

[Pesic and van der Aalst 2006] 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, Schonenberg, and van der Aalst 2007] 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 2008] 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 1977] 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 2001] Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

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

[Sun, Xu, and Su 2012] 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 and Pesic 2006] 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 1996] Vardi, M. Y. 1996. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS. Springer. 238–266.

[Westergaard 2011] Westergaard, M. 2011. Better algorithms for analyzing and enacting declarative workflow languages using LTL. In *BPM*, 83–98.