From LTL on Process Traces to Finite-State Automata

Francesco Chiariello¹²*, Fabrizio Maria Maggi³ and Fabio Patrizi¹

¹DIAG - Sapienza University of Rome, Italy
²DIETI - University of Naples Federico II, Italy
³KRDB - Free University of Bozen-Bolzano, Italy

Abstract
Linear Temporal Logic on process traces (or LTLₚ) is a logic introduced to specify and reason over the temporal properties of (the traces generated by) business processes. So far, its relation with finite-state automata has not been explored and researchers resorted to more expressive logics and the corresponding automata construction algorithms. In this paper, we present a tool, named LTLₚ2DFA, to automatically construct the automata associated with LTLₚ specifications and show how, by considering process traces as first-class citizens, this results in simpler automata and better construction algorithms.

Keywords
DECLARE, Declarative Process Specifications, Finite-State Automata, Temporal Logics

1. Introduction
declare [1] is the most common declarative process specification language. This type of language allows one to specify what should be done rather than how it should be done, as is instead the case for imperative models such as Petri nets [2, 3] and BPMN [4, 5]. declare consists of a set of templates for expressing constraints over process activities. For example, the template Response(a, b) says that ‘whenever a occurs, b must occur afterward’. The constraints are then obtained by instantiating the template variables (a and b in the example above) with a particular activity.

It has been shown that the semantics of declare can be grounded into Linear Temporal Logic on finite traces (LTLₖ) [6]. Variants on finite traces of well-established temporal logics have been considered for analyzing terminating tasks, such as operational processes (see, e.g., [7, 8]). Since process traces (also called simple finite traces in the literature) are finite, LTLₖ turns out to be, as observed by De Giacomo et al. [9], a more natural choice for expressing process-trace properties than LTL (on infinite traces) [10], originally used to formalize declare [11]. Using LTLₖ allows for easily constructing Deterministic Finite Automata (DFAs) representing the process constraints. As a consequence, there have been a number of works from the Business Process Management (BPM) and Process Mining (PM) communities which directly use LTLₖ as
a specification language, usually taking advantage of the automata-representation of the LTL$_f$ formulae. For example, LTL$_f$ specifications have been considered in [12] for Trace Alignment, in [13] for Runtime Monitoring, in [14] for Vacuity Detection, and in [15] to measure the degree of compliance of process models with event logs.

In addition to finiteness, process traces feature another notable property, which distinguishes them from generic (finite) traces. Namely, at each time step, the former contains exactly one activity (also referred to as the declare assumption in [9], and which we rename simplicity assumption), while the latter may include any number of activities. This raises the question of whether LTL$_f$, which is powerful enough to deal with generic traces, is in fact too general for process traces. Specifically, the problem is whether the (automata-based) machinery used to check LTL$_f$ properties on generic traces can be simplified in the presence of process-traces only.

Observe that while process traces can be dealt with in LTL$_f$ (see [9]), this significantly increases the size of the LTL$_f$ formula and, in turn, the construction time of the corresponding automaton. To overcome all these problems and make the semantics of the temporal logic match that of declare, Fionda and Greco [16] introduced LTL on process traces (or LTL$_p$), which natively incorporates the simplicity assumption, without yielding the growth in the size of the formula.

Here, we show how using LTL$_p$ formulae leads to simpler automata than those obtained by using LTL$_f$, and provide a tool, named LTLp2DFA, to construct such automata. Besides being simpler, automata could be obtained more efficiently, by exploiting the simplicity assumption in the automata construction. The simplification has already been used in [17, 18, 19] to improve the Answer Set Programming encoding [20] of various Declarative PM tasks for the analysis of real-life logs. Also, if one wants to take advantage of Automata Learning techniques for Process Discovery [21] of declarative models, LTL$_p$ turns out to be a better specification language than LTL$_f$.

2. LTL on Process Traces

Given a set $\Sigma$ of propositional symbols, also called activities, a process trace $\pi$ is a finite non-empty sequence of activities of $\Sigma$, i.e. $\pi \in \Sigma^+$. An LTL$_p$ formula $\varphi$ over $\Sigma$ is defined by the following grammar:

$$\varphi ::= a | \neg \varphi | (\varphi \& \varphi) | (\varphi | \varphi) | (\varphi \rightarrow \varphi) | X(\varphi) | WX(\varphi) | G(\varphi) | F(\varphi) | \varphi U \varphi | \varphi R \varphi,$$

where $a \in \Sigma$; $X$(next), $WX$(weak next), $G$(globally), $F$(eventually), $U$(until), $R$(release) are the temporal operators; and $\neg$(negation), $\&$(conjunction), $\mid$(disjunction), $\rightarrow$(implication) are the classical Boolean operators. Note that we do not require formulae to be in negation normal form (i.e. we allow negation to be in front of any formula) and therefore some operators could be defined in terms of the others. However, we still list them here to make the grammar match the syntax of LTLp2DFA.

Due to space limitations, we do not report the semantics here. We just observe that it is formally analogous to the semantics of LTL$_f$ (once process traces are considered instead of finite traces) and we refer to [16] for further details.
The following theorem establishes a connection between formulae in LTL$p$ and finite-state automata.

**Theorem.** Given an LTL$p$ formula $\varphi$ over $\Sigma$, there exists a DFA $\mathcal{A}_\varphi = (\Sigma, Q, q_0, \delta, F)$ such that $\mathcal{A}_\varphi$ accepts exactly the process traces satisfying $\varphi$.

Note that the alphabet of the automaton $\mathcal{A}_\varphi$ coincides with the set of activities $\Sigma$, while working with LTL$f$ would require an exponentially larger alphabet (the power set $2^\Sigma$). The automaton $\mathcal{A}_\varphi$ can indeed be obtained following the LTLf2NFA algorithm reported in [9] considering in the construction of the transition function only singleton interpretations, i.e. propositional interpretations that are singletons (and determinizing the obtained automaton).

### 3. Overview of LTLp2DFA

The tool is written in Python and is built on top of the FLLOAT library, simplifying the returned automata to take into account only singleton interpretations. LTLp2DFA is available as a capsule at https://codeocean.com/capsule/2735129/tree/v1 and can be run in the cloud. The source code is also available at https://github.com/fracchiariello/LTLp2DFA, together with a tutorial (an Interactive Python Notebook) and a video demonstration.

Let us consider again the template $\textit{Response}(a, b)$. It corresponds to the LTL$p$ formula $\varphi_1 = G(a \rightarrow F(b))$, or equivalently, $\varphi_2 = G(a \rightarrow X(F(b)))$. The automaton obtained with LTLp2DFA is the same for both formulae and is reported in Figure 1 (a). Compare this automaton with the ones returned by FLLOAT on $\varphi_1$ (b) and $\varphi_2$ (c). To compactly represent the automaton, FLLOAT’s output is a symbolic automaton where, instead of propositional interpretations, the transitions are labeled by propositional formulae. The meaning is that when reading an interpretation, the transition labelled with the formula satisfied by the interpretation is followed. Our tool exploits instead the simplicity assumption and the transitions are directly labeled with activities. Note that $a$ and $b$ are variables and the automaton is associated with the template. For a particular constraint, $a$ matches the activation activity and $b$ the target activity. A special symbol * is then added that matches any other activity. The same trick can be applied to improve the simplicity assumption for LTL$f$. The result of adding the (improved) simplicity assumption to $\varphi_1$ or, equivalently, to $\varphi_2$ is in (d). The effect of the assumption is that a sink state is introduced that is reached when zero, two or more activities are executed at a time. Regarding the other transitions, the formulae are just an (involved) way of listing the corresponding activities.

### 4. Conclusion

We have provided a tool to convert LTL$p$ formulae to finite-state automata. The automata representation makes it easier to check the conformance of processes specified by such formulae with event logs. Thus, LTLp2DFA paves the way for the practical use of LTL$p$ as a process specification language. We have also shown that, being the logic tailored to BPM and PM applications, it is a better choice (in terms of simplicity and performance) than LTL$f$. Therefore,

---

1https://github.com/whitemech/flloat
the tool enables LTL_p to potentially replace LTL_f (in the same way LTL_f replaced LTL), for any such application. Since LTL_p is more general than DECLARE (being able to express the same process-trace properties as LTL_f), the tool could be easily embedded in Declare4Py [22], the reference Python tool for DECLARE-based PM, to support all the tasks involving automata-based checking like, for example, process discovery, conformance checking and log generation.

5. Acknowledgments

Work partly supported by ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the UNIBZ project CAT, the PNRR MUR project PE0000013-FAIR, the PRIN project RIPER (No. 20203FFYLK), and the project "Borgo 4.0" POR Campania FESR 2014-2020.

References