From LTL on Process Traces to Finite-State Automata

Francesco Chiariello[1,2,*], Fabrizio Maria Maggi[3] and Fabio Patrizi[1]

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

Abstract

Linear Temporal Logic on process traces (or LTL$_p$) 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 LTL$_p$ 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 (LTL$_f$) [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, LTL$_f$ 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 LTL$_f$ 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 LTL$_f$ as

21st International Conference on Business Process Management, Utrecht, The Netherlands, September 11-15, 2023

*Corresponding author.
chiariello@diag.uniroma1.it (F. Chiariello); maggi@inf.unibz.it (F. M. Maggi); patrizi@diag.uniroma1.it (F. Patrizi)
https://www.francescochiariello.me/ (F. Chiariello)
0000-0001-7855-7480 (F. Chiariello); 0000-0002-9089-6896 (F. M. Maggi); 0000-0002-9116-251X (F. Patrizi)

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 Σ of propositional symbols, also called activities, a *process trace* π is a finite non-empty sequence of activities of Σ, i.e. $\pi \in \Sigma^+$.

An LTL$_p$ formula φ over Σ is defined by the following grammar:

$$\varphi ::= a \mid {\sim}\varphi \mid (\varphi \& \varphi) \mid (\varphi | \varphi) \mid (\varphi\text{->}\varphi) \mid \mathbf{X}(\varphi) \mid \mathbf{WX}(\varphi) \mid \mathbf{G}(\varphi) \mid \mathbf{F}(\varphi) \mid \varphi\mathbf{U}\varphi \mid \varphi\mathbf{R}\varphi,$$

where $a \in \Sigma$; $\mathbf{X}$(next), $\mathbf{WX}$(weak next), $\mathbf{G}$(globally), $\mathbf{F}$(eventually), $\mathbf{U}$(until), $\mathbf{R}$(release) are the temporal operators; and ~(negation), &(conjunction), |(disjunction), −>(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* φ *over* Σ*, there exists a DFA* $\mathcal{A}_\varphi = (\Sigma, Q, q_0, \delta, F)$ *such that* $\mathcal{A}_\varphi$ *accepts exactly the process traces satisfying* φ*.*

Note that the alphabet of the automaton $\mathcal{A}_\varphi$ coincides with the set of activities Σ, while working with LTL_f would require an exponentially larger alphabet (the power set 2^Σ). 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[1], 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 $Response(a, b)$. It corresponds to the LTL_p formula $\varphi_1 = \mathbf{G}(a\text{->}\mathbf{F}(b))$, or equivalently, $\varphi_2 = \mathbf{G}(a\text{->}\mathbf{X}(\mathbf{F}(b)))$. 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 φ_1 (b) and φ_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 φ_1 or, equivalently, to φ_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,

[1]https://github.com/whitemech/flloat

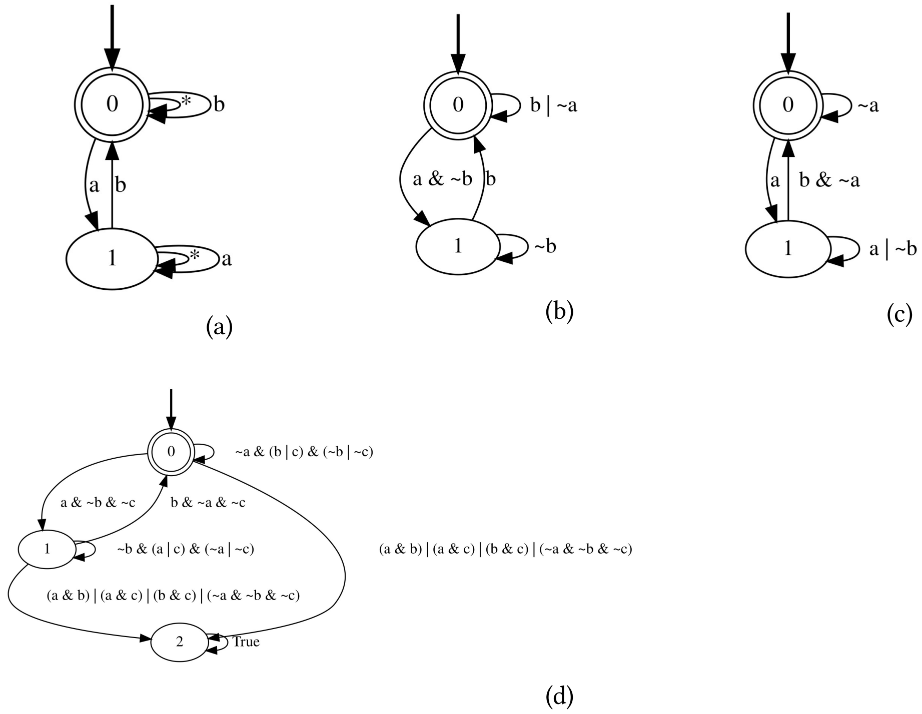

Figure 1: Automata for the *Response* template: (a) using LTL_p, (b) and (c) using LTL_f, (d) using LTL_f with simplicity assumption.

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

[1] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113.

[2] W. M. P. van der Aalst, The application of Petri nets to workflow management, J. Circuits Syst. Comput. 8 (1998) 21–66.
[3] W. M. van der Aalst, C. Stahl, Modeling business processes - a Petri net-oriented approach, in: CoopIS series, 2011.
[4] S. A. White, Introduction to BPMN, Ibm Cooperation 2 (2004) 0.
[5] T. Allweyer, BPMN 2.0 : introduction to the standard for business process modeling, 2016.
[6] G. De Giacomo, M. Y. Vardi, Linear Temporal Logic and Linear Dynamic Logic on finite traces, in: IJCAI, IJCAI/AAAI, 2013, pp. 854–860.
[7] F. Belardinelli, A. Lomuscio, A. Murano, S. Rubin, Alternating-time temporal logic on finite traces, in: IJCAI, ijcai.org, 2018, pp. 77–83.
[8] A. Murano, M. Parente, S. Rubin, L. Sorrentino, Model-checking graded computation-tree logic with finite path semantics, Theor. Comput. Sci. 806 (2020) 577–586.
[9] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on LTL on finite traces: Insensitivity to infiniteness, in: AAAI, AAAI Press, 2014, pp. 1027–1033.
[10] A. Pnueli, The temporal logic of programs, in: FOCS, 1977, pp. 46–57.
[11] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: Full support for loosely-structured processes, in: EDOC, 2007, pp. 287–300.
[12] G. De Giacomo, F. M. Maggi, A. Marrella, F. Patrizi, On the disruptive effectiveness of automated planning for LTLf-based trace alignment, in: AAAI, AAAI Press, 2017, pp. 3555–3561.
[13] G. De Giacomo, R. De Masellis, F. M. Maggi, M. Montali, Monitoring constraints and metaconstraints with temporal logics on finite traces, ACM Trans. Softw. Eng. Methodol. 31 (2022) 68:1–68:44.
[14] F. M. Maggi, M. Montali, C. Di Ciccio, J. Mendling, Semantical vacuity detection in declarative process mining, in: BPM, 2016.
[15] A. Cecconi, C. Di Ciccio, A. Senderovich, Measurement of rule-based LTLf declarative process specifications, in: ICPM, 2022, pp. 96–103.
[16] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623.
[17] F. Chiariello, F. M. Maggi, F. Patrizi, ASP-based declarative process mining, in: AAAI, AAAI Press, 2022, pp. 5539–5547.
[18] F. Chiariello, F. Maggi, F. Patrizi, ASP-based declarative process mining (extended abstract), in: (ICLP), Electronic Proceedings in Theoretical Computer Science (EPTCS), 2022.
[19] F. Chiariello, F. M. Maggi, F. Patrizi, A tool for compiling declarative process mining problems in ASP, Softw. Impacts 14 (2022) 100435.
[20] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.
[21] S. Agostinelli, F. Chiariello, F. M. Maggi, A. Marrella, F. Patrizi, Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis, Inf. Syst. 114 (2023) 102180.
[22] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4py: A Python library for declarative process mining, in: BPM Demos, volume 3216 of *CEUR Workshop Proceedings*, 2022, pp. 117–121.