EasyChair Preprint

№ 8675

ASP-Based Declarative Process Mining

Francesco Chiariello, Fabrizio Maria Maggi and Fabio Patrizi

August 12, 2022

ASP-Based Declarative Process Mining (Extended Abstract)

Francesco Chiariello
DIAG
Sapienza University of Rome
chiariello@diag.uniroma1.it

Fabrizio Maria Maggi
KRDB
Free University of Bozen-Bolzano
maggi@inf.unibz.it

Fabio Patrizi
DIAG
Sapienza University of Rome
patrizi@diag.uniroma1.it

We propose Answer Set Programming (ASP) as an approach for modeling and solving problems from the area of Declarative Process Mining (DPM). We consider here three classical problems, namely, Log Generation, Conformance Checking, and Query Checking. These problems are addressed from both a *control-flow* and a *data-aware* perspective. The approach is based on the representation of process specifications as (finite-state) automata. Since these are strictly more expressive than the de-facto DPM standard specification language DECLARE, more general specifications than those typical of DPM can be handled, such as formulas in linear-time temporal logic over finite traces. (Full version available in the Proceedings of the 36th AAAI Conference on Artificial Intelligence [4]).

1 Contribution

Process Mining (PM) [1] is a research area concerned with analyzing *event logs* stored by (enterprise) information systems, with the aim of better understanding the (business) processes that produce the logs and how they are enacted in the organization owning the systems. A *process* can be thought of as a set of event sequences that reach a desired goal, whose observed sequences, called *process traces*, constitute the event log. Different formalisms have been proposed by the Business Process Management (BPM) community for modeling processes, including Petri nets and BPMN [10]. In Declarative Process Mining (DPM) [2] process models are specified declaratively, in a constraint-based fashion, i.e., as sets of constraints, which must be satisfied during the process execution. This type of specifications is particularly suitable for knowledge intensive processes [5] that include a large variety of behaviors and can be more compactly represented under an open world assumption (all behaviors that do not violate the constraints are allowed). Typical process modeling languages for specifying processes in a declarative way are DECLARE [2] and LTL$_f$ [6].

We consider three classical problems from (D)PM:

- *Log generation* [9], i.e., the problem of generating a set of traces of a given length, compliant with a process model;
- *Conformance checking* [3], i.e., the problem of checking whether the traces of a log are compliant with a process model;
- *Query checking* [8], i.e., the problem of finding properties of (the process associated to) a log by checking its conformance with the instantiation of candidate template formulas (also called *queries*).

In these problems, *traces* are finite sequences of *events*, which represent activity executions. *Activities* model the atomic operations a process may perform and include *attributes*, which events instantiate at execution time with specific values.

The *control-flow* perspective focuses only on process activities, and the specification languages used in this setting include DECLARE or the more general LTL$_f$. In the *data-aware* perspective, instead, also attributes and their values, i.e., the data, are taken into account, and richer logics are used, such as MP-DECLARE [3] or LTL$_f$ with local conditions (L-LTL$_f$); this is the setting adopted in the paper.

To solve the problems of our interest, we exploit the fact that, analogously to the case of LTL$_f$, for every L-LTL$_f$ formula, there exists a finite-state automaton (FSA) accepting exactly the traces that satisfy the formula. While every problem has its own specific features, they all share some common parts. As a consequence, the encoding approach of each problem includes the following steps:

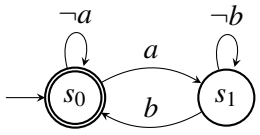
- construction of the FSAs corresponding to each L-LTL$_f$ input specification;
- definition of an encoding schema for FSAs, as a set of ASP rules;
- definition of an encoding schema for traces, as a set of ASP facts;
- definition of ASP rules to simulate the execution of an FSA on a trace;
- definition of ASP generation and test rules (when needed) for the specific problem.

For Log Generation, the generation rules require that each position (up to the specified length) contains exactly one activity whose attributes are assigned one value from their respective domain. Then, test rules check whether the FSAs corresponding to the process model accept the generated trace. Conformance Checking is even simpler since traces are given. To deal with many traces, we associate an index to each of them and then add a new argument to predicates, which represents the index. Then, for every pair of index and FSA, we check whether the FSA accepts the trace identified by the index. For Query Checking, an instantiation of the variables in the template formula is first guessed, in order to obtain a proper L-LTL$_f$ formula. Then, all traces are checked against the FSA corresponding to the so-obtained formula. Special care is required in the ASP encoding to deal with formula variables; to this end, we introduce suitable predicates modeling their instantiation.

For all problems, we provide the ASP encoding, an experimental evaluation and, when possible, a comparison with SoA tools. The obtained results show that our approach considerably outperforms the SoA MP-DECLARE *Log Generator* [9] based on Alloy [7] (in turn based on SAT solvers), which does not exploit the FSA representation of formulas. Slightly worse results are obtained for Conformance Checking wrt the DECLARE *Analyzer* [3], which is however tailored to DECLARE and, differently from the ASP-based approach, cannot be used to check general LTL$_f$ formulas. As to Query Checking, this is the first solution in a data-aware setting, thus, while the experiments show the feasibility of the approach, no comparison with other tools is carried out. The ASP implementation of the approach is not optimized, as the focus of the work is on feasibility rather than performance, so further improvements are possible.

As an example of FSA encoding, consider the (control-flow) *response* constraint $\varphi = \mathbf{G}(a \rightarrow \mathbf{F}b)$ saying that "whenever a occurs, it must be eventually followed by b".[1] The FSA shown in Figure 1 accepts all and only the traces that satisfy φ. The corresponding ASP representation is reported next to the automaton. Predicates `init` and `accepting` are used to model initial and accepting states, respectively. Predicate `trans` models state transitions and are labeled each with a unique integer, identifying the transition. Finally, predicate `hold` models which transitions are enabled by the trace event at time T. The automaton execution on the input trace is then simulated by reading the trace event by event, with the automaton starting in its initial state, and then following, at every step, the transition enabled by the read event. If the formulae labeling the transitions express more complex properties involving data conditions, these can be modeled in the body of the rules for predicate `hold`. For example, the following

[1] Differently from the generic traces usually considered with LTL$_f$, here two activities cannot be true at the same time.



$\texttt{init}(s_0).$
$\texttt{accepting}(s_0).$
$\texttt{trans}(s_0,1,s_1).$
$\texttt{hold}(1,T) \leftarrow \texttt{trace}(a,T).$
$\texttt{trans}(s_1,2,s_0).$
$\texttt{hold}(2,T) \leftarrow \texttt{trace}(b,T).$
$\texttt{trans}(s_0,3,s_0).$
$\texttt{hold}(3,T) \leftarrow \texttt{not hold}(1,T),\texttt{time}(T).$
$\texttt{trans}(s_1,4,s_1).$
$\texttt{hold}(4,T) \leftarrow \texttt{not hold}(2,T),\texttt{time}(T).$

Figure 1: FSA of formula $\varphi = \mathbf{G}(a \rightarrow \mathbf{F}b)$ and its ASP representation.

rule expresses that transition 1 is enabled at time T if a occurs at T, with a value less than 5 for attribute *number*: $\texttt{hold}(1,T) \leftarrow \texttt{trace}(a,T),\texttt{has\_val}(\textit{number},V,T),V < 5$.

By showing how to handle LTL_f specifications, namely by exploiting the FSA representation for reducing problems to reachability of accepting states, we have paved the way for the use of ASP as a solving approach to DPM problems and, potentially, to all problems involving temporal specifications over finite traces.

Acknowledgements Work partly supported by: ERC Advanced Grant WhiteMech (No. 834228), EU ICT-48 2020 project TAILOR (No. 952215), Sapienza Project DRAPE, UNIBZ project CAT.

References

[1] Wil M. P. van der Aalst (2016): *Process Mining - Data Science in Action, Second Edition*. Springer, doi:10.1007/978-3-662-49851-4.

[2] Wil M. P. van der Aalst, Maja Pesic & Helen Schonenberg (2009): *Declarative workflows: Balancing between flexibility and support*. *Comput. Sci. Res. Dev.* 23(2), pp. 99–113, doi:10.1007/s00450-009-0057-9.

[3] Andrea Burattin, Fabrizio Maria Maggi & Alessandro Sperduti (2016): *Conformance checking based on multi-perspective declarative process models*. *Expert Syst. Appl.* 65, doi:10.1016/j.eswa.2016.08.040.

[4] Francesco Chiariello, Fabrizio Maria Maggi & Fabio Patrizi (2022): *ASP-Based Declarative Process Mining*. In: *Proc. of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*.

[5] Claudio Di Ciccio, Andrea Marrella & Alessandro Russo (2015): *Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches*. *J. Data Semant.*

[6] Giuseppe De Giacomo & Moshe Y. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In: *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence*, IJCAI/AAAI.

[7] Daniel Jackson (2012): *Software Abstractions: logic, language, and analysis*. MIT Press.

[8] Margus Räim, Claudio Di Ciccio, Fabrizio Maria Maggi, Massimo Mecella & Jan Mendling (2014): *Log-Based Understanding of Business Processes through Temporal Logic Query Checking*. In: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, doi:10.1007/978-3-662-45563-0_5.

[9] Vasyl Skydanienko, Chiara Di Francescomarino, Chiara Ghidini & Fabrizio Maria Maggi (2018): *A Tool for Generating Event Logs from Multi-Perspective Declare Models*. In: *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*.

[10] Mathias Weske (2012): *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, doi:10.1007/978-3-642-28616-2.