Software Impacts 14 (2022) 100435

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

A tool for compiling Declarative Process Mining problems in ASP

Francesco Chiariello [a,*], Fabrizio Maria Maggi [b], Fabio Patrizi [a]

[a] *Sapienza University of Rome, Italy*
[b] *Free University of Bozen-Bolzano, Italy*

ARTICLE INFO

Keywords:
Process Mining
Business Process Management
Answer Set Programming

ABSTRACT

We present a tool for compiling three problems from the Process Mining community into Answer Set Programming: Log Generation, Conformance Checking, and Query Checking. For each problem, two versions are addressed, one considering only the control-flow perspective and the other considering also the data perspective. The tool can support companies in analyzing their business processes; it is highly flexible and general, and can be easily modified to address other problems from Declarative Process Mining.

Code metadata

Current code version	*v1*
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2022-176
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/1375188/tree/v1
Legal Code License	Apache-2.0
Code versioning system used	git
Software code languages, tools, and services used	Python, clingo
Compilation requirements, operating environments & dependencies	
If available Link to developer documentation/manual	
Support email for questions	chiariello@diag.uniroma1.it

1. Introduction

Process Mining (PM) is a research area at the intersection of Business Process Management (BPM) and Data Mining. It aims to get insights into business processes by analyzing event logs stored by enterprise information systems. An *event log* is a collection of finite sequences – or *traces* – of activities, each handling a different case. Such traces may contain additional data attached to activities, in the form of *attribute values*. We talk about *control-flow* perspective when focusing on the activities occurring in a trace, and about *data* perspective when also considering activity attributes and their respective values. A *process model* is a specification of a process, whose executions generate traces of interest. Such models can be provided in the form of executable structures, such as Petri Nets or BPMN, or as formulas in a formal language, such as DECLARE [1] or LTL_f [2], as in Declarative Process Mining (DPM).

We address three problems from DPM. *Log Generation* is the problem of generating logs consistent with an input process model, which possibly satisfy further requirements, such as trace length. This problem arises from the need for controlling data with the aim of experimentally testing and validating specific aspects of PM techniques. *Conformance Checking* is the problem of checking whether the traces of a log conform to an input process model. This allows for checking whether a process execution behaves as expected or not. It is indeed common that some traces in an event log deviate from their nominal behavior, as the results, e.g., of manual activities. Finally, *Query Checking* is the problem of discovering some properties of a process, by checking candidate template properties against the traces in the process event log. This allows, e.g., for discovering a (possibly partial) process specification.

We have devised a tool for compiling instances of these three problems into Answer Set Programming. Our implementation can cope with both the control-flow and the data perspectives.

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: chiariello@diag.uniroma1.it (F. Chiariello), maggi@inf.unibz.it (F.M. Maggi), patrizi@diag.uniroma1.it (F. Patrizi).

https://doi.org/10.1016/j.simpa.2022.100435
Received 5 October 2022; Received in revised form 23 October 2022; Accepted 25 October 2022

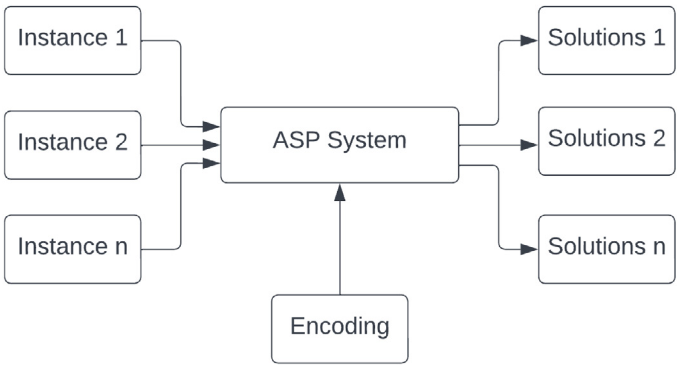

Fig. 1. Schematic overview of the approach.

Answer Set Programming (ASP) [3] is a declarative approach for solving combinatorial search problems. In ASP, a problem is modeled as a *logic program*, which is essentially a logical theory, and its solution(s) correspond to the *answer sets*, i.e., a specific class of models, of the program itself. Answer sets can be obtained by feeding an ASP system, such as `clingo`[1] or `dlv`,[2] with the logic program (see Fig. 1).

In addition, we also provide process models and event logs for testing the tool, which consists of a set of scripts for converting each problem instance into an ASP program.

2. Description

As common in Declarative Problem Solving, to guarantee elaboration tolerance [4] we split a problem into two parts: the problem class and the problem instance. We encode each problem class considered here (i.e., Log Generation,[3] Conformance Checking,[4] and Query Checking[5]) into an ASP program. Furthermore, we provide scripts for converting process models (represented as `.decl` file) and event logs (represented as `.xes` files), representing, together, the problem instance, into ASP. We then input the problem class and the problem instance to the ASP system `clingo`, which returns the solutions to the original problem. Notice that an instance may have zero, one, or many solutions. In the case of Log Generation, we also provide a script for converting the obtained answer sets into a `.xes` file, i.e., the standard representation formalism for event logs.

Finally, observe that while we take as input a `.decl` file, thus handling only DECLARE constraints, one can in principle consider any LTL_f formula [2], by simply converting it into an automaton (using one of the available tools like Lydia[6] or LTLf2DFA[7][8] and then representing it in ASP in the same way we handle DECLARE.

As a usage example, consider the files contained in https://github.com/fracchiariello/process-mining-ASP and Conformance Checking. This problem takes as input a process model and an event log. We can solve the problem by providing `clingo` the corresponding input files, together with the ASP encoding of Conformance Checking:

```
clingo conformance_encoding.lp 2012_80.lp BPI_Challenge_2012.asp templates.lp
```

where: `BPI_Challenge_2012.asp` is the event log (taken from the Business Process Intelligence Challenge, BPI 2012,[9]) `2012_80.lp` is the process model, and `templates.lp` contains the automata of the DECLARE templates.

Scripts are written in Python and use only standard libraries, with the exception of `pm4py`,[10] here used to read `.xes` files. For the requirements of the ASP system used we invite the reader to check the relative page.

3. Impact

The software has been used in [5], where it has been tested on synthetic and real-life logs taken from various BPI challenges.

The tool for Log Generation has been integrated into RuM[11] [6], a state-of-the-art desktop application for solving several PM tasks. Such integration has provided advantages to both our tool and RuM. From the perspective of our (command-line) tool, RuM brings the benefit of an intuitive GUI, which greatly improves usability. From the perspective of RuM, our tool facilitates the extension of the Log Generation functionality to constraints beyond those available in DECLARE. Indeed, in the previous version, such constraints had to be specified in a counterintuitive way by resorting to the Alloy[12] [7] language, which is essentially First-order logic; with our tool, instead, they can be specified directly in LTL_f, which provides a much more natural way of expressing them. More importantly, with the newly integrated ASP-based approach, RuM exhibits significantly better time performance.

Concerning Conformance Checking, compared with the state-of-the-art tool Declare Analyzer [8], our approach exhibits slightly worse execution-time performance but has the advantage of greater flexibility. Indeed, Declare Analyzer is an ad-hoc tool specifically tailored for the limited set of constraints provided by DECLARE, while our tool does not suffer from such a limitation and can deal with any constraint expressed in LTL_f.

Wrt the Query Checking problem, since this problem cannot be currently solved by existing tools, our Query Checking tool can be used as a basis to extend state-of-the-art PM tools (e.g., RuM or ProM[13]) with this novel process analysis functionality.

As for potential applications, since event logs are the basis for any PM algorithm, our Log Generation tool can be used for testing other PM techniques in a controlled environment, i.e., where specific features of the input log can be tuned as desired. The Conformance Checking and Query Checking tools could instead be used by companies interested in checking whether their event logs adhere to a process model, or finding the properties of the process underlying an event log.

[1] https://github.com/potassco/clingo
[2] https://dlv.demacs.unical.it/home
[3] https://codeocean.com/capsule/8240890/tree/v1
[4] https://codeocean.com/capsule/2601782/tree/v1
[5] https://codeocean.com/capsule/0075725/tree/v1
[6] https://github.com/whitemech/lydia
[7] https://github.com/whitemech/LTLf2DFA
[8] http://ltlf2dfa.diag.uniroma1.it/
[9] https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204
[10] https://pm4py.fit.fraunhofer.de/
[11] https://rulemining.org/
[12] https://alloytools.org/
[13] https://www.promtools.org/doku.php

Finally, we observe that Conformance Checking is an essential building block for many fundamental PM problems, which ultimately reduce to checking conformance of an event log wrt to a process model; moreover, our approach being declarative, the encoding can be easily modified and tailored for other problems. Therefore, in the future, our tool may be used as an essential component to develop new approaches for more complex problems, such as Process Discovery and Process Model Repair [9].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Work partly supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the EU ICT-49 2021 project AIPlan4EU (No. 101016442), and the UNIBZ, Italy project CAT.

References

[1] Wil M.P. van der Aalst, Maja Pesic, Helen Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2) (2009) 99–113.

[2] Giuseppe De Giacomo, Moshe Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI/AAAI, 2013.

[3] Ilkka Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (3–4) (1999) 241–273.

[4] John McCarthy, Elaboration tolerance, in: Progress, 1999.

[5] Francesco Chiariello, Fabrizio Maria Maggi, Fabio Patrizi, ASP-based declarative process mining, in: Proc. of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022, 2022.

[6] Anti Alman, Claudio Di Ciccio, Dominik Haas, Fabrizio Maria Maggi, Alexander Nolte, Rule mining with RuM, in: Boudewijn F. van Dongen, Marco Montali, Moe Thandar Wynn (Eds.), 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020, IEEE, 2020, pp. 121–128.

[7] Daniel Jackson, Software Abstractions: Logic, Language, and Analysis, MIT Press, 2012.

[8] Andrea Burattin, Fabrizio M. Maggi, Alessandro Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Syst. Appl. 65 (2016) 194–211.

[9] Dirk Fahland, Wil M.P. van der Aalst, Model repair - aligning process models to reality, Inf. Syst. 47 (2015) 220–243.