

ASP-Based Log Generation with Purposes in DECLARE4Py

Ivan Donadello^{1,*}, Fabrizio Maria Maggi¹, Francesco Riva² and Manpreet Singh³

¹Free University of Bozen-Bolzano, Bolzano, Italy

²Datalane SRL, Verona, Italy

³Wuerth Italia Srl, Egna, Italy

Abstract

Process mining techniques are meant to extract non-trivial information from complex data. Controlled experiments of the algorithms underlying process mining techniques often require logs of process executions that fit the specific purposes of each specific test. Therefore, many tools for the log generation from both procedural models (e.g., Petri nets or BPMN models) and declarative models (e.g., based on LTL_f or DECLARE) have been developed. However, the log generation from declarative models still lacks tools for log generation that address specific purposes such as the specification of trace length distributions, the setting of the number of variants that should appear in the log, or the specification of the number of activations of a constraint that should be contained in a trace. We address this research gap by proposing an extension of the DECLARE4Py Python library that generates synthetic event logs using an Answer Set Programming-based solution whose flexibility supports the encoding of specific purposes.

Keywords

Process Mining, Declarative Models, Log Generation, Answer Set Programming

1. Introduction


Process Mining (PM) is a research area that analyzes the execution data of a business process (an event log) to extract useful information for process improvement. This is not a trivial task as event logs are sets of process instances (a.k.a. traces) that are a complex type of data. A trace is composed of a sequence of events arranged in chronological order and each event contains a set of attributes that can be both symbolic (e.g., the name of the activity executed or the involved resource) and numeric (e.g., a timestamp). In addition, process instances are samples of a process model, that is, background knowledge that constraints the execution order of the events. This background knowledge can be expressed with procedural models (e.g., Petri nets or BPMN models), which specify the exact control flow of the activities in a trace, or with declarative models (that is, constraints expressed in Linear Temporal Logic on finite traces or DECLARE [1]) that specify the constraints over process activities that should be satisfied during the process


ICPM Doctoral Consortium and Demo Track 2023

*Corresponding author.

✉ ivan.donadello@unibz.it (I. Donadello); maggi@inf.unibz.it (F. M. Maggi); f.riva@datalane.nl (F. Riva); mani.sw.dev@gmail.com (M. Singh)

ORCID 0000-0002-0701-5729 (I. Donadello); 0000-0002-9089-6896 (F. M. Maggi)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

execution. The former models are more fine-grained, the latter more coarse-grained but more flexible.

The controlled evaluation of PM algorithms requires event logs that fit the purposes for which each specific experiment has been designed. For example, one purpose can be to test how the performance of an algorithm is affected when the distribution of trace lengths in the event log varies. However, although several tools have been developed for generating event logs by simulating (declarative and procedural) process models [2, 3, 4, 5] only PURPLE (a tool presented in [4]) produces event logs that fulfill a given property/purpose. Since PURPLE uses procedural process models to generate logs, a purpose-guided log generator for declarative models is still missing. To fill this gap, we extend the DECLARE4PY [6] Python library, which implements classical PM tasks starting from the MP-DECLARE [7] language, with an Answer Set Programming (ASP) functionality for generating purpose-guided event logs starting from declarative models [8]. This ASP-based method performs the simulation of an input declarative model by converting its associated deterministic finite automaton (DFA) into a logical program whose solution is given by an ASP solver. This solution is extremely flexible as it supports the encoding of the desired purposes as a logical program on top of the DFA encoding.

2. Log Generation with DECLARE4PY

The ASP-based solution adopted in DECLARE4PY takes as inputs the number N of traces to generate, the minimum length m and the maximum length n of a trace (i.e., the number of events in the trace), a process model containing MP-DECLARE constraints, and encodes these inputs in an ASP program to be solved by the Clingo solver (<https://potassco.org/>). The solution is a set of N traces that satisfy the input model and the minimum/maximum length specification. We leverage this solution to generate event logs with the following four purposes as additional inputs.

Users can specify a **trace length distribution**, that is, an input probability distribution on the lengths of the traces in the log. DECLARE4PY supports three probability distributions: i) the *custom distribution*, where the user has to specify the probability of a generated trace to have length k for each $k \in [m, m + 1, \dots, n]$; ii) the *uniform distribution* where all the trace lengths have the same probability to appear, i.e., $1/(n - m + 1)$; iii) the *Gaussian distribution* where the user has to specify a mean and a variance for sampling the trace lengths from a Gaussian distribution. Once k has been sampled, the Clingo solver is called $n - m + 1$ times for each trace length k .

Users can specify the **number of variants** V , so the generated event log can be segmented into V groups of traces having the same control-flow (i.e., the same sequence of activity names) but that can differ for other attributes, such as the timestamps or the resources. Here, Clingo is called different times: in the first call, V traces are generated representing the variants; then, for each variant, Clingo is called to generate traces with the control-flow of the variant. In this second call, the input ASP program also encodes the control-flow of the variant.

Users can specify a subset of constraints in the input model to generate **negative traces**, that is, traces that do not satisfy (at least one of/all) the constraints in the specified subset. This allows users to obtain event logs with labelings defined by MP-DECLARE constraints. Such

labeled traces can be used to train and test Machine Learning-based process mining algorithms. The positive traces satisfy all constraints in the input model, whereas the negative traces do not satisfy the specified subset of constraints. Also in this case, the subset of constraints to be violated is encoded in an ASP program.

Users can specify the **number of activations** for a subset of constraints in the input model. For example, the user can specify for the constraint `Response[(CRP, J), ReleaseB]` an activation number of 3 that means that the event with activity name CRP with payload J should occur three times. The user provides as input a range of possible values for the number of activations for a specific constraint and then `DECLARE4PY` randomly selects a value in that range for each generated trace. The subset of the constraints for which the number of activations is specified and the user-defined value range for the number of activations are encoded in ASP.

3. Tool Maturity

The better computational times of the adopted ASP solution with respect to the state-of-the-art (declarative) log generators have already been shown in [8]. Therefore, we performed a comparison in terms of diversity of the generated traces between our tool and the Alloy-based tool presented in [5]. We chose this tool as it is the only one available that generates synthetic logs starting from MP-DECLARE models. The tool generates the traces by using a SAT solver (instead of an ASP solver as done in `DECLARE4PY`) starting from a representation of the input MP-DECLARE model in Alloy (<https://alloytools.org/>). The diversity is measured with the average of the syntactic distances among the traces in the generated event log. A higher average distance indicates a higher diversification of the generated traces making the event log a more interesting benchmark for testing purposes. As syntactic distance, we considered the normalized Damerau-Levenshtein Distance (DLD). We considered the activity names and resources available in the *Sepsis* log¹ (13 activity names, 26 resources), in the *BPIC15_4* log² (87 activity names, 10 resources) and in the Road Traffic Fine Management log³ (*RTFM*, 10 activity names, 143 resources), in order to define three MP-DECLARE models. In each model, 5 constraints were defined, and the model was used to generate a log with 100 traces of lengths ranging from 10 to 20 events. For each synthetic log, we measured the DLD between all pairs of synthetic traces and computed the average. The DLD was measured on both the control-flow and the resources (see Table 1). We can notice that `DECLARE4PY` has better performance than the Alloy-based tool, that is, `DECLARE4PY` guarantees a higher variability in both control-flow and resources. Finally, the log generator in `DECLARE4PY`, being purpose-guided, represents an improvement of the existing tools for log generation also in terms of provided functionalities.

¹10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

²10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

³10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

Table 1

Average of the normalized DLD for different logs generated using DECLARE4Py and Alloy.

Dataset	DECLARE4Py		Alloy-based	
	Res. Flow	Ctrl. Flow	Res. Flow	Ctrl. Flow
<i>Sepsis</i>	0.565	0.409	0.027	0.028
<i>BPIC15_4</i>	0.631	0.798	0.126	0.050
<i>RTFM</i>	0.525	0.363	0.366	0.308

4. Screencast and Website

The GitHub repository of DECLARE4Py⁴ contains the source code of the tool and all the tutorials. A specific tutorial for the ASP-based log generator⁵ shows how to run the log generator using all the available options. The video presentation of this paper can be accessed at <https://www.dropbox.com/scl/fi/cbihgbw34smkib7u1sry/Screen-Recording-2023-09-12-at-19.17.17.mov?rlkey=pvzt6cuj5yk98azi611zd79xy&dl=0>.

References

- [1] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: Full support for loosely-structured processes, in: EDOC, IEEE Computer Society, 2007, pp. 287–300.
- [2] A. Burattin, PLG2: Multiperspective process randomization with online and offline simulations, in: BPM (Demos), volume 1789 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 1–6.
- [3] C. Di Ciccio, M. L. Bernardi, M. Cimitile, F. M. Maggi, Generating event logs through the simulation of Declare models, in: EOMAS@CAiSE, volume 231 of *Lecture Notes in Business Information Processing*, Springer, 2015, pp. 20–36.
- [4] A. Burattin, B. Re, L. Rossi, F. Tiezzi, PURPLE: A PURPose-guided Log GENERator (Extended Abstract), in: ICPM Doctoral Consortium / Demo, volume 3299 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 90–94.
- [5] V. Skydanienco, C. Di Francescomarino, C. Ghidini, F. M. Maggi, A tool for generating event logs from multi-perspective Declare models, in: BPM (Dissertation/Demos/Industry), *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 111–115.
- [6] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4Py: A Python library for declarative process mining, in: BPM (Demos), *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 117–121.
- [7] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, *Expert Syst. Appl.* 65 (2016) 194–211.
- [8] F. Chiariello, F. M. Maggi, F. Patrizi, ASP-based declarative process mining, in: AAAI, AAAI Press, 2022, pp. 5539–5547.

⁴<https://github.com/ivanDonadello/Declare4Py>

⁵https://github.com/ivanDonadello/Declare4Py/blob/main/docs/source/tutorials/9.Log_Generation.ipynb