ASP for Process Mining: Some Possible Future Lines of Research*

Mario Alviano[1,*,†], Valeria Fionda[1,*,†], Antonio Ielo[1,*,†] and Francesco Ricca[1,*,†]

[1]Department of Mathematics and Computer Science, University of Calabria

Abstract

Answer Set Programming (ASP), a well-known declarative programming paradigm, has recently been applied to Process Mining, particularly for tasks involving declarative specifications of business processes. Declare, the most popular declarative process modeling language, allows to model processes using sets of constraints expressed in Linear Temporal Logic (LTL), which valid traces must satisfy. An encoding for Declare constraints that directly models their semantics as ASP rules has been recently introduced and shown to compare well with existing implementations. We argue that this approach can be the basis for a number of exciting follow-ups such as explainable process mining via unsatisfiable core analysis, provenance for Declare, or using Generative Datalog to support the implementation of probabilistic extensions for Declare. The paper fosters the application of ASP in multiple contexts of declarative process mining.

Keywords

Answer Set Programming, Linear Temporal Logic over Finite Traces, Declarative Process Mining

1. Introduction

A *process* is a sequence of interrelated activities performed to achieve specific goals. In stark contrast to projects, processes are recurrent and have well-defined inputs and outputs. Due to the fact that processes are repeated over time, understanding, optimizing and improving processes is an appealing problem of practical interest, as each future enactment of the process benefits of these improvements. In the case of a manufacturing process, this could mean a faster execution speed, requiring less resources, or being overall cheaper. Process Mining [1, 2] emerges as an interdisciplinary research field, encompassing tools and techniques from computer science, formal methods, data science and business process management to study processes. The central objects of Process Mining techniques are the *event log* and the *process model*. Event logs consist of all the activities that have been performed in order to execute a process, and can be naturally partitioned into *traces*, that is, group together all events produced by the same execution of the process. Process models, on the other hand, are mathematical abstractions that allow to study processes and their properties. Process models can be either described by a domain expert, or learned from event logs, which is referred to as performing the *process discovery* [3] task. Another classic task is the *conformance checking* [4] task, which amounts to compute whether a trace is compliant to a given process model or not. There exist different formalisms to express process models. In particular, there's a distinction between declarative process models and procedural (or imperative) process models [5]. Procedural models aim to explicitly describe compliant traces. The most common process models in Process Mining literature are variants of the Petri Net, an automaton-like computational structure which is procedural in nature. Declarative process models, on the other hand, consist usually of high-level specifications that *shrink* the space of valid traces, by forbidding invalid behavior. It is well-known in the process mining literature that, in practice, imperative process models and declarative process

Proceedings of the 1st International Workshop on Explainable Knowledge Aware Process Intelligence, June 20–22, 2024, Roccella Jonica, Italy

*Corresponding author.

†These authors contributed equally.

mario.alviano@unical.it (M. Alviano); valeria.fionda@unical.it (V. Fionda); antonio.ielo@unical.it (A. Ielo); francesco.ricca@unical.it (F. Ricca)

0000-0002-2052-2063 (M. Alviano); 0000-0002-7018-1324 (V. Fionda); 0009-0006-9644-7975 (A. Ielo); 0000-0001-8218-3178 (F. Ricca)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

models are effective in different contexts [6]: imperative process easily model well-structured processes while declarative process models are more apt for loosely structured processes [7]. Among declarative process modeling languages, Declare [8] has been the most successful one. The Declare language consists of a set of templates based on a set of *temporal patterns* which frequently arise when writing specifications [9]. Each Declare template describes a temporal relationship between some activities occurring in the same process trace — and, implicitly, defining *what should not happen* in a given trace for it to be considered compliant to the template. Thus, a process model in Declare is a set of constraints, obtained by instantiating some templates with specific activities. The original definition of the Declare language [8] does not provide a logic-based semantic for the templates — however, a successful stream of research has formalized and analyzed Declare through the lens of Linear Temporal Logic over Finite Traces (LTL_f; [10, 11, 12]). This has enabled process mining practicioners to exploit formal techniques grounded in LTL_f to provide specifications for processes, without the hassle and intricacies [13] of writing a complete LTL_f specifications from scratch. Furthermore, it has also provided effective techniques to tackle Declare reasoning problems. Declare-based reasoning tasks are usually dealt either with ad-hoc procedures scanning each trace [14], regular expressions [7], or by reasoning on their corresponding automaton (as in the case of monitoring [11]), which is obtained from their corresponding LTL_f definition through established techniques [15]. Recently, a prototype to perform conformance checking and query checking of Declare models by using Answer Set Programming (ASP; [16]) has been presented [17] and shown to compare well with existing implementations. ASP is a well-known declarative programming paradigm widely and successfully applied both in academia and industry [18] to solve problems in combinatorial optimization, and knowledge representation and reasoning. The promising result obtained by the new prototype makes us wonder what are the possible follow-ups of the work: it may be the case that recent advancements in ASP, especially works on minimal unsatisfiable cores [19, 20] and Generative Datalog [21, 22], could further contribute to Declare-based process mining tasks. In this paper we recall the *direct* ASP encoding for Declare [17], and propose a few possible lines of research that could stem from this work.

2. ASP Encodings for Declare

This section describes a way to encode the semantics of Declare templates in ASP. In particular, we describe a *direct* encoding of Declare semantics [17], mapping Declare templates to ASP rules, without relying on translations to LTL_f [23, 24] or finite state machines [25, 26]. We assume readers are familiar with the Declare process modeling language, ASP and the input language of the CLINGO system. Interested readers can refer to [27] for a starting point on Declare literature and to [28] for an introduction to ASP modeling.

2.1. Encoding the input

We start by describing how to encode event logs and Declare models.

Encoding the event log. Let $\mathcal{L}$ be an event log of process traces over the alphabet $\mathcal{A}$, which denotes the names of the process activities. We assume each trace $\pi \in \mathcal{L}$ is assigned a unique identifier $id(\pi)$. Each trace is encoded by means of the predicate $trace/3$, where the atom $trace(i, t, a)$ models that in the trace π with $id(\pi) = i$, the t-th activity to occur is a. Additional properties of traces - such as frequency in the log, or whether the trace is a positive example or negative example, can be modeled by means of additional predicates that refer to the identifier of the trace. Recall that in a *process trace* [29] each state contains exactly one propositional symbol.

Encoding Declare models. A Declare model is a set of Declare constraints. Each Declare constraint is assigned a unique identifier, and is encoded by means of the predicates $constraint/2$ and $bind/3$. In particular, the atom $constraint(c, t)$ models that the constraint identified by c is an instantiation of the

Declare template t. The atom $bind(c, i, a)$ models that in constraint c, the argument i is bound to the activity a. As an example, consider the constraint $Response(a, b)$:

```
constraint(c, "Response").
bind(c, arg_0, "a").
bind(c, arg_1, "b").
```

Additionally, in the *query checking* task, some input constraints are only *partially specified* and contain placeholder variables to which actual activities can be assigned. To encode that a variable can span over a set of activities, we use the $domain/2$ predicate, where an atom $domain(x, y)$ models that variable x can be assigned the value y. An atom $varbind(c, i, a)$ models that in constraint c, the argument i is bound to the placeholder variable a. For example, consider the partially specified constraint $Response(a, ?x)$, where x is a placeholder variable that can be bound to activities in the domain $\{b, c\}$:

```
constraint(c, "Response").
bind(c, arg_0, "a").
varbind(c, arg_1, x).
domain(x, "b"). domain(x, "c").
```

2.2. Encoding Declare semantics

We show an example of how Declare constraints can be *directly* encoded in ASP rules, without relying on intermediate representations. We will use the constraint $Response(a, b)$ as an example. The general idea is to write rules that encode conditions that would make the constraint invalid over a given input trace (which we can formally define by inspecting the negation of the template LTL_f definition). Then, a constraint is valid if conditions for its failure are not met. Failure conditions are defined through $fail(c, i)$ atoms, that model that constraint c does not hold over trace i.

Example: encoding the Response template. Consider the $Response(a, b)$, which states that *every time a is executed, b must be executed as well.* Its LTL_f definition is $Response(a, b) = \mathbf{G}(a \rightarrow \mathbf{F}b)$, hence we want ASP rules that encode its failure conditions:

$$\begin{aligned}\neg\mathbf{G}(a \rightarrow \mathbf{F}b) &= \neg(\neg\mathbf{F}(\neg(\neg a \vee \mathbf{F}b))) \\ &= \mathbf{F}(a \wedge \neg\mathbf{F}b)\end{aligned}$$

Indeed, the failure condition for $Response(a, b)$ is that **at some point it is true a occurs in the trace but such occurrence of a is not followed by a b**. We can can express this failure condition in the following way:

```
witness(C,TID,T) :-
  constraint(C, "Response"),
  bind(C, arg_0, A),
  bind(C, arg_1, B),
  trace(TID, T, A),
  trace(TID, T', B),
  T' > T.

fail(C,TID) :-
  constraint(C, "Response"),
  bind(C, arg_0, A),
  trace(TID, T, A),
  not witness(C, TID, T).
```

2.3. Encoding Declare tasks

Let Π be a logic program which contains the failure condition definition for each Declare template.

Conformance checking. To perform conformance checking of a model $\mathcal{M}$ over an event log $\mathcal{L}$, we compute the (unique) stable model of $\Pi \cup \mathcal{L} \cup \mathcal{M}$, projecting it over the predicate $sat/2$. If $sat(c, i)$ belongs to the answer set, then constraint c holds true over trace i in the log. In order to define $sat/2$, we add the following rule:

```
sat(C,TID) :- constraint(C,_), trace(TID,_,_), not fail(C,TID).
```

Query checking. Recall that *query checking* is the problem of computing whether there exists a way to bind activities to a partially specified Declare template such that the *support* over a given log is greater than a percentage s. The support of a Declare constraint over a log is the fraction of traces that are compliant to the constraint. To perform query checking of a partially specified Declare model $\mathcal{M}$ over a log $\mathcal{L}$, we compute the answer sets of the logic program $\Pi \cup \mathcal{M} \cup \mathcal{L} \cup \Pi_{QC}$, where Π_{QC} is the following program:

```
#const max_violations=0.
{ assign(Var, Val): domain(Var, Val) } = 1.
bind(C, Arg, Val) :- varbind(C, Arg, Var), assign(Var, Val).

:- #count{TID: trace(TID,_,_), constraint(C,_), fail(C,TID)} >= max_violations.
```

The choice rule generates possible assignments of placeholder values to actual values. If we wish for support to be greater or equal to s, the percentage of violating traces should be less than $1 - s$. Since ASP can't handle non-integer numerical values, we set the integer constant $max\_violations$ to the integer $\lceil |\mathcal{L}| \cdot (1 - s) \rceil$ — that is, the maximum (integer) number of traces in the log that can be violated to yield an overall support of s. Solutions to the query checking problem, in terms of variables-activities assignments, can be retrieved by projecting answer set onto atoms matching the $assign/2$ signature, where the atom $assign(x, y)$ models that the variable with identifier x is assigned the value y among its domain.

Other options to encode Declare in ASP are *indirect* approaches, relying (also ASP-encoding wise) to the LTL_f definition of each template, or its translation into a finite state machine. As reported in [17], the direct encoding results in better performances for conformance checking and query checking compared to the indirect ASP- methods.

3. Discussion and Future Lines of Research

The declarative nature and feature-richness of ASP allows us to provide fully declarative solutions for slight variations of the previous tasks, or even new tasks, using only a few extra rules. As an example, it is easy to extend the query checking encoding with preference criteria over assignments by exploiting weak constraints, or use the conformance checking encoding to perform (bounded) satisfiability of a Declare model, by omitting input facts describing the log and using a choice rule to generate a (log containing a) single trace. However, there is another advantage: once Declare tasks are represented as logic programs, we can apply on top of them techniques that were developed to reason about logic programs under the stable model semantics. We believe this can become a good starting point to investigate several areas in Declare-based Process Mining.

Provenance & Explainability. Provenance [30] is the problem of *determining which inputs provide specific outputs*. In the case of recursive Datalog programs (such as the logic program required to perform conformance checking), the problem of *why*-provenance, that is, computing a subset of facts

that is sufficient to yield a result, is NP-complete [31], thus ASP is a suitable tool for the job. In our setting, provenance can be used to reason about *which* events in a trace cause the activation, violation or satisfaction of certain constraints. Another important issue dealing with declarative specifications, and Declare is no exception, is being able to reason about satisfiability of said specifications. Recent works on *minimal unsatisfiable cores* [19, 20] for ASP programs (e.g., *why is this logic program unsatisfiable?*) can provide techniques and methods to reason about inconsistency in Declare. SAT-based techniques based on hitting set enumeration have been used to address Declare model repair [32]. An ASP-based implementation of Declare semantics can provide a starting point to apply these techniques in the context of process mining.

Data-aware Declare. The original Declare semantics does not allow to take into account *data payloads* associated to events, which is however an essential aspect of process mining. MP-Declare [33] is an extension of Declare language that takes events' payloads into account. Some systems implement MP-Declare constraints' semantics by ad hoc procedures, however some SAT-based techniques have also been devised [34]. With respect to SAT, ASP provides a more natural way to model data payloads, since ASP atoms have a natural relational structure. Techniques shown in [35], which proposes the Declare-as-automata ASP encoding, already provides a way to take into account event payloads on Declare constraints, albeit in a limited way.

Probabilistic Declare. Another interesting aspect original Declare semantics does not cover is uncertainty. In order for a trace to be compliant with a Declare model, a trace must satisfy all the constraints in the model. Recently, a probabilistic extension of Declare has been proposed [36], where each constraint in a model has to be satisfied with a given probability. Generative Datalog [21, 22], a probabilistic extension of the Datalog language, could provide an interesting basis to apply ASP-based methods, carrying over the advantages of the declarative approach, to such probabilistic extensions of Declare.

Acknowledgements

This work was partially supported by the Italian Ministry of University and Research (MUR) under PRIN project PINPOINT - CUP H23C22000280006, PRIN project HypeKG - CUP H53D23003710006, PRIN project PRODE - CUP H53D23003420006, PRIN PNRR project DISTORT - CUP H53D23008170001, PNRR project FAIR "Future AI Research" - Spoke 9 - WP9.1 and WP9.2 - CUP H23C22000860006, PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement" - CUP H23C22000370006, PNRR project SERICS "SEcurity and RIghts in the CyberSpace" - CUP H73C22000880001; by Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); and under the NRRP MUR program funded by the NextGenerationEU. Mario Alviano and Francesco Ricca are members of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[1] W. M. P. van der Aalst, Process mining: A 360 degree overview, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 3–34. URL: https://doi.org/10.1007/978-3-031-08848-3_1. doi:10.1007/978-3-031-08848-3_1.

[2] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016. URL: https://doi.org/10.1007/978-3-662-49851-4. doi:10.1007/978-3-662-49851-4.

[3] W. M. P. van der Aalst, Foundations of process discovery, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Infor-*

mation Processing, Springer, 2022, pp. 37–75. URL: https://doi.org/10.1007/978-3-031-08848-3_2. doi:10.1007/978-3-031-08848-3_2.

[4] J. Carmona, B. F. van Dongen, M. Weidlich, Conformance checking: Foundations, milestones and challenges, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 155–190. URL: https://doi.org/10.1007/978-3-031-08848-3_5. doi:10.1007/978-3-031-08848-3_5.

[5] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H. A. Reijers, Imperative versus declarative process modeling languages: An empirical investigation, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I, volume 99 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 383–394. URL: https://doi.org/10.1007/978-3-642-28108-2_37. doi:10.1007/978-3-642-28108-2_37.

[6] 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. URL: https://doi.org/10.1007/s00450-009-0057-9. doi:10.1007/S00450-009-0057-9.

[7] C. D. Ciccio, M. Mecella, On the discovery of declarative control flows for artful processes, ACM Trans. Manag. Inf. Syst. 5 (2015) 24:1–24:37. URL: https://doi.org/10.1145/2629447. doi:10.1145/2629447.

[8] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for loosely-structured processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA, IEEE Computer Society, 2007, pp. 287–300. URL: https://doi.org/10.1109/EDOC.2007.14. doi:10.1109/EDOC.2007.14.

[9] M. B. Dwyer, G. S. Avrunin, J. C. Corbett, Patterns in property specifications for finite-state verification, in: B. W. Boehm, D. Garlan, J. Kramer (Eds.), Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999, ACM, 1999, pp. 411–420. URL: https://doi.org/10.1145/302405.302672. doi:10.1145/302405.302672.

[10] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 854–860. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

[11] G. D. Giacomo, R. D. Masellis, M. Grasso, F. M. Maggi, M. Montali, Monitoring business metaconstraints based on LTL and LDL for finite traces, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, volume 8659 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 1–17. URL: https://doi.org/10.1007/978-3-319-10172-9_1. doi:10.1007/978-3-319-10172-9_1.

[12] G. D. Giacomo, R. D. Masellis, M. Montali, Reasoning on LTL on finite traces: Insensitivity to infiniteness, in: C. E. Brodley, P. Stone (Eds.), Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada, AAAI Press, 2014, pp. 1027–1033. URL: https://doi.org/10.1609/aaai.v28i1.8872. doi:10.1609/AAAI.V28I1.8872.

[13] B. Greenman, S. Saarinen, T. Nelson, S. Krishnamurthi, Little tricky logic: Misconceptions in the understanding of LTL, Art Sci. Eng. Program. 7 (2023). URL: https://doi.org/10.22152/programming-journal.org/2023/7/7. doi:10.22152/PROGRAMMING-JOURNAL.ORG/2023/7/7.

[14] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4py: A python library for declarative process mining, in: C. Janiesch, C. D. Francescomarino, T. Grisold, A. Kumar, J. Mendling, B. T. Pentland, H. A. Reijers, M. Weske, R. Winter (Eds.), Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2022 co-located with 20th International Conference on Business Process Management (BPM 2022), Münster, Germany, September 11th to 16th, 2022, volume 3216 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 117–121. URL: https://ceur-ws.org/Vol-3216/paper_249.pdf.

[15] G. D. Giacomo, M. Favorito, Compositional approach to translate ltlf/ldlf into deterministic finite automata, in: S. Biundo, M. Do, R. Goldman, M. Katz, Q. Yang, H. H. Zhuo (Eds.), Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS

2021, Guangzhou, China (virtual), August 2-13, 2021, AAAI Press, 2021, pp. 122–130. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15954.

[16] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. URL: https://doi.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[17] F. Chiariello, V. Fionda, A. Ielo, F. Ricca, A direct ASP encoding for declare, in: M. Gebser, I. Sergey (Eds.), Practical Aspects of Declarative Languages - 26th International Symposium, PADL 2024, London, UK, January 15-16, 2024, Proceedings, volume 14512 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 116–133. URL: https://doi.org/10.1007/978-3-031-52038-9_8. doi:10.1007/978-3-031-52038-9_8.

[18] A. A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, Künstliche Intell. 32 (2018) 165–176. URL: https://doi.org/10.1007/s13218-018-0548-6. doi:10.1007/S13218-018-0548-6.

[19] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, ASP and subset minimality: Enumeration, cautious reasoning and muses, Artif. Intell. 320 (2023) 103931. URL: https://doi.org/10.1016/j.artint.2023.103931. doi:10.1016/J.ARTINT.2023.103931.

[20] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, Enumeration of minimal models and muses in WASP, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 29–42. URL: https://doi.org/10.1007/978-3-031-15707-3_3. doi:10.1007/978-3-031-15707-3_3.

[21] M. Alviano, M. Lanzinger, M. Morak, A. Pieris, Generative datalog with stable negation, in: F. Geerts, H. Q. Ngo, S. Sintos (Eds.), Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023, ACM, 2023, pp. 21–32. URL: https://doi.org/10.1145/3584372.3588656. doi:10.1145/3584372.3588656.

[22] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, Z. Vagena, Declarative probabilistic programming with datalog, ACM Trans. Database Syst. 42 (2017) 22:1–22:35. URL: https://doi.org/10.1145/3132700. doi:10.1145/3132700.

[23] A. Ielo, M. Law, V. Fionda, F. Ricca, G. D. Giacomo, A. Russo, Towards ilp-based LTL f passive learning, in: E. Bellodi, F. A. Lisi, R. Zese (Eds.), Inductive Logic Programming - 32nd International Conference, ILP 2023, Bari, Italy, November 13-15, 2023, Proceedings, volume 14363 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 30–45. URL: https://doi.org/10.1007/978-3-031-49299-0_3. doi:10.1007/978-3-031-49299-0_3.

[24] I. Kuhlmann, C. Corea, J. Grant, Non-automata based conformance checking of declarative process specifications based on ASP, in: J. D. Weerdt, L. Pufahl (Eds.), Business Process Management Workshops - BPM 2023 International Workshops, Utrecht, The Netherlands, September 11-15, 2023, Revised Selected Papers, volume 492 of *Lecture Notes in Business Information Processing*, Springer, 2023, pp. 396–408. URL: https://doi.org/10.1007/978-3-031-50974-2_30. doi:10.1007/978-3-031-50974-2_30.

[25] F. Chiariello, Automata-based temporal reasoning in answer set programming with application to process mining, in: A. Brunello, A. Gianola, F. Mogavero (Eds.), Short Paper Proceedings of the 5th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the 22nd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2023), Rome, Italy, November 7, 2023, volume 3629 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 37–42. URL: https://ceur-ws.org/Vol-3629/paper6.pdf.

[26] F. Chiariello, F. M. Maggi, F. Patrizi, From LTL on process traces to finite-state automata, in: D. Fahland, A. Jiménez-Ramírez, A. Kumar, J. Mendling, B. T. Pentland, S. Rinderle-Ma, T. Slaats, J. Versendaal, B. Weber, M. Weske, K. Winter (Eds.), Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum at BPM 2023 co-located with 21st International Conference on Business Process Management (BPM 2023), Utrecht, The Netherlands, September 11th to 15th, 2023, volume 3469 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 127–131. URL: https://ceur-ws.org/Vol-3469/paper-23.pdf.

[27] C. D. Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring,

in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 108–152. URL: https://doi.org/10.1007/978-3-031-08848-3_4. doi:10.1007/978-3-031-08848-3_4.

[28] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012. URL: https://doi.org/10.2200/S00457ED1V01Y201211AIM019. doi:10.2200/S00457ED1V01Y201211AIM019.

[29] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623. URL: https://doi.org/10.1613/jair.1.11256. doi:10.1613/JAIR.1.11256.

[30] J. Cheney, L. Chiticariu, W. C. Tan, Provenance in databases: Why, how, and where, Found. Trends Databases 1 (2009) 379–474. URL: https://doi.org/10.1561/1900000006. doi:10.1561/1900000006.

[31] M. Calautti, E. Livshits, A. Pieris, M. Schneider, The complexity of why-provenance for datalog queries, CoRR abs/2303.12773 (2023). URL: https://doi.org/10.48550/arXiv.2303.12773. doi:10.48550/ARXIV.2303.12773. arXiv:2303.12773.

[32] C. Corea, S. Nagel, J. Mendling, P. Delfmann, Interactive and minimal repair of declarative process models, in: A. Polyvyanyy, M. T. Wynn, A. V. Looy, M. Reichert (Eds.), Business Process Management Forum - BPM Forum 2021, Rome, Italy, September 06-10, 2021, Proceedings, volume 427 of *Lecture Notes in Business Information Processing*, Springer, 2021, pp. 3–19. URL: https://doi.org/10.1007/978-3-030-85440-9_1. doi:10.1007/978-3-030-85440-9_1.

[33] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Syst. Appl. 65 (2016) 194–211. URL: https://doi.org/10.1016/j.eswa.2016.08.040. doi:10.1016/J.ESWA.2016.08.040.

[34] F. M. Maggi, A. Marrella, F. Patrizi, V. Skydanienko, Data-aware declarative process mining with SAT, ACM Trans. Intell. Syst. Technol. 14 (2023) 75:1–75:26. URL: https://doi.org/10.1145/3600106. doi:10.1145/3600106.

[35] F. Chiariello, F. M. Maggi, F. Patrizi, Asp-based declarative process mining, in: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 5539–5547. URL: https://doi.org/10.1609/aaai.v36i5.20493. doi:10.1609/AAAI.V36I5.20493.

[36] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Inf. Syst. 109 (2022) 102033. URL: https://doi.org/10.1016/j.is.2022.102033. doi:10.1016/J.IS.2022.102033.