Data-Aware Declarative Process Mining with SAT

FABRIZIO MARIA MAGGI, Free University of Bozen-Bolzano, Italy
ANDREA MARRELLA, Sapienza University of Rome, Italy
FABIO PATRIZI, Sapienza University of Rome, Italy
VASYL SKYDANIENKO, University of Tartu, Estonia

Process Mining is a family of techniques for analyzing business process execution data recorded in event logs. Process models can be obtained as output of automated process discovery techniques or can be used as input of techniques for conformance checking or model enhancement. In Declarative Process Mining, process models are represented as sets of temporal constraints (instead of procedural descriptions where all control-flow details are explicitly modeled). An open research direction in Declarative Process Mining is whether multi-perspective specifications can be supported, i.e., specifications that not only describe the process behavior from the control-flow point of view, but also from other perspectives like data or time. In this paper, we address this question by considering SAT (Propositional Satisfiability Problem) as a solving technology for a number of classical problems in Declarative Process Mining, namely log generation, conformance checking and temporal query checking. To do so, we first express each problem as a suitable FO (First-Order) theory whose bounded models represent solutions to the problem, and then find a bounded model of such theory by compilation into SAT.

CCS Concepts: • **Applied Computing** → **Rule Mining**.

Additional Key Words and Phrases: Process Mining, SAT, Alloy, Multi-Perspective Models, Declarative Models.

1 INTRODUCTION

Declarative process modeling languages have been recently proposed to represent *knowledge-intensive* business processes [21], that is, processes whose control flow depends not only on the actions performed but also on the involved data, as well as the relationships among such data. The proposed declarative languages are used to express process models in the form of constraints, and aim at balancing flexibility and support. With them, indeed, the modeler can elicit the (minimal) set of *constraints* that must be satisfied in order to execute the process correctly, without explicitly stating how the participants involved in the process must actually act. In a word, the modeler specifies the *what* but not the *how*.

While a number of declarative languages are available for this purpose (LTL, LDL, together with potential first-order extensions, and possibly interpreted over finite traces [19]) the *de-facto* standard in declarative process modeling is Declare [50], which, in its original propositional variant, can be seen as a strict fragment of LTL_f, i.e., the linear-time temporal logic LTL interpreted over finite traces [19]. Declare provides a graphical language for modeling business processes in terms of *activities* (atomic units of work) and *constraints* over *sequences of activities*, called *traces*, used to model process executions. Constraints are predefined formulae called *templates*, essentially LTL_f formula templates to be instantiated with the activities of interest, ranging from standard sequential

Authors' addresses: Fabrizio Maria Maggi, Free University of Bozen-Bolzano, Bolzano, Italy, maggi@inf.unibz.it; Andrea Marrella, Sapienza University of Rome, Rome, Italy, marrella@diag.uniroma1.it; Fabio Patrizi, Sapienza University of Rome, Rome, Italy, patrizi@diag.uniroma1.it; Vasyl Skydanienko, University of Tartu, Tartu, Estonia, patrizi@diag.uniroma1.it.

2157-6904/2023/5-ART $15.00
https://doi.org/10.1145/3600106

patterns to loose relations, prohibitions and cardinality constraints. For instance: the template formula $\mathbf{F}x$ (*presence*) states that activity x must eventually occur in a process trace; template $\mathbf{G}(x \rightarrow \mathbf{XF}y)$ (*response*) expresses that whenever activity x occurs in a trace, y must eventually occur after it. A Declare specification consists of a set of constraints of this form, to be satisfied in conjunction.

The semantics of Declare has been defined using different logic-based approaches that enable a wide range of reasoning and verification capabilities. Originally, it was meant to constrain control-flow aspects only (in this sense it is said to be *single-perspective*), while disregarding other crucial aspects of business processes, such as the data (i.e., the *payload*) carried by events, possible conditions over them, as well as their interaction over time [37, 41, 56]. Previous works have tackled these limitations by providing the language with first-order (FO) features that, combined with the temporal operators, enable Declare to express properties about data. The resulting language allows for modeling properties of interest from both the control-flow and the data perspective, hence the name MP-Declare (*Multi-Perspective Declare*) [10].

Various approaches have been devised for the problem of verifying whether an *event log*, i.e., a set of traces produced each by a process execution, satisfies a set of MP-Declare constraints [10, 20, 40], whereas very few have been proposed for the wider spectrum of Data-Aware Declarative Process Mining problems that arise when data and control-flow aspects are combined. As a result, a theoretically-founded, fully integrated tool for Data-Aware Declarative Process Mining is still missing. This paper aims at filling this gap by adopting SAT, which stands for *Propositional Satisfiability Problem*, as a solving technology for a number of problems in Data-Aware Declarative Process Mining, namely the following: *(i) Event Log Generation*, i.e., the problem of generating an event log all of which traces satisfy an input specification; *(ii) Conformance Checking*, i.e., the problem of checking whether all traces in an input event log satisfy an input specification; *(iii) Query Checking*, i.e., the problem of discovering the activities in an input trace that fulfill an input query (formula with variables).

SAT, the Boolean Satisfiability problem, consists in finding a *(propositional) model* of a Boolean formula γ, i.e., an assignment of truth values to the propositional variables occurring in γ, which makes γ true. This is one of the most studied problems in Computer Science, proven to be NP-complete in the early 70's [13]. Since then, increasingly more efficient algorithms have been studied over the years, leading to the current solvers based on the *conflict-driven clause-learning* (CDCL) approach [23], which have proven extremely efficient on real-world problem instances. The classical approach to take advantage of such efficiency consists in reducing an instance of a problem P (in NP) of interest to instances of SAT, which is also the approach we adopt here (although indirectly).

While ultimately using SAT as the solution engine, we do not provide a direct compilation schema from the problem specification to the corresponding SAT instance; instead, we take advantage of the Alloy Analyzer tool [32] by providing a compilation schema from the original problems into the Alloy input language. Alloy Analyzer is a tool that takes as input a first-order logic (FOL) specification φ, together with an integer bound b, and builds, if any, a structure containing, overall, at most b objects, which satisfies φ; formally, Alloy Analyzer builds (if any) a *finite-model* ι of φ with interpretation domain of cardinality bounded by b. In order to search for the model ι, Alloy Analyzer first reduces, for the given bound, the specification φ into a SAT formula, then calls a SAT solver for its solution, and finally translates the found model (if any) back into FOL (as a set of facts). The input specification and the output model are written in Alloy, the Alloy Analyzer input language, which can be regarded as a concrete FOL syntax.

Our interest in this tool lies in the fact that each problem above can be easily specified in FOL, thus making Alloy Analyzer a readily available solver for our problems, as well as the SAT technology. The whole approach, including the proposed compilation and the actual solution steps, is illustrated in Figure 1, where the internal steps of Alloy Analyzer are shown in the bottom part. We observe that the compilation of each problem instance into Alloy is automated.

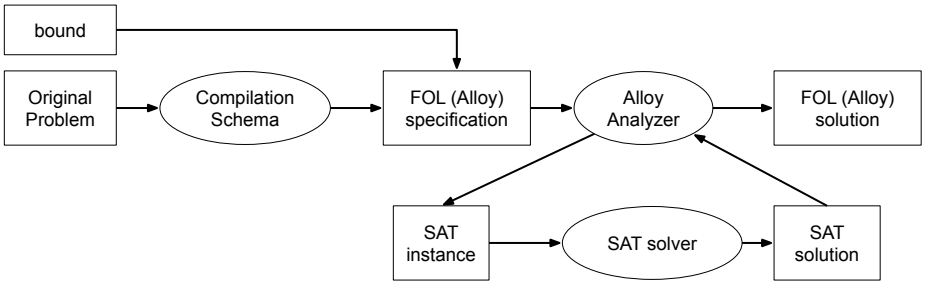

Fig. 1. Illustration of the proposed approach.

The contributions of this paper are multiple. First, we propose a compilation schema of *Event Log Generation*, *Conformance Checking*, and *Query Checking* into FOL, which makes Alloy Analyzer effective as a solver, currently missing, for Data-Aware Declarative Process Mining. We stress that, by considering the compilation step from FOL to SAT that Alloy Analyzer takes, we are in fact proposing a compilation schema into SAT. We also observe that having a FOL formalization yields the desirable side-result of opening the way to other forms of FOL-based reasoning, which can be useful, in principle, to solve also other Declarative Process Mining problems where data play a central role.

Second, the proposed approach represents a significant step forward in the solution of all three Data-Aware Declarative Process Mining problems addressed. In fact, with SAT, it has been possible for the first time to implement a log generator based on declarative process models including data-aware rules. As shown in the experimentation, the log generator is applicable also to generate reasonably large logs starting from reasonably large process models. The log generator is available in the widely used Declarative Process Mining tool RuM [3]. Similarly, the query checking problem has been addressed for the first time with a solution supporting data-aware queries, i.e., queries containing also conditions on the data perspective of a process. In our experiments, we show that SAT is applicable to perform data-aware queries on real-life logs in a reasonable amount of time. We have used SAT also for conformance checking. Although the ad hoc algorithms available in the literature for data-aware conformance checking (tailored to the Declare set of rules) are generally more efficient than the solution implemented SAT, the proposed SAT-based solution is general purpose and, therefore, can be easily used to check any FOL rule.

Third, we carried out an experimental evaluation over the selected problems, aimed at showing the feasibility of the approach and the effectiveness of SAT for Data-Aware Declarative Process Mining. The efficiency of the approach was tested by performing experiments based on synthetic datasets of different complexity to measure the execution times needed to perform the three proposed tasks. In addition, to show what type of insights can be discovered with the presented approach, and to check its scalability in real-life scenarios, we applied it to analyze a real-life event log containing events of sepsis cases from a hospital. The SAT solver used in our experiments was SAT4J. However, the SAT instance Alloy produces can be provided as input to any SAT solver, thus making virtually all SAT solvers, as well as their corresponding optimizations, directly available for Declarative Process Mining.

Finally, we have made publicly available an actual implementation of the proposed approach, in particular the compiler from the problem instances into Alloy, together with the experimental data that can be taken as a baseline for future research in the area.

The paper is organized as follows. In Section 2, we briefly introduce some basic notions about Process Mining, the Declare language and its variants, and the three problems considered here. In Section 3, we present the technical

framework, by recalling the basics of FOL, explaining how log traces can be encoded in the FOL formalism and briefly discussing MP-Declare. In Section 4, we present the formalization of Event Log Generation, Conformance Checking, and Query Checking into FOL, and thus the Alloy language. In Section 5, we explain how the problems are solved by resorting to Alloy Analyzer. In Section 6, we discuss the experiments we carried out. Finally, in Section 7, we discuss the related work, and in Section 8 we draw some conclusions.

2 PRELIMINARIES

In this section, we introduce the Declare language and provide background information about Process Mining.

2.1 The Declare Modeling Language and its Multi-Perspective extension

The comparison between the use of procedural and declarative process modeling languages to model a business process has been largely investigated in the last years [51, 53, 62]. The results of these studies have shown that procedural models are more suitable to support the execution of business processes in stable and predictable environments characterized by predefined procedures. In contrast, declarative process modeling languages like Declare work under an "open world" assumption and provide process participants with a set of rules that should not be violated. This approach is suitable to model unstable and unpredictable processes since process participants have the flexibility to follow any path that does not violate the modeled rules. Declare has been first presented in [50].

A Declare model consists of a set of constraints applied to (atomic) activities. The semantics of Declare constraints can be expressed in the linear-time temporal logic interpreted over finite traces, LTL_f [19].An example of Declare constraint is the *response* constraint RESPONSE(a, b), which states that if activity a *occurs*, b must eventually *follow*. Constraints of type *alternate* express stronger types of relations specifying that activities must alternate without repetitions in between. Even stronger ordering relations are specified by constraints of type *chain*, which state that activities must occur one immediately after the other. Other constraints refer to the cardinality of an activity, e.g., a has to occur at least once or at most once in a trace. There are also constraints that force activities to be mutually exclusive or not to occur one after the other in a trace.

Concerning the semantics, the response constraint RESPONSE(a, b) is satisfied by, e.g., traces τ_1 = c a b, τ_2 = b c b and τ_3 = a b b, whereas it is not satisfied by τ_4 = a c b a, since the second occurrence of a is never followed by one of b. An *activation* of a constraint in a trace is an event that "triggers" the constraint by imposing an obligation on the occurrence of another event (the *target*). For example, for RESPONSE(a, b), a is an activation, because its occurrence forces b (the target) to be executed eventually.

Recently, an extension of the Declare language, Multi-Perspective Declare (MP-Declare), has been presented in [10]. This extension not only captures control-flow constraints (like Declare), but takes also into consideration the data perspective. In particular, in MP-Declare, two types of data conditions can be defined: *activation condition* and *correlation condition*. In these conditions, data related to the activation of a constraint is expressed in the form `A.data`, whereas data which relates to the target is expressed in the form `T.data`.

When the activation of an MP-Declare constraint occurs, the constraint is activated only if the corresponding activation condition is satisfied. For example, constraint ABSENCE(1, `SendInvoice`) without conditions indicates that an invoice should never be sent. However, if we add an activation condition such as the following:[1]

$$\text{ABSENCE}(1, \texttt{SendInvoice})\ [\texttt{A.amount} < 100]$$

this means that an invoice cannot be sent only if the paid amount is lower than 100.

A correlation condition must be valid when the target of a constraint occurs and can involve data related to both activation and target. Therefore, this type of conditions can be used to specify correlations between two events. For example, the constraint:

[1]Activation condition and correlation condition are specified in a constraint with the format [Activation] [Correlation]

$$\text{RESPONSE}(\texttt{ReceivePayment}, \texttt{SendInvoice})$$
$$[\texttt{A.amount} > 100][\texttt{T.orderID} = \texttt{A.orderID}]$$

will only be activated if the paid amount is greater than 100. When this happens, the payment must be followed by the delivery of an invoice for the same order.

2.2 Process Mining

Process Mining [57] is a field that studies techniques for analyzing business processes based on event logs collected at process execution time. Some techniques, i.e., *Automated Process Discovery* [4], build process models that capture the behavior of the process as recorded in an event log. Other, i.e., *Conformance checking* [11], allow users to compare the real behavior of a process derived from an event log with the expected behavior represented by a process model, suitably expressed in a formal language, or to extend/enhance a process model by using the information retrieved from a log. The increasing interest from industry in this area is witnessed by the growing number of software vendors providing tools for Process Mining.[2][3][4][5][6]

As already mentioned, the main input of any Process Mining technique is an *event log*. XES (eXtensible Event Stream) [29, 60] is the standard for representing event logs in XML format. This standard represents a log as a set of traces (i.e., process executions) and traces as sequences of events. Each event in a trace represents the execution of an activity in the process. An event must always record a timestamp (when the corresponding activity was executed) and can (optionally) record additional information such as the resource executing the activity, or other data elements related to the event.

One of the main open issues in Process Mining concerns the development of multi-perspective analysis techniques that fully leverage the heterogeneous information available in an event log, as stated in [28]. This is especially true in the context of Data-Aware Declarative Process Mining, which is where this paper stands. Specifically, we put forward SAT as a solving technology for the following classical problems from Data-Aware Declarative Process Mining: *Generation of Event Logs*, *Conformance Checking*, and *Temporal Query Checking*, which we detail below.

Generation of Event Logs. One problem in Automated Process Discovery is that, despite the public availability of several real-life logs that can be used to test and evaluate process discovery techniques, these are often incomplete and/or contain noise. This aspect significantly complicates the evaluation of the techniques, as it prevents the user from selectively stressing specific features of the implemented algorithms, by controlling and tuning specific features of the input traces. Event Log Generation is the problem of generating synthetic logs whose trace fulfill predefined requirements, that the user provides as input.

Conformance Checking. Conformance checking techniques allow users to compare the behavior of a process observed in an event log with a model of the same process representing its expected behavior [2, 54, 58]. There are several contexts in which the analysis of the deviations of the execution of a business process from a prescriptive specification is critical, such as process auditing [59] or risk analysis [27].

Query Checking. Query Checking [52] aims at discovering temporal properties of a trace that are not known a priori, but respect a predefined structure. The inputs of Query Checking include a trace and a *query*, i.e., a temporal logic formula containing one or more *placeholders*. The output is a set of temporal logic formulas which derive from the input query by replacing all the placeholders with propositional formulas, which make the overall formula satisfied in the trace.

[2] http://fluxicon.com/disco/
[3] https://www.celonis.com/intelligent-business-cloud
[4] https://www.signavio.com/products/process-intelligence/
[5] https://www.my-invenio.com/
[6] https://www.minit.io/

3 THE FRAMEWORK

In this section, we introduce the technical framework used in the paper. Essentially, we provide an axiomatization of traces in first-order logic (FOL) interpreted over the naturals (and additional objects). We then show how MP-Declare constraints can be captured as FOL formulas and formalize the problems of our interest discussed above.

3.1 Log Traces as FO interpretations

A *event e* is an expression of the form $a(t, att_1 = v_1, \ldots, att_{n_A} = v_{n_A})$ where: a is an activity name (coming from a finite set), t is the event's timestamp, each att_i is an *attribute name*, and each v_i is the value assigned to att_i. An event represents the execution of activity a, at time t, with attributes att_i assigned to values v_i. Activity attributes are typically, but not necessarily, *typed*; when this is the case, the value of the typed attribute can range only over a fixed set of values, such as integers, strings, or enumerated sets. A *trace* is a finite sequence of events: $\tau = e_1 \cdots e_n$ with *non-decreasing* timestamps, i.e., denoted by t_i the timestamp value of event e_i, it holds that $t_i \leq t_{i+1}$, for $i = 1, \ldots, n-1$.

In our approach, we model traces as structures that satisfy suitable first-order (FO) theories. In order to show how this is done, we briefly recall some basics of FOL.

First-order Logic (FOL). A (FO) *signature* is a tuple $\sigma = \langle C, \mathcal{P}, \mathcal{F} \rangle$, where: C is a possibly infinite set of *constant* symbols, $\mathcal{P}$ is a finite set of *predicate* symbols, and $\mathcal{F}$ is a finite set of *function* symbols. Predicate and function symbols have finite arity. When needed, we write P/a (resp. f/a) to indicate that a predicate $P \in \mathcal{P}$ (function $f \in \mathcal{F}$) symbol has arity a.

An *interpretation* ι of a signature σ over a (possibly infinite) *universe* Δ is a pair $\iota = \langle \Delta, \cdot^\iota \rangle$, where $\cdot^\iota$ is the *interpretation function*, i.e., a function associating:

- each constant symbol $c \in C$ with an object $c^\iota \in \Delta$;
- each predicate symbol $P \in \mathcal{P}$ of arity a with a relation $P^\iota \subseteq \Delta^a$;
- each function symbol $f \in \mathcal{F}$ of arity a with a mapping $f^\iota : \Delta^a \to \Delta$.

In the rest of the paper, it will be convenient to visually distinguish constants from other symbols. To this end, we use different fonts: v stands for a constant symbol and v for variables and other placeholders (e.g., in events).

Given a FO vocabulary σ and a numerable set V of variable symbols, formulas φ of FOL over σ respect the following syntax:

$$\varphi = P(\vec{t}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi, \text{ where:}$$

- $P \in \mathcal{P}$ is a predicate symbol;
- $\vec{t}$ is a tuple $\langle t_1, \ldots, t_a \rangle$, with a the arity of P, and each t_i a *term* from σ, i.e., a variable or a constant symbol from σ, or a *function term*, i.e., an expression of the form $f(\vec{t'})$, with f a function symbol from σ and $\vec{t'}$ a tuple of terms from σ, of size equal to the arity of f.

As standard, we define the following abbreviations: $\varphi_1 \vee \varphi_2 \doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\forall x.\varphi \doteq \neg\exists x.\neg\varphi$, $\varphi_1 \to \varphi_2 \doteq \neg\varphi_1 \vee \varphi_2$, and $\varphi_1 \leftrightarrow \varphi_2 \doteq (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$. The variable x in $\exists x.\varphi$ (resp. $\forall x.\varphi$) is an existentially (resp. universally) quantified variable. Variables in a FO formula do not have to be quantified, in which case they are called *free variables*. Formulas containing only quantified variables are called *closed* formulas, or *sentences*, as opposed to *open* formulas, containing unquantified variables.

FO formulas are interpreted over FO interpretations and *variable assignments*, i.e., total mappings $v : V \mapsto \Delta$, for Δ the universe of the interpretation in question. Formula interpretations require interpreting terms first. Given an interpretation $\iota = \langle \Delta, \cdot^\iota \rangle$ of σ and a term t over σ, the interpretation of t under v and ι is the object $t^\iota_v \in \Delta$ such that:

- $t^\iota_v = v(t)$, if t is a variable symbol;
- $t^\iota_v = c^\iota$, if t is the constant symbol c;

- $t^{\iota}_{v} = f^{\iota}(t_1{}^{\iota}_{v}, \ldots, t_a{}^{\iota}_{v})$, for a the arity of f, if t is the function term $f(t_1, \ldots, t_a)$.

In this paper, we assume that every interpretation ι of σ is such that: *(i)* Δ is finite; *(ii)* $C = \Delta$, i.e., every object in Δ is *named* by a constant, and the objects themselves are used as such names; *(iii)* for every $c \in C$, $\iota(c) = c$, i.e., every constant is interpreted as itself, that is we use the object itself as its name.

Notice, in particular, that the first assumptions implies that we deal only with *finite interpretations*.

A FO interpretation ι is said to *satisfy* a FO formula φ under assignment v, written $\iota, v \models \varphi$, iff one of the following holds:

- $\varphi = P(t_1, \ldots, t_a)$ and $\langle t_1{}^{\iota}_{v}, \ldots, t_a{}^{\iota}_{v} \rangle \in P^{\iota}$;
- $\varphi = \neg\phi$ and $\iota, v \not\models \phi$;
- $\varphi = \phi_1 \wedge \phi_2$ and $\iota, v \models \phi_i$, for $i = 1, 2$;
- $\varphi = \exists x.\phi$ and there exists $o \in \Delta$ such that $\iota, v' \models \phi$, with $v'(x) = o$ and $v'(v) = v(v)$, for $v \neq x$.

An interpretation ι *satisfies* a formula φ, written $\iota \models \varphi$, iff $\iota, v \models \varphi$ for all assignments v. A *FO theory* Γ is a possibly infinite set of FO sentences. Here, we focus on finite theories. An interpretation is said to satisfy a theory Γ, if $\iota \models \varphi$ for all $\varphi \in \Gamma$. When this is the case, ι is said to be a *model* of Γ.

FO Trace Theories. We model traces as models of a particular theory. A signature $\sigma = \langle C, \mathcal{P}, \mathcal{F} \rangle$ is said to be a *trace signature* if (we use, as standard, the infix notation for relational operators):

- C is a finite set containing integer values, including 0, plus a *finite number* of additional object constants (possibly integer);
- $\mathcal{P} = \{N/1, Seq/3, AttV/3, <, \leq, =, \geq, >\}$;
- $\mathcal{F} = \{+, -\}$.

We consider only those interpretations ι which interpret N as a *finite interval* of $\mathbb{N}_0$ including 0, i.e., $N^{\iota} = [0, n-1]$, for some n, and where all relational operators and functions on integers are interpreted as the restriction on N^{ι} of the standard corresponding operators on $\mathbb{N}_0$ (this cannot be expressed in FOL as, in particular, there is no axiomatization for integers). Since, as previously noted, we deal with finite interpretations, also models are finite.

Given a trace signature, we next define a *trace theory* Γ. The intuition is that the models of Γ are such that: *Seq* is a predicate used to model a finite sequence indexed from 0 to some n, of activities with non-decreasing (integer) timestamps –we call the indices of the sequence *events* (as they uniquely identify an activity together with its attribute values and timestamp); *AttV* is a (possibly partial) functional relation mapping events and attribute names into objects.

Formally, Γ is the theory including the following sentences:

- *Seq* is a non-empty indexed sequence of activities and non-decreasing timestamps:
  - indexes and timestamps are integers: $\forall i, a, t.Seq(i, a, t) \rightarrow N(i) \wedge N(t)$
  - *Seq* is non-empty and starts with index 0: $\exists a, t.Seq(0, a, t)$
  - *Seq* is indexed from 0 to some final index i:
    $\exists i, a, t.Seq(i, a, t) \wedge$
    $(\neg\exists i', a', t'.i' > i \wedge Seq(i', a', t')) \wedge$ (i is the last index of *Seq*)
    $(\forall i'.0 < i' < i \rightarrow \exists a, t.Seq(i', a, t))$ (*Seq* is indexed from 0 to i)
  - timestamps are non-decreasing:
    $\forall i, a, t, a', t'.(Seq(i, a, t) \wedge Seq(i+1, a', t')) \rightarrow t' \geq t$
- *AttV* is a functional relation, mapping events and attribute names (which cannot be integers) into objects:
  - $\forall i, n, v.AttV(i, n, v) \rightarrow$
    $\neg N(n) \wedge$ (attribute names cannot be integers)
    $\neg\exists v'.v \neq v' \wedge AttV(i, n, v')$ (*AttV* is functional, i.e., given an event i and an attribute name n assigns at most one value v to n).

It is easy to see that every model ι of the theory above corresponds to a trace $\tau_\iota = e_1 \cdots e_n$. In particular, *Seq* represents the sequence of events, each represented by its position in the sequence and its associated activity and timestamp, and *AttV* is the assignment of values to the relevant attributes at each event. In detail, τ_ι can be obtained by assigning $e_i = a(t, att_1 = v_1, \ldots, att_{n_i} = v_{n_i})$, for a, t, and all att_j and v_j such that: $\langle i, a, t\rangle \in Seq^\iota$, and $\langle i, att_j, v_j\rangle \in AttV^\iota$. Similarly, one can see that the vice-versa also holds, i.e., from every trace τ, a model ι_τ of Γ can be obtained.

Example 3.1. Consider the trace $\tau_1 = \mathsf{a}_1(\mathsf{t} = 2, \mathsf{v} = \mathsf{a})\ \mathsf{a}_2(\mathsf{t} = 5, \mathsf{w} = \mathsf{b}, \mathsf{y} = \mathsf{d})\ \mathsf{a}_3(\mathsf{t} = 9, \mathsf{s} = \mathsf{c})\ \mathsf{a}_3(\mathsf{t} = 20, \mathsf{s} = \mathsf{d})$, which represents the sequence of events including activity a_1 with timestamp 2 and attribute v assigned to value a, then activity a_2 with timestamp 5 and attributes w and y respectively assigned to value b and d, and so on. Trace τ_1 is captured by the following interpretation ι_{τ_1} of the trace signature σ (for brevity, we omit the interpretation of functions + and − and relational operators but these can be easily obtained as their restriction to $N^{\iota_{\tau_1}}$):

- $\Delta = C = \{0, 1, 2, 3, 5, 9, 20\} \cup \{\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \mathsf{v}, \mathsf{w}, \mathsf{y}, \mathsf{s}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$
- $N^{\iota_{\tau_1}} = \{0, 1, 2, 3, 5, 9, 20\}$
- $Seq^{\iota_{\tau_1}} = \{\langle 0, \mathsf{a}_1, 2\rangle, \langle 1, \mathsf{a}_2, 5\rangle, \langle 2, \mathsf{a}_3, 9\rangle, \langle 3, \mathsf{a}_3, 20\rangle\}$
- $AttV^{\iota_{\tau_1}} = \{\langle 0, \mathsf{v}, \mathsf{a}\rangle, \langle 1, \mathsf{w}, \mathsf{b}\rangle, \langle 1, \mathsf{y}, \mathsf{d}\rangle, \langle 2, \mathsf{s}, \mathsf{c}\rangle, \langle 3, \mathsf{s}, \mathsf{d}\rangle\}$

Based on the correspondence above, in this paper we refer to a *trace* as a model of the trace theory Γ. As already mentioned, by the definition of Γ, traces can only be finite.

3.2 MP-Declare

We are interested in capturing formal properties of traces to express and solve typical problems from Declarative Process Mining. In general, our approach can deal with any property expressed as a FO formula over the (possibly extended) signature σ. However, in Process Mining, there are a number of patterns that typically occur. In fact, these are so common that the Process Mining community has devised a specific language, MP-Declare [10], for this purpose, which includes a number of parametric formulas called *templates*. We can formally prove that MP-Declare is captured by FO in our framework (this is a rather simple exercise), so by being able to solve problems stated in FO, we can also solve problems for MP-Declare. In this section, we show some typical templates, together with their translation in FOL. Their formal semantics, fully detailed in [10], can be obtained directly from the FOL translation.

For a FOL formula φ, we write $free(\varphi)$ to denote the set of φ's free variables. By a slight abuse of notation, we consider a vector $\vec{x}$ also as a set, namely the set including all the components of $\vec{x}$. We report below some typical MP-Declare constraints, together with an informal description.

Existence. The *Existence* constraint states the occurrence, in a trace, of some event containing activity A and satisfying a property φ_a (activation condition), expressed in FOL. Importantly, φ_a refers to a single event only. Constraint of this class have the following form:

$$Existence(A, \varphi_a) = \exists i, t, \vec{x}.Seq(i, A, t) \wedge \varphi_a,$$

where: $free(\varphi_a) \subseteq \{i, t\} \cup \vec{x}$ and φ_a does not mention *Seq* but can mention *AttV* only in atoms of the form $AttV(i, \cdot, \cdot)$.

Example 3.2. The following *Existence* constraint requires the occurrence of at least one event with activity `RefundOrder` and attribute `amount` having an integer value v equal to 10:

$$Existence(\mathsf{a}, AttV(i, \mathsf{amount}, 10)).$$

Absence. The *Absence* constraint is the opposite of *Existence*, and is thus simply defined as its negation:

$$Existence(A, \varphi_a) = \neg Absence(A, \varphi_a).$$

Choice. This constraint generalizes Existence by stating the occurrence of an event containing at least one of two activities A or B, and satisfying the activation condition φ_a:

$$Choice(A, B, \varphi_a) = Existence(A, \varphi_a) \vee Existence(B, \varphi_a).$$

Example 3.3. The following *Choice* constraint models the occurrence of a payment activity carried out by debit or credit card, for an amount equal to 100:

$$Choice(\texttt{PayDebitCard}, \texttt{PayCreditCard}, AttV(i, \texttt{amount}, 100)).$$

ExclusiveChoice. This is a specialization of Choice, which requires that exactly one between activities A and B occurs in some event that satisfies φ_a:

$$ExclusiveChoice(A, B, \phi) = (Existence(A, \varphi_a) \wedge Absence(B, \varphi_a)) \vee (Absence(A, \varphi_a) \wedge Existence(B, \varphi_a)).$$

Responded Existence. This constraint requires that if some event satisfying activation condition φ_a occurs with activity A, then some event containing activity B and satisfying correlation condition φ_c occurs. Notice that Responded Existence does not impose any ordering constraint between the event containing A and that containing B:

$$RespExistence(A, \varphi_a, B, \varphi_c) = \forall i, t.(\forall \vec{x}.((Seq(i, A, t) \wedge \varphi_a) \rightarrow (\exists i', t'.(\exists \vec{x}'.Seq(i', B, t') \wedge \varphi_c)))),$$

where: $free(\varphi_a) \subseteq \{i, t\} \cup \vec{x}$, $free(\varphi_c) \subseteq \{i, t, i', t'\} \cup \vec{x} \cup \vec{x}'$, and φ_a and φ_c do not mention Seq but can mention $AttV$ only in atoms of the forms, respectively, $AttV(i, \cdot, \cdot)$ and $AttV(i, \cdot, \cdot)$ or $AttV(i', \cdot, \cdot)$.

Example 3.4. The following constraint requires that if a child x has booked a room in an hotel then also an adult x' responsible for x must have booked, no matter in which order:

$$RespExistence(\texttt{Reserve}, \varphi_a, \texttt{Reserve}, \varphi_c), \text{ with:}$$

$$\varphi_a = AttV(i, \texttt{age}, \texttt{child}) \wedge AttV(i, \texttt{name}, x)$$

$$\varphi_c = AttV(i', \texttt{age}, \texttt{adult}) \wedge AttV(i', \texttt{name}, x') \wedge AttV(i', \texttt{resp\_for}, x).$$

Any trace stemming from the hotel reservation process that contains, e.g., the following two events (t and t' stand for two arbitrary timestamps):

- $Reserve(t, \texttt{name} = \texttt{Alice}, \texttt{age} = \texttt{child}, \texttt{resp\_for} = \texttt{none})$
- $Reserve(t', \texttt{name} = \texttt{John}, \texttt{age} = \texttt{adult}, \texttt{resp\_for} = \texttt{Alice})$

satisfies the constraint, independently of the ordering of t and t' (or, equivalently of i and i').

Response. The *Response* template specializes Responded Existence by requiring that the event containing B occurs *after* that containing A:

$$Response(A, \varphi_a, B, \varphi_c) = \forall i, t.(\forall \vec{x}.((Seq(i, A, t) \wedge \varphi_a) \rightarrow (\exists i', t'.i' > i \wedge (\exists \vec{x}'.Seq(i', B, t') \wedge \varphi_c)))),$$

where φ_a and φ_c are as above.

Example 3.5. Constraint $Response(A, \varphi_a, B, \varphi_c)$, with φ_a and φ_c as in Example 3.4 requires that the registration of a child be followed by that of an adult responsible for him/her, but not the vice-versa. Thus, a trace containing the events of Example 3.4 with $t' < t$ does not satisfy the constraint, whereas one where $t' > t$ does.

While the above list is non-exhaustive, we observe that all the constraints share a similar structure and none of them contains more than 2 free variables, i.e., template parameters. As a result, modulo the specific formulas φ_a and φ_c (the activation and correlation conditions of each template), the examples above can be easily adapted to the remaining Declare templates.

4 PROBLEMS

In previous sections we have shown how traces and MP-Declare constraints can be formalized in FOL. Now, we combine these two to provide a FOL formalization of the problems of our interest, which can then be solved by resorting to techniques for (finite) FO reasoning, through compilation into SAT . We stress that our framework is general enough to deal with *all possible* FOL constraints, not only those obtained as translation of MP-Declare templates, which are sufficient to capture typical problems from Declarative Process Mining. Nonetheless, for ease of exposition, in the rest of the section we present examples of MP-Declare constraints only. Analogous considerations hold for the experiments, which, while carried out on MP-Declare templates only, can be performed, in general, on any set of FOL constraints.

4.1 Generation of Event Logs

The *Event Log Generation* problem consists in finding a set , of desired cardinality, of traces that satisfy a sentence φ over σ. Phrased in our framework, the problem amounts to finding a set of models of formula $\varphi \wedge \Phi_\Gamma$, where $\Phi_\Gamma = \bigwedge_{\phi \in \Gamma} \phi$ is the conjunction of all the sentences occurring in (trace theory) Γ.

Since φ is an arbitrary FOL formula and the problem of checking whether one such formula admits any (even finite) model is undecidable [24], the desired set of traces is in general not computable. However, if one fixes the maximum length of the traces and the interpretation domain of ι, then it is immediate to see that the set is effectively computable, as the logic used essentially reduces to propositional logic. This is the typical scenario in Process Mining.

Example 4.1. Consider a set containing the following two MP-Declare constraints:

- $Existence(\mathsf{a}, t > 3 \wedge \forall n, v. AttV(i, n, v) \rightarrow v \neq \mathsf{d})$ (there exists an event with activity a occurring after time(stamp) 3, with all attributes assigned to a value other than d);
- $Response(\mathsf{a}, \varphi_a, \mathsf{b}, \varphi_c)$, with:
  - activation condition $\varphi_a = t \geq 1 \wedge AttV(i, \mathtt{att}_1, v_1)$,
  - correlation condition $\varphi_c = AttV(i', \mathtt{att}_2, v_2) \wedge v_1 \neq v_2$

  (every event with activity a that occurs at or after time 1 and with attribute $\mathtt{att}_1$ assigned to some value v_1 is followed by an event with activity b and attribute $\mathtt{att}_2$ assigned to some value v_2 different from v_1).

Assume that one needs to generate a set of traces of length at most, say, 10, satisfying such constraints. Moreover, assume that:

- the set of available activities is $A = \{\mathsf{a}_1, \ldots, \mathsf{a}_6\}$;
- the set of attribute names of interest is $AN = \{\mathsf{v}, \mathsf{w}, \mathsf{y}, \mathsf{s}\}$;
- the set of possible attribute values is: $AV = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$
- we are interested in the time interval $TI = [0, 100]$, i.e., the timestamps can take values only in this range.

Notice that, for ease of presentation, we are not expressing the fact that certain attribute names can be associated with certain activities only or that some values can be used only as attribute names or values. This class of constraints, however, can be easily expressed (and consequently enforced) by introducing additional predicates and constraints (in the form of FOL formulas) on top of Γ.

Considering the correspondence between models of Γ and traces, the log generation problem consists in finding a number of distinct models with interpretation domain $\Delta = TI \cup A \cup AN \cup AV$. Obviously, Δ being finite, all such models can be effectively computed (they are finite and finitely many).

For instance, the following traces (i.e., models of Γ) satisfy the constraints of Ex. 4.1:

- trace of Example 3.1, i.e., $\tau_1 = \mathsf{a}_1(\mathsf{t} = 2, \mathsf{v} = \mathsf{a})\ \mathsf{a}_2(\mathsf{t} = 5, \mathsf{w} = \mathsf{b}, \mathsf{y} = \mathsf{d})\ \mathsf{a}_3(\mathsf{t} = 9, \mathsf{s} = \mathsf{c})\ \mathsf{a}_3(\mathsf{t} = 20, \mathsf{s} = \mathsf{d})$;
- trace $\tau_2 = \mathsf{a}_2(\mathsf{t} = 5, \mathsf{v} = \mathsf{c}, \mathsf{w} = \mathsf{v})$ (notice that v is used as both an attribute name and an attribute value).

4.2 Conformance Checking

Conformance checking is the problem of checking whether all the traces τ in an event log L, satisfy a set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ of MP-Declare constraints (FOL sentences). It can be easily seen that, all traces being finite, the problem is decidable. Indeed, the problem essentially reduces to standard (boolean) query evaluation in databases [1], by seeing each trace in L, which is a FO interpretation ι, as a database over the schema σ, and each sentence $\varphi \in \Phi$ as a Boolean query over it.

Example 4.2. Consider again the set of constraints of Ex. 4.1. The first constraint is satisfied by a trace if it contains an event with timestamp t greater than 3, with activity a_2, and such that the value d is not assigned to any of its attributes. The second constraint requires that whenever an event e occurs with timestamp greater or equal than 1, with activity a_1, and with value a assigned to attribute v, then an event e' occurs strictly later with activity a_3 and such that attribute v at e and attribute s at e' are assigned distinct values.

Consider again trace $\tau_1 = \mathsf{a}_1(\mathsf{t} = 2, \mathsf{v} = \mathsf{a})\ \mathsf{a}_2(\mathsf{t} = 5, \mathsf{w} = \mathsf{b}, \mathsf{y} = \mathsf{d})\ \mathsf{a}_3(\mathsf{t} = 9, \mathsf{s} = \mathsf{c})\ \mathsf{a}_3(\mathsf{t} = 20, \mathsf{s} = \mathsf{d})$ of Example 3.1. It can be easily seen that both constraints are satisfied by the corresponding interpretation ι_{τ_1} described in Example 3.1.

4.3 Query Checking

Query checking can be described as the problem of *query answering* [1] over traces. To formally define this, we preliminarily need the notions of *query* and respective *answer*. A *query* is a FO formula. If the formula is a sentence, it is called a *Boolean* query. Intuitively, given an interpretation $\iota = \langle \Delta, \cdot^\iota \rangle$ and a query φ over the same signature σ, the *answer* to φ over interpretation ι, denoted φ^ι, is the set of assignments to the free variables (the placeholders) of φ that make ι satisfy φ. Formally, φ^ι is defined by induction as follows.

Let $v_1, \ldots, v_n$ be the free variables of a query φ, sorted according to some (arbitrary) order. Then, the *answer to φ (over ι)* is the relation $\varphi^\iota \subseteq \Delta^n$ s.t. $\langle a_1, \ldots, a_n \rangle \in \varphi^\iota$ iff for any assignment ν s.t. $\nu(v_i) = a_i$, it is the case that $\iota, \nu \models \varphi$.

The *Query Checking* problem consists in determining φ^ι. Notice that if the interpretation domain Δ is infinite, then φ^ι cannot be computed in general, while it is solvable if Δ is finite. Again, the latter is the setting of interest in Process Mining.

Example 4.3. Consider once again trace τ_1 from Example 3.1: $\tau_1 = \mathsf{a}_1(\mathsf{t} = 2, \mathsf{v} = \mathsf{a})\ \mathsf{a}_2(\mathsf{t} = 5, \mathsf{w} = \mathsf{b}, \mathsf{y} = \mathsf{d})\ \mathsf{a}_3(\mathsf{t} = 9, \mathsf{s} = \mathsf{c})\ \mathsf{a}_3(\mathsf{t} = 20, \mathsf{s} = \mathsf{d})$. Assume we are interested in all the pairs of events involved in the satisfaction of a response constraint. These are the activities $?x$ and $?y$ (leading ? denotes query variables) such that the following formula holds (with φ_a and φ_c generic activation and correlation conditions):

$$\varphi = Response(?x, \varphi_a, ?y, \varphi_c) = \forall i, t.(\forall ?x.((Seq(i, ?x, t) \wedge \varphi_a) \rightarrow (\exists i', t'.i' \geq i \wedge (\exists ?y.Seq(i', ?y, t') \wedge \varphi_c))))$$

Notice that the formula φ defined above has two free variables, $?x$ and $?y$. Thus, any answer φ^ι is a relation $\varphi^\iota \subseteq \Delta^2$. Specifically, the relation contains all those pairs of activities a_1 and a_2 that make φ true, once used to replace $?x$ and $?y$. For instance, if φ_a and φ_c are both $true$, then $\varphi^{\iota_{\tau_1}} = \{\langle \mathsf{a}_1, \mathsf{a}_2 \rangle, \langle \mathsf{a}_1, \mathsf{a}_3 \rangle, \langle \mathsf{a}_2, \mathsf{a}_3 \rangle, \langle \mathsf{a}_3, \mathsf{a}_3 \rangle\}$

If φ_a and φ_c are as in Example 4.1, we have: $\varphi^{\iota_{\tau_1}} = \{\langle \mathsf{a}_1, \mathsf{a}_3 \rangle, \langle \mathsf{a}_2, \mathsf{a}_1 \rangle, \langle \mathsf{a}_2, \mathsf{a}_2 \rangle, \langle \mathsf{a}_2, \mathsf{a}_3 \rangle, \langle \mathsf{a}_3, \mathsf{a}_1 \rangle, \langle \mathsf{a}_3, \mathsf{a}_2 \rangle, \langle \mathsf{a}_3, \mathsf{a}_3 \rangle\}$. The counter-intuitive answer to this query, i.e., the fact that some pairs contain activities in reversed order wrt that in which they appear in the trace, is due to the fact that whenever φ_a does not hold, as a consequence of the implication, φ trivially evaluates to true, no matter what activity is assigned to $?y$. It is demanded to the designer of the query to take care of this aspect.

5 ALLOY FOR DECLARATIVE PROCESS MINING

All problems presented in the previous section can be cast and solved into as many reasoning tasks in FOL (these can be ultimately regarded as problems typical of databases). Solution is possible by finiteness of the interpretation domain. As described above, solving these problems requires in practice either constructing a set of models or building a relation that satisfies the input constraints, which can be done by reduction to SAT. To this end, we resort to the tool Alloy [32], which is specifically designed for this purpose.

Alloy has been described in some detail in Section 1 (and Figure 1). For convenience, we remind here that it is a tool for the construction of FOL models over finite interpretation domains, which takes in input a FO theory and an interpretation domain (in fact, a bound on the size of the interpretation domain) and returns, if any, a model of the input theory. To do so, the tool compiles the problem into SAT, using propositions to represent FO atoms built from predicate symbols and the finitely many objects in the interpretation domain. We are not interested here in the compilation step, but in: *(i)* how we can cast our problems into that of finding a model for a FO theory; and *(ii)* how

Table 1. Time (in seconds) required for generating logs of different sizes containing traces of different lengths from Declare models of increasing sizes (without data conditions).

Trace length →	5	10	15	20	25	30	40	50
Log Size ↓				***5 constraints***				
100	0.21	0.31	0.51	0.66	1.11	1.82	3.88	7.47
250	0.52	0.89	1.02	1.32	1.8	2.61	4.93	8.82
500	0.89	1.36	2.11	2.4	3.54	4.3	7.34	11.31
1,000	1.77	2.61	3.79	4.54	5.7	6.95	11.15	17.46
2,500	4.43	7.09	8.83	11.15	14.28	17.19	24.2	31.18
5,000	9.98	13.1	17.46	22.1	27.95	32.6	45.13	58.96
10,000	19.61	26.35	34.69	46.36	53.87	63.98	83.99	112.69
Log Size ↓				***10 constraints***				
100	0.21	0.35	0.51	0.97	1.46	2.34	6.01	13.62
250	0.43	0.71	1.03	1.54	2.26	3.41	8.21	15.25
500	0.88	1.38	1.97	2.59	3.6	5.09	9.48	17.62
1,000	1.85	3.19	3.8	4.94	6.31	7.94	13.5	22.94
2,500	4.92	6.81	9.74	11.34	13.9	17.18	25.95	38.84
5,000	9.23	13.03	17.18	22.15	28.68	32.1	46.75	63.85
10,000	17.55	26.63	35.47	44.37	55.22	63.3	87.52	119.22
Log Size ↓				***15 constraints***				
100	0.2	0.47	0.59	1.02	1.83	3.31	8.85	19.74
250	0.5	0.75	1.1	1.68	2.58	4.15	10.09	20.68
500	0.93	1.39	1.92	2.79	3.98	6.54	12.15	23.46
1,000	1.81	2.75	3.75	5.01	6.62	9.46	16.22	28.48
2,500	4.48	6.74	9.74	11.51	14.74	19.02	29.03	43.77
5,000	9.29	13.92	17.27	22.53	29.47	33.63	48.68	69.4
10,000	15.63	27.05	35.21	45.05	56.48	66.56	95.61	121.65
Log Size ↓				***20 constraints***				
100	0.21	0.35	0.61	1.23	2.27	4.88	11.31	26.18
250	0.48	0.77	1.14	1.82	3.06	5.3	12.4	28.08
500	0.9	1.4	2.1	2.93	4.41	6.62	14.25	30.91
1,000	1.71	2.81	3.84	5.18	7.05	10.02	19.22	35.27
2,500	4.55	6.8	9.64	12.24	15.05	19.13	31.39	50.91
5,000	5.83	13.24	17.94	22.65	30.91	35.02	51.39	76.8
10,000	6.28	26.73	37.3	44.96	56.12	70.95	94.12	129.6

Table 2. Time (in seconds) required for generating logs of different sizes containing traces of different lengths from Declare models of increasing sizes (with activation conditions).

Trace length →	5	10	15	20	25	30	40	50
Log Size ↓				***5 constraints***				
100	0.36	0.63	0.84	1.18	1.72	2.32	5	8.71
250	0.88	1.42	2.08	3.12	3.54	4.45	7.59	11.97
500	1.76	2.56	3.91	4.84	6.17	7.93	12.46	17.73
1,000	3.12	5.08	7.43	9.66	12.14	15.02	21.2	28.96
2,500	8.71	12.88	18.86	24.05	29.28	34.94	49.52	65.41
5,000	15.44	25.65	36.43	48.75	60.29	69.55	95.09	121.26
10,000	31.15	52.96	75.54	97.55	119.47	145.05	197.75	246.47
Log Size ↓				***10 constraints***				
100	0.46	0.74	1.07	1.59	2.13	3.16	7.2	15.16
250	0.8	1.43	2.02	3.04	3.77	5.13	9.94	19.39
500	1.54	2.58	3.77	5.08	6.76	8.47	14.46	25.7
1,000	3.19	5.31	7.53	9.66	12.48	16.09	23.84	37.21
2,500	7.71	13.13	17.84	24.02	29.74	36.32	52.3	73.05
5,000	15.79	25.75	36.45	46.39	59.35	71.16	96.93	129.3
10,000	31.46	51.7	73.12	93.8	119.4	139.41	193.07	246.37
Log Size ↓				***15 constraints***				
100	0.35	0.6	1.32	1.54	2.63	4.15	9.35	21.02
250	0.81	1.41	2.04	2.93	4.25	6.73	12.33	24.65
500	1.61	2.66	3.83	5.18	7	10.47	16.81	30.47
1,000	3.14	5.54	7.98	9.89	12.83	16.2	26.5	41.96
2,500	7.77	13.73	18.14	23.53	30.44	38.16	54.03	77.14
5,000	16.09	25.37	36.14	47.86	60.98	72.27	100.7	135.21
10,000	33.11	52.62	72.07	95.28	119.78	139.93	198.73	254.18
Log Size ↓				***20 constraints***				
100	0.37	0.62	1.18	1.74	2.95	4.86	12.37	28.61
250	0.82	1.42	2.11	3.16	4.82	7.1	15.19	32.08
500	1.58	2.67	3.96	5.77	7.63	10.42	20.44	37.23
1,000	3.18	5.24	7.61	10.57	13.32	17.13	29.06	49.98
2,500	7.86	13.31	18.45	24.19	31.01	37.68	57.05	86.88
5,000	16.11	26.19	37.08	47.08	60.36	72.82	105.14	142.35
10,000	31.56	52.87	76.02	94.98	119.41	146.47	196.85	266.97

effective the (SAT-based) solution approach of Alloy is in solving the problems of our interest. In this section, we discuss the former point while, in the section about experiments, we deal with the latter.

5.1 Generation of Event Logs

Generation of event logs is possibly the simplest problem Alloy can solve. We simply take a trace theory Γ and the conjunction of the constraints to satisfy, say φ, and ask Alloy to search for the models of $\Gamma \cup \{\varphi\}$, for a given (finite) Δ. Observe that, if needed, we can add, on top of the input theory, additional constraints, as long as these are consistent with $\Gamma \cup \{\varphi\}$. For instance, we may require that the set of attribute names be restricted to a certain desired set:

$$\forall i, a, v. AttV(i, a, v) \rightarrow a = \mathsf{v} \vee a = \mathsf{w} \vee a = \mathsf{y} \vee a = \mathsf{s},$$

analogously for attribute values:

$$\forall i, a, v. AttV(i, a, v) \rightarrow v = \mathsf{a} \vee v = \mathsf{b} \vee v = \mathsf{c} \vee v = \mathsf{d},$$

Table 3. Time (in seconds) required for generating logs of different sizes containing traces of different lengths from Declare models of increasing sizes (with activation and correlation conditions).

Trace length →	5	10	15	20	25	30	40	50
Log Size ↓				***5 constraints***				
100	0.35	0.63	0.92	1.31	1.93	2.67	5.54	9.71
250	0.87	1.47	2.04	2.82	3.79	4.99	9.41	13.98
500	1.75	2.79	4.13	5.65	7.67	9.2	13.68	21.42
1,000	3.27	5.36	7.83	10.15	13.49	16.94	24.33	35.55
2,500	7.86	13.85	19.04	26.1	32.38	39.54	55.88	76.03
5,000	16.15	27.79	38.31	51.29	64.12	78.4	111.16	146.22
10,000	33.08	56.65	77.12	100.42	127.92	160.07	223.49	290.7
Log Size ↓				***10 constraints***				
100	0.35	0.62	1.06	1.67	2.35	3.69	8.3	17.2
250	0.82	1.49	2.17	3.02	4.38	5.84	12.18	21.63
500	1.63	2.83	4.05	5.63	7.52	9.79	17.45	28.69
1,000	3.26	5.57	7.8	11.32	13.92	17.74	28.18	43.74
2,500	7.8	13.68	19.56	25.42	33.62	42.1	60.08	86.17
5,000	15.43	26.7	40.15	51.71	65.48	79.69	114.23	157.13
10,000	33.87	52.8	77.44	103.33	129.5	161.36	231.33	304.12
Log Size ↓				***15 constraints***				
100	0.37	0.77	1.21	1.81	2.99	4.87	11.27	24.77
250	0.83	1.46	2.22	3.38	4.87	7.28	14.7	29.65
500	1.62	3.01	4.24	5.8	7.98	10.99	20.59	35.45
1,000	3.21	5.52	7.93	11.56	14.41	18.58	31.16	50.4
2,500	8.2	13.98	19.57	26.03	34.15	43.6	63.62	93.14
5,000	15.84	27.5	38.38	52.57	67	81.95	118.94	165.5
10,000	32.87	54.58	77.29	104.44	130.79	164.6	231.09	313.23
Log Size ↓				***20 constraints***				
100	0.38	0.81	1.16	2.07	3.52	5.83	14.58	32.69
250	0.85	1.51	2.35	3.57	5.55	8.69	18.14	36.23
500	1.66	2.91	4.26	6.03	8.73	12.97	23.14	44.12
1,000	3.43	5.62	8.07	11.23	15.12	20.38	34.26	59.13
2,500	8	13.55	20.23	26.73	35.95	43.55	69.83	103.36
5,000	15.86	27.69	39.27	52.5	67.59	84.35	124.89	173.83
10,000	32.77	54.71	81.2	105.34	133.49	167.7	240.21	328.5

or we may even require that certain attributes take only specific values:

$$\forall i, v. AttV(i, \mathsf{v}, v) \rightarrow v = \mathsf{a} \vee v = \mathsf{d}.$$

To obtain many traces, we ask Alloy once it has returned a model, to build one more, until enough it has produced as many as needed or no more are available. We can incrementally increase the size of the interpretation domain to obtain additional traces.

5.2 Conformance Checking

To solve conformance checking in Alloy, we proceed as follows. For every trace τ in the input set L, we write a theory Γ_τ which admits, for a suitable fixed interpretation domain Δ, a single model ι_τ of Γ, capturing τ. Then, we add the following sentence to Γ_τ, with R a 0-ary relation (i.e., a proposition):

$$\phi \doteq R \leftrightarrow \bigwedge_{\varphi \in \Phi} \varphi,$$

Table 4. Time (in seconds) required for generating logs of different sizes containing traces of different lengths from Declare models of increasing alphabet sizes (without data conditions).

Trace length →	20	25	30	40	50
Log Size ↓			***5 activities***		
100	1.37	1.84	2.82	6.09	13.15
250	2.71	3.61	4.82	8.86	16.89
500	5.05	6.78	8.24	13.12	22.48
1,000	9.54	12.03	15.42	23.27	33.47
2,500	23.87	30.49	36.51	50.71	70.16
5,000	47.6	58.15	70	99.83	130.38
10,000	97.52	119.34	147.08	194.98	248.05
Log Size ↓			***10 activities***		
100	1.97	3.13	4.96	12.22	28.59
250	3.31	4.95	7.13	15.47	32.45
500	5.84	7.99	11.05	21.08	38.32
1,000	11.71	14.23	18.23	30.59	50.66
2,500	25.86	34.34	40.34	60.47	90.27
5,000	51.91	64.92	80.63	110.93	157.23
10,000	106.41	129.41	157.58	214.94	289.57
Log Size ↓			***15 activities***		
100	2.46	3.37	5.62	14.03	29.46
250	3.86	5.63	8.15	16.32	33.84
500	6.67	10.8	12.1	22.59	40.58
1,000	12.58	16.09	20.31	33.6	54.18
2,500	28.96	37.09	46.5	66.5	97.41
5,000	58.51	72	87.61	122.5	166.1
10,000	115.44	142.93	176.29	237.81	311.67
Log Size ↓			***20 activities***		
100	2.65	4.17	6.24	15.67	33.1
250	4.6	6.7	9.29	22.96	37.86
500	7.7	10.84	14.1	25.33	46.57
1,000	13.95	18.13	24.71	37.47	62.34
2,500	33.92	42.07	53.74	77.21	111.13
5,000	68.72	84.84	102.84	145.04	196.41
10,000	137.77	171.12	203.24	280.06	363.04

where Φ is the set of input formulas to be checked. Then, we run Alloy on the resulting theory, i.e., $\Gamma_\tau \cup \{\phi\}$, and the interpretation domain Δ, concluding that τ satisfies the input specification Φ iff the theory admits a model where R is true (by construction, if such a model exists is unique).

Example 5.1. As an example, consider the constraints from Example 4.1:

$$\Phi = \{Existence(\mathsf{a}, t > 3 \wedge \forall n, v. AttV(i, n, v) \rightarrow v \neq \mathsf{d}), Response(\mathsf{a}, \varphi_a, \mathsf{b}, \varphi_c)\},$$

where:

- $\varphi_a = t \geq 1 \wedge AttV(i, \mathtt{att}_1, v_1)$, and
- correlation condition $\varphi_c = AttV(i', \mathtt{att}_2, v_2) \wedge v_1 \neq v_2$

Table 5. Time (in seconds) required for executing queries (without data conditions) of different types on traces of different lengths and with alphabet of possible activities of different sizes.

Trace length →	20	25	30	40	50
Alphabet Size ↓			**Response[?X,?Y]**		
5	0.2	1	1.05	2.04	2.18
10	0.4	0.53	0.69	1.42	2.67
15	0.6	1.19	0.98	1.46	2.48
20	0.89	1.02	1.35	2.06	3.06
Alphabet Size ↓			**Existence[?X]**		
5	0.1	0.15	0.17	0.23	0.35
10	0.1	0.14	0.16	0.24	0.34
15	0.11	0.16	0.21	0.3	0.43
20	0.14	0.18	0.23	0.4	0.55
Alphabet Size ↓			**Absence[?X]**		
5	$4 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$7 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.11
10	$5 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.1	0.11	0.17
15	$6 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.11	0.18	0.24
20	0.1	0.12	0.13	0.2	0.31
Alphabet Size ↓			**RespondedExistence[?X,?Y]**		
5	0.12	0.14	0.16	0.19	0.2
10	0.3	0.35	0.48	0.54	0.62
15	0.7	0.85	1.1	1.2	1.47
20	1.32	1.63	1.8	2.37	2.67
Alphabet Size ↓			**AlternateResponse[?X,?Y]**		
5	1.02	3.41	6.96	26.99	75.44
10	1.25	3.05	7.33	28.48	80.16
15	1.54	3.57	8.4	30.26	82.92
20	2.14	4.6	9.4	33.1	87.4
Alphabet Size ↓			**ChainResponse[?X,?Y]**		
5	$7 \cdot 10^{-2}$	0.14	0.1	0.15	0.29
10	$7 \cdot 10^{-2}$	0.1	0.16	0.23	0.4
15	0.12	0.15	0.22	0.33	0.53
20	0.17	0.21	0.3	0.44	0.71
Alphabet Size ↓			**Precedence[?X,?Y]**		
5	$7 \cdot 10^{-2}$	0.1	0.13	0.2	0.34
10	0.18	0.21	0.27	0.42	0.63
15	0.39	0.48	0.53	0.8	1.14
20	0.69	0.91	1.19	1.48	1.9
Alphabet Size ↓			**AlternatePrecedence[?X,?Y]**		
5	0.94	2.68	6.21	25.66	77.58
10	1.25	2.93	6.97	27.03	81.8
15	1.47	3.38	8.07	28.32	86.71
20	2.1	4.26	8.53	30.39	90.94
Alphabet Size ↓			**ChainPrecedence[?X,?Y]**		
5	$6 \cdot 10^{-2}$	0.1	0.1	0.16	0.37
10	$8 \cdot 10^{-2}$	0.14	0.19	0.33	0.51
15	0.15	0.2	0.19	0.32	0.53
20	0.18	0.21	0.26	0.54	0.71
Alphabet Size ↓			**NotRespondedExistence[?X,?Y]**		
5	$4 \cdot 10^{-2}$	$6 \cdot 10^{-2}$	$7 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.12
10	$7 \cdot 10^{-2}$	0.11	0.11	0.14	0.23
15	$9 \cdot 10^{-2}$	0.12	0.14	0.29	0.28
20	$8 \cdot 10^{-2}$	0.15	0.14	0.23	0.3
Alphabet Size ↓			**NotResponse[?X,?Y]**		
5	$8 \cdot 10^{-2}$	0.15	0.25	0.62	1.44
10	0.1	0.17	0.3	0.76	1.64
15	0.43	0.41	0.65	1.18	1.91
20	0.84	0.85	0.91	1.4	2.41
Alphabet Size ↓			**NotChainResponse[?X,?Y]**		
5	$8 \cdot 10^{-2}$	0.13	0.2	0.24	0.67
10	0.27	0.31	0.46	0.69	0.92
15	0.68	0.73	0.95	1.5	2.04
20	1.33	1.7	2	2.63	3.29

Table 6. Time (in seconds) required for executing queries (with activation conditions) of different types on traces of different lengths and with alphabet of possible activities of different sizes.

Trace length →	20	25	30	40	50
Alphabet Size ↓			**Response[?X,?Y]|?|**		
5	0.12	0.18	0.28	0.85	1.88
10	0.27	0.38	0.54	1	1.91
15	0.46	0.62	0.91	1.56	2.75
20	0.86	1.2	1.31	2.49	3.54
Alphabet Size ↓			**Existence[?X]|?**		
5	0.1	0.1	0.14	0.21	0.26
10	0.11	0.2	0.21	0.26	0.36
15	0.13	0.17	0.21	0.32	0.46
20	0.15	0.21	0.28	0.42	0.62
Alphabet Size ↓			**Absence[?X]|?**		
5	$5 \cdot 10^{-2}$	$7 \cdot 10^{-2}$	$7 \cdot 10^{-2}$	$9 \cdot 10^{-2}$	0.15
10	$9 \cdot 10^{-2}$	0.11	0.14	0.15	0.21
15	0.13	0.16	0.19	0.21	0.3
20	0.16	0.22	0.23	0.28	0.36
Alphabet Size ↓			**RespondedExistence[?X,?Y]|?|**		
5	0.12	0.2	0.2	0.24	0.3
10	0.36	0.56	0.74	0.82	0.92
15	0.81	1.07	1.21	1.89	2.27
20	1.41	2.49	2.19	3.14	3.94
Alphabet Size ↓			**AlternateResponse[?X,?Y]|?|**		
5	1.03	2.78	6.79	26.16	75.04
10	1.22	3.15	8.51	28.62	78.96
15	1.62	3.8	8.42	31.07	83.15
20	2.23	4.89	9.71	34.06	89.32
Alphabet Size ↓			**ChainResponse[?X,?Y]|?|**		
5	$6 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.15	0.24	0.57
10	$8 \cdot 10^{-2}$	0.12	0.15	0.29	0.52
15	0.14	0.16	0.22	0.38	0.72
20	0.25	0.33	0.33	0.63	0.96
Alphabet Size ↓			**Precedence[?X,?Y]|?|**		
5	$8 \cdot 10^{-2}$	0.16	0.14	0.3	0.56
10	0.2	0.26	0.32	0.56	0.77
15	0.45	0.52	0.63	0.97	1.27
20	0.83	1.02	1.16	1.58	2.18
Alphabet Size ↓			**AlternatePrecedence[?X,?Y]|?|**		
5	0.94	2.58	6.36	26.37	78.67
10	1.11	3	6.92	26.99	83.15
15	1.57	3.74	8.4	28.71	86.37
20	2.21	4.74	9.21	31.29	90.13
Alphabet Size ↓			**ChainPrecedence[?X,?Y]|?|**		
5	$7 \cdot 10^{-2}$	$8 \cdot 10^{-2}$	0.1	0.19	0.42
10	$8 \cdot 10^{-2}$	0.12	0.19	0.33	0.49
15	0.14	0.18	0.23	0.37	0.61
20	0.2	0.24	0.3	0.5	0.82
Alphabet Size ↓			**NotRespondedExistence[?X,?Y]|?|**		
5	$9 \cdot 10^{-2}$	$7 \cdot 10^{-2}$	0.12	0.12	0.14
10	0.31	0.23	0.28	0.3	0.4
15	0.77	0.81	1.35	0.82	0.88
20	1.54	1.42	2.07	1.81	1.9
Alphabet Size ↓			**NotResponse[?X,?Y]|?|**		
5	0.14	0.22	0.32	0.85	1.56
10	0.39	0.43	0.55	1.09	2
15	1.21	1.24	1.28	1.69	2.84
20	2.21	2.39	2.91	3.18	4.3
Alphabet Size ↓			**NotChainResponse[?X,?Y]|?|**		
5	0.17	0.22	0.26	0.44	0.73
10	0.6	0.8	0.93	1.26	1.51
15	1.46	1.85	1.97	2.38	3.21
20	2.83	3.24	3.67	4.87	6.32

Table 7. Time (in seconds) required for executing queries (with activation and correlation conditions) of different types on traces of different lengths and with alphabet of possible activities of different sizes.

Trace length →	20	25	30	40	50
Alphabet Size ↓			**Response[?X,?Y]|?|?**		
5	0.23	0.39	0.5	0.87	1.94
10	0.87	1.08	1.24	1.8	2.8
15	2.44	2.5	2.96	3.58	5.52
20	4.72	5.32	6.33	8.12	9.43
Alphabet Size ↓			**RespondedExistence[?X,?Y]|?|?**		
5	0.25	0.3	0.33	0.42	0.51
10	1.04	1.31	1.39	1.84	2.09
15	2.8	3.28	3.23	4.18	4.96
20	4.52	5.1	5.97	7.26	8.44
Alphabet Size ↓			**AlternateResponse[?X,?Y]|?|?**		
5	1.22	3.19	7.42	28.38	78.61
10	2.02	4.15	8.74	31.06	84.86
15	3.87	6.13	11.38	34.87	90.53
20	6.25	8.97	16.38	40.47	100.11
Alphabet Size ↓			**ChainResponse[?X,?Y]|?|?**		
5	0.18	0.18	0.24	0.28	0.51
10	0.73	0.57	0.66	0.72	1
15	1.84	2.02	2.18	1.86	2.32
20	4.19	3.43	6.25	5.36	7.05
Alphabet Size ↓			**Precedence[?X,?Y]|?|?**		
5	0.26	0.38	0.41	0.52	0.79
10	1.07	1.12	1.31	1.62	2.12
15	2.63	2.79	3.13	3.87	4.7
20	5.18	6.16	6.65	8.15	9.45
Alphabet Size ↓			**AlternatePrecedence[?X,?Y]|?|?**		
5	1.18	3.46	8.53	42.95	91.01
10	2.02	3.94	8.22	29.95	88.06
15	3.56	5.95	10.71	33.66	93.54
20	6.47	9.09	14.12	37.93	101.58
Alphabet Size ↓			**ChainPrecedence[?X,?Y]|?|?**		
5	0.18	0.15	0.27	0.28	0.4
10	0.68	0.5	0.59	0.75	1.01
15	1.91	2.07	2.01	1.68	2.28
20	3.74	3.44	5.21	4.35	4.86
Alphabet Size ↓			**NotRespondedExistence[?X,?Y]|?|?**		
5	0.22	0.16	0.24	0.26	0.22
10	0.94	0.79	0.84	0.93	1.2
15	2.57	2.65	2.78	2.31	2.75
20	4.35	4.88	5.26	6.31	7.48
Alphabet Size ↓			**NotResponse[?X,?Y]|?|?**		
5	0.27	0.32	0.46	0.87	1.66
10	1.07	1.05	1.27	1.72	2.88
15	3.02	3.5	3.58	3.96	5.13
20	4.17	5.05	5.58	7.38	10.34
Alphabet Size ↓			**NotChainResponse[?X,?Y]|?|?**		
5	0.3	0.35	0.42	0.56	0.81
10	1.33	1.64	1.74	2.4	3
15	3.08	3.68	4.14	5.15	6.16
20	4.2	4.79	6.18	6.98	8.63

are, respectively, the activation and correlation condition of *Response*. Consider also the trace of Example 3.1:

$$\tau_1 = \mathsf{a}_1(\mathsf{t}=2, \mathsf{v}=\mathsf{a})\ \mathsf{a}_2(\mathsf{t}=5, \mathsf{w}=\mathsf{b}, \mathsf{y}=\mathsf{d})\ \mathsf{a}_3(\mathsf{t}=9, \mathsf{s}=\mathsf{c})\ \mathsf{a}_3(\mathsf{t}=20, \mathsf{s}=\mathsf{d}).$$

The theory Γ_{τ_1} includes the following sentences:

- $Seq(0, \mathsf{a}_1, 2) \wedge Seq(1, \mathsf{a}_2, 5) \wedge Seq(2, \mathsf{a}_3, 9) \wedge Seq(3, \mathsf{a}_3, 20)$
- $AttV(0, \mathsf{v}, \mathsf{a}) \wedge AttV(1, \mathsf{w}, \mathsf{b}) \wedge AttV(1, \mathsf{y}, \mathsf{d}) \wedge AttV(2, \mathsf{s}, \mathsf{c}) \wedge AttV(3, \mathsf{s}, \mathsf{d})$
- $N(0) \wedge N(1) \wedge N(2) \wedge N(3) \wedge N(5) \wedge N(9) \wedge N(20)$
- $\forall i, a, t.Seq(i, a, t) \rightarrow (i = 0 \wedge a = \mathsf{a}_1 \wedge t = 2) \vee (i = 1 \wedge a = \mathsf{a}_2 \wedge t = 5) \vee (i = 2 \wedge a = \mathsf{a}_3 \wedge t = 9) \vee (i = 3 \wedge a = \mathsf{a}_3 \wedge t = 20)$
- $\forall i, n, v.AttV(i, n, v) \rightarrow AttV(i = 0, n = \mathsf{v}, v = \mathsf{a}) \vee AttV(i = 1, n = \mathsf{w}, v = \mathsf{b}) \vee AttV(i = 1, n = \mathsf{y}, v\mathsf{d}) \vee AttV(i = 2, n = \mathsf{s}, v = \mathsf{c}) \vee AttV(i = 3, n = \mathsf{s}, v = \mathsf{d})$
- $\forall i, .N(i) \rightarrow N(0) \vee N(1) \vee N(2) \vee N(3) \vee N(5) \vee N(9) \vee N(20)$.

Observe that, for fixed $\Delta = C = \{0, 1, 2, 3, 5, 9, 20, \mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \mathsf{v}, \mathsf{w}, \mathsf{y}, \mathsf{s}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$, the only model of Γ_{τ_1} (which represents τ_1) is also one of Γ.

We then define $\phi = \textsc{Existence}(\mathsf{a}_2, \varphi) \wedge \textsc{Response}(\mathsf{a}_1, \varphi_a, \mathsf{a}_2, \varphi_c)$, for $\varphi = t > 3 \wedge \forall n, v.AttV(i, n, v)$, and φ_a and φ_c as above.

Then, using Alloy, we search for a model of $\Gamma_{\tau_1} \cup \{\phi\}$, providing Δ as input. In the case at hand, the returned (unique) model will have R true, indicating that τ satisfies Φ.

5.3 Query Checking

For query checking, we proceed in a way similar to that of conformance checking, which, is in fact, a special case of query checking with a Boolean query.

Given a trace $\tau \in L$, we produce the associated theory Γ_τ, which admits a unique model, as explained in 5.2. Then, given the query $\varphi(\vec{x})$ (with free variables $\vec{x}$), we add:

- a new predicate symbol *Ans* with arity $|\vec{x}|$ to σ, then
- the following sentence to Γ_τ:

$$\forall \vec{x}.Ans(\vec{x}) \leftrightarrow \varphi(\vec{x}).$$

As a result, the (unique) model of the resulting theory $\Gamma_\tau \cup \{\varphi\}$ coincides with that of Γ_τ but additionally includes relation *Ans*, which, by the above definition, contains all and only the tuples occurring in φ^ι, for ι the unique model of Γ_τ. By feeding Alloy with the theory $\Gamma_\tau \cup \{\varphi\}$ and the interpretation domain associated with τ, we can easily answer the query.

Example 5.2. For an example of query checking, consider the problem defined in Example 4.3, where the query

$$\varphi = \forall i, t.(\forall \vec{x}.((Seq(i, ?x, t) \wedge \varphi_a) \rightarrow$$

$$(\exists i', t'.i' \geq i \wedge (\exists \vec{x}'.Seq(i', ?y, t') \wedge \varphi_c))))$$

is evaluated against the trace of Example 3.1: $\tau_1 = \mathsf{a}_1(\mathsf{t}=2, \mathsf{v}=\mathsf{a}), \mathsf{a}_2(\mathsf{t}=5, \mathsf{w}=\mathsf{b}, \mathsf{y}=\mathsf{d}), \mathsf{a}_3(\mathsf{t}=9, \mathsf{s}=\mathsf{c}), \mathsf{a}_3(\mathsf{t}=20, \mathsf{s}=\mathsf{d})$.

To answer the query, we simply feed Alloy with the theory obtained by joining Γ_{τ_1} shown in Example 5.1 plus the sentence $\forall \vec{x}.Ans(\vec{x}) \leftrightarrow \varphi$, and with the interpretation domain Δ associated with τ, then ask Alloy to find a model of the input theory. The content of relation *Ans* corresponds to the sought answer to φ.

6 EXPERIMENTS

We implemented the framework described in this paper in a tool available at https://github.com/darksoullock/MPDeclareLogGenerator/. The efficiency of the tool was tested by performing several experiments measuring the execution times needed for the tool to perform tasks of different complexity. It is worth noticing that we do not use baselines. This is due to the fact that there are no tools in the Process Mining literature that can be used as baseline for the (data-aware) log generation and query checking tasks. On the other hand, the tool presented in [10] for data-aware conformance checking is generally more efficient than the SAT-based solution.[7] Each measured execution time was averaged over 3 runs. The experiments were conducted on a single core of a 64-bit 2.2 Ghz Intel Core i5-5200U processor with 16GB of RAM. The SAT solver used was SAT4J.

In order to assess the performance of the log generation approach, we generated logs with different characteristics using different Declare models. In a first set of experiments, we sampled the generation times that resulted by varying the following parameters: *(i)* the number of constraints in the Declare model (5, 10, 15, and 20 constraints), *(ii)* the number of events per trace (5, 10, 15, 20, 25, 30, 40, and 50 events), and *(iii)* the number of traces in the log (100, 250, 500, 1000, 2500, 5000, and 10 000 traces). The number of activities in the Declare models is fixed to 10. The constraints employed are all of type *response*. The constraints are considered without data conditions, with activation conditions, and with activation and correlation conditions. The results of our experimentation are summarized in Tables 1-7.

When using a Declare model including constraints without data conditions, in the worst case, namely to generate a log of 10 000 traces of length 50 from a model containing 20 constraints, the execution time is slightly above 2 minutes. When using constraints with activation conditions, the execution time in the worst case is of around 4.5 minutes. In the case of constraints with activation and correlation conditions the execution time in the worst case is of 5.5 minutes. In all the tested cases, the peak of the RAM consumption is around 400 MB.

We also tested the trend of the execution times when varying the alphabet size (5, 10, 15, and 20 activities) of the Declare models (without data conditions). The number of constraints in these experiments is fixed to 20. As expected, for increasing alphabet sizes, the execution times increase. In the worst case, namely to generate a log of 10 000 traces of length 50 from a model containing 20 activities, the execution time is of around 6 minutes.

Finally, we executed a set of experiments to test the performance of query checking.[8] We sampled the query times by varying the following parameters: *(i)* the type of query, *(ii)* the number of events in the input trace (20, 25, 30, 40, and 50 events), and *(iii)* the alphabet of possible activities (5, 10, 15, and 20 activities). The queries cover all the standard Declare constraints and are considered without data conditions, with activation conditions, and with activation and correlation conditions. For all query types the execution times range from few milliseconds to 10 seconds. The only exception is the case of the queries of type *alternate* that require, in the worst case, more than 1.5 minutes.

To show what type of insights can be discovered with the presented approach, and to check its scalability in real-life scenarios, we applied our tool to analyze a real-life event log containing events of sepsis cases from a hospital. Sepsis is a life threatening condition typically caused by an infection. One case represents the pathway through the hospital. The events were recorded by the ERP (Enterprise Resource Planning) system of the hospital. There are about 1000 cases with in total 15,000 events that were recorded for 16 different activities. Moreover, 39 data attributes are recorded, e.g., the group responsible for the activity, the results of tests and information from checklists.

[7] As mentioned in the introduction, the disadvantage of this solution is that it requires to implement a dedicated algorithm for any rule that needs to be checked (only the algorithms implementing the semantics of the Declare set of rules are publicly available).

[8] We did not assess the conformance checking tool explicitly since, as already mentioned, conformance checking is a special case of query checking.

Starting from the description of the log provided by domain experts in [43], we have formulated the following queries:

- *Response[CRP,?Y]|?|*
- *Response[Leucocytes,?Y]|?|*
- *Response[LacticAcid,?Y]|?|*

allowing us to understand what happens in a process execution when three different types of tests (CRP, Leucocytes and LacticAcid) are carried out with certain results. In the following paragraphs, we summarize our main findings. The complete lists of discovered constraints are available at https://github.com/darksoullock/MPDeclareLogGenerator/tree/master/data/2020.

Executing query *Response[CRP,?Y]|?|* on the real-life log under analysis took 5 minutes and 5 seconds. This query allows domain experts to carry on an analysis on what can happen when the CRP test is conducted. From the results, we can see that two main classes of patients are available. The ones of age between 19 and 41 for which the CRP varies between values 4.0 and 172.0 and the ones of age between 42 and 66 for which the CRP varies between values 340.0 and 508.0. For younger patients the diagnosis is *E* for the older ones is *GA*. Both classes of patients are required to carry on the same tests, e.g., *Leucocytes* and *LacticAcid*.

Executing query *Response[Leucocytes,?Y]|?|* took 4 minutes and 58 seconds. From the results obtained with this query, we can see that, here, there there are 3 clusters of patients. The ones of age between 19 and 41, the ones of age between 42 and 66 and the ones of age between 67 and 91. For all of them *Leucocytes* can vary between values -0.8 and 126.89 indicating that the result of this type of test does not depend on the age of the patient. The diagnosis in this case can be different for patients of the same age. However, for younger patients the diagnosis can be only *E* or *K*, whereas for the older ones there is much more variety in terms of possible diagnosis.

Executing query *Response[LacticAcid,?Y]|?|* took 5 minutes and 19 seconds. Here, the patients are not clustered anymore based on their age, but only based on the resulting values of the test. When the resulting value of the test varies between -0.8 and 4.76, the possible actions eventually executed can be *Release_B*, *Release_D*, *LacticAcid*, *IV_Liquid*, *CRP*, *ER_Triage*, *Leucocytes*, *Release_A*, *Release_C*, *Release_E*, *IV_Antibiotics*, $ER\_Sepsis_{T}riage$, *Admission_IC*, *Admission_NC*, and *Return_ER*. When the resulting value of the test varies between 4.76 and 10.33, the possible actions eventually executed are the same except *Release_C* and *Release_E*. When the resulting value of the test varies between 10.33 and 15.89, the possible actions eventually executed can be *CRP*, *Leucocytes*, and *Release_B*.

As for memory consumption, the following peak of RAM used was measured during the testing of the three real-world queries:

- *Response[CRP,?Y]|?|*: 489 MB.
- *Response[Leucocytes,?Y]|?|*: 503 MB.
- *Response[LacticAcid,?Y]|?|*: 549 MB.

These results enable us to conclude that memory consumption seems not to be a problem for enacting query checking, since the peak of used memory is far from saturate the RAM available in the machine used for the experiments.

7 RELATED WORK

Over the years, several propositional SAT based encodings have been proposed to deal with various data mining challenges including itemset and sequence mining problems. Among the most relevant and recent works on this matter, in [25, 26], a declarative framework for mining high utility itemsets from transaction databases is presented. Specifically, the frequent itemset mining problem is first reformulated as a propositional satifiability one, and then solved employing a SAT solver. Similarly, in [47], an efficient parallel SAT-based encoding for addressing the itemsets mining problem in presence of large input datasets is proposed. In [31], the authors show how the

community detection algorithms in social network analysis can be expressed as a Partial Max-SAT optimization problem and solved using SAT tools. The authors show that the proposed solution outperforms existing state-of-the-art methods in detecting relevant communities. In [30], the problem of mining strong negative association rules in pattern discovery is investigated through a SAT-based approach that is able to address the challenge in an efficient way.

Other studies focus on SAT implementation for efficient constrained clustering [18] and pattern discovery [46]. In particular, in [46] the authors present a constraint-based language coupled with an approach driven by query answering to discover patterns within sequences of items. Queries and built-in constraints of the language are encoded and solved using the SAT framework.

While the above approaches rely on a SAT based reformulation of relevant data mining problems, i.e., they investigate the existence of relations and patterns between data items, in the process mining community some works exist that employ SAT-based solutions to perform control-flow analysis on well-structured business processes. For example, in [33], a simulation-based look-ahead approach for multi-perspective declarative process models is introduced. This approach transforms the problem of a context-aware process simulation into a SAT problem, by translating a declarative multi-perspective process model and the current state of a process execution into a specification of the logic language Alloy. Via a SAT solver, process trajectories are generated that either satisfy or violate this specification. The simulated process trajectories are used to derive consequences and effects of certain decisions at any time of process execution. In [6], the authors show how important conformance artefacts like alignments, anti-alignments and multi-alignments can be computed by encoding the problem as a SAT instance, advocating for a unified family of techniques that can compute conformance artefacts in the same way.

Following the research direction traced by [5], in our contribution we demonstrate that SAT can be considered as a suitable solving technology for three relevant problems in Declarative Process Mining, namely log generation, conformance checking and temporal query checking. In the literature, different extensions of Declare have been proposed. Some of these extensions have been used as a basis to develop Process Mining algorithms. In [42, 61], the authors introduce *Timed Declare*, an extension of Declare whose semantics is based on a metric temporal logic that allows Declare constraints to be enriched with temporal conditions on timestamps. In [38], such extension is used for the discovery of Timed Declare constraints.

In the work proposed in [39], another extension of Declare has been proposed whose semantics is defined by using a first-order variant of LTL that allows Declare constraints to be enriched with conditions on data. This semantics has been used in [7, 39] for the development of process discovery algorithms that produce data-aware Declare constraints from logs. In [10], MP-Declare is introduced and a technique for conformance checking based on MP-Declare is presented. Techniques for the discovery of MP-Declare constraints from event logs are presented in [35, 55].

Lines of research that are also related to the analysis of logs with declarative patterns are the ones concerning sequential pattern mining and association rules. For example, Bose and van der Aalst in [8, 9] employ sequential pattern mining to discover sequential patterns as tandem repeats, maximal repeats and alphabet repeats to be used as features for training a classifier. Similarly, in [34], association rules are used to discover co-occurrence patterns in the context of deviant classes in a healthcare scenario. In [12, 36], discriminative mining is used to discover discriminative patterns, i.e., patterns that, although not necessarily very frequent, clearly discriminate between deviant and non-deviant traces. A benchmark collecting all these works and evaluating and comparing them in terms of different feature types and classifiers is presented in [48, 49].

Different types of patterns have also been combined together in [14–17]. In particular, in [14], in order to avoid the redundant representation deriving from mixing different families of patterns, the authors propose an ensemble learning approach in which multiple learners are trained encoding the log according to different types of patterns. In [15], data attributes have also been taken into account in the discovery phase as well as for training the classifier.

Finally, in [16, 17], the authors enhance their previous work [14] by proposing an alternative multi-learning approach probabilistically combining various classification methods.

The difference between the above approach and the declarative process mining approaches is in the type of patterns that are used for representing the log characteristics. While in association rules and sequential pattern mining patterns are expressed as item sets and n-grams, in declarative process mining they are expressed as temporal rules.

8 CONCLUSIONS

Process mining is the leading Big Data technology for business process analysis [57]. By extracting knowledge from event logs in information systems, process mining builds a digital footprint of a company's real workflow enabling consultants and data scientists to concretely improve overall company performance, revenues, and profitability. This paper presents a framework that takes advantage of AI techniques (bounded FO reasoning and SAT) to solve problems in Declarative Process Mining, a research area particularly relevant for realizing intelligent Business Process Management Systems to effectively enact knowledge-intensive processes [22, 44, 45].

We have shown how three specific Data-Aware Declarative Process Mining problems (namely log generation, conformance checking and query checking) can be expressed as suitable FOL theories and solved by compilation into SAT using Alloy. Such a solution approach yields two great advantages. Firstly, since by using Alloy we can translate any instance of Log Generation, Conformance Checking and Query Checking, into a SAT encoding, it follows that we can use any SAT solver for such problems. This, in turn, allows for taking advantage of all possible future improvements in the SAT technology for the problems of our interest. Secondly, our approach is more general than existing ones. Indeed, previous approaches for Declarative Process Mining, and more specifically for Log Generation, Conformance Checking and Query Checking, are tailored for MP-Declare. As we have shown in the paper, we can rewrite all the MP-Declare templates as formulas in FOL (interpreted over a finite interpretation domain), which Alloy then translates into SAT; however, Alloy can translate any arbitrary FOL formula, not only those encoding MP-Declare templates, thus, since MP-Declare is strictly less expressive than FOL, our approach can deal with a strictly larger set of specifications.

The choice of adopting FOL theories and the SAT technology is motivated by the need to both express the process specification at the correct abstraction level required to dealing with processes in the presence of data and to perform automated reasoning over the specified process. The use of FOL and the SAT technology has allowed us to develop a very clean and simple-to-manage framework for Declarative Process Mining based on relevant data manipulated by the process, without compromising efficiency and effectiveness of the proposed solution.

As future work, we would like to compare the performance of our tool using different SAT solvers. We are currently working on an approach based on SAT for monitoring the compliance of ongoing process executions with respect to a set of MP-Declare constraints.

ACKNOWLEDGMENTS

This research is supported by the PNRR MUR project PE0000013-FAIR, the H2020 project DataCloud (Grant number 101016835), and the UNIBZ project CAT.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. 2011. Conformance Checking Using Cost-Based Fitness Analysis. In *IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011*. IEEE, 55–64.

[3] Anti Alman, Claudio Di Ciccio, Dominik Haas, Fabrizio Maria Maggi, and Alexander Nolte. 2020. Rule Mining with RuM. In *2nd International Conference on Process Mining, ICPM 2020*. IEEE, 121–128.

[4] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. 2019. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Trans. Knowl. Data Eng.* 31, 4 (2019), 686–705.
[5] Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. 2019. Encoding Conformance Checking Artefacts in SAT. In *Business Process Management Workshops*. Springer, 160–171.
[6] Mathilde Boltenhagen, Thomas Chatain, and Josep Carmona. 2021. Optimized SAT encoding of conformance checking artefacts. *Computing* 103, 1 (2021), 29–50.
[7] R. P. Jagadeesh Chandra Bose, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. 2013. Enhancing Declare Maps Based on Event Correlations. In *11th International Conference on Business Process Management, BPM 2013*. Springer, 97–112.
[8] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. 2009. Abstractions in Process Mining: A Taxonomy of Patterns. In *7th International Conference on Business Process Management, BPM 2009*. Springer, 159–175.
[9] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. 2013. Discovering signature patterns from event logs. In *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013*. IEEE, 111–118.
[10] Andrea Burattin, Fabrizio Maria Maggi, and Alessandro Sperduti. 2016. Conformance checking based on multi-perspective declarative process models. *Expert Syst. Appl.* 65 (2016), 194–211.
[11] Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. 2018. *Conformance Checking - Relating Processes and Models*. Springer.
[12] Ning Chen, Steven C. H. Hoi, and Xiaokui Xiao. 2011. Software process evaluation: A machine learning approach. In *26th International Conference on Automated Software Engineering, ASE 2011*. IEEE, 333–342.
[13] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *3rd Annual ACM Symposium on Theory of Computing, STOC 1971*. ACM, 151–158.
[14] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. 2015. A Multi-view Learning Approach to the Discovery of Deviant Process Instances. In *OTM Conferences*. 146–165.
[15] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. 2016. A multi-view multi-dimensional ensemble learning approach to mining business process deviances. In *2016 International Joint Conference on Neural Networks, IJCNN 2016*. 3809–3816.
[16] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. 2016. A Robust and Versatile Multi-View Learning Framework for the Detection of Deviant Business Process Instances. *Int. J. Cooperative Inf. Syst.* 25, 4 (2016).
[17] Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. 2017. Extensions, Analysis and Experimental Assessment of a Probabilistic Ensemble-learning Framework for Detecting Deviances in Business Process Instances. In *19th International Conference on Enterprise Information Systems, ICEIS 2017*. 162–173.
[18] Ian Davidson, SS Ravi, and Leonid Shamis. 2010. A SAT-based framework for efficient constrained clustering. In *2010 SIAM International Conference on Data Mining, SDM 2010*. 94–105.
[19] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *23rd International Joint Conference on Artificial Intelligence*. IJCAI 2013, 854–860.
[20] Riccardo De Masellis, Fabrizio Maria Maggi, and Marco Montali. 2014. Monitoring data-aware business constraints with finite state automata. In *International Conference on Software and Systems Process, ICSSP 2014*. 134–143.
[21] Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. 2015. Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. *Journal on Data Semantics* 4, 1 (2015), 29–57.
[22] Marlon Dumas, Fabiana Fournier, Lior Limonad, Andrea Marrella, Marco Montali, Jana-Rebecca Rehse, Rafael Accorsi, Diego Calvanese, Giuseppe De Giacomo, Dirk Fahland, Avigdor Gal, Marcello La Rosa, Hagen Völzer, and Ingo Weber. 2023. AI-Augmented Business Process Management Systems: A Research Manifesto. *ACM Trans. Manage. Inf. Syst.* 14, 1 (2023).
[23] Vijay Ganesh and Moshe Y. Vardi. 2020. On the Unreasonable Effectiveness of SAT Solvers. In *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 547–566.
[24] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. 2007. *Finite Model Theory and Its Applications*. Springer.
[25] Amel Hidouri, Said Jabbour, Badran Raddaoui, and Boutheina Ben Yaghlane. 2020. A SAT-based approach for mining high utility itemsets from transaction databases. In *22nd International Conference on Big Data Analytics and Knowledge Discovery, DaWaK 2020*. Springer, 91–106.
[26] Amel Hidouri, Said Jabbour, Badran Raddaoui, and Boutheina Ben Yaghlane. 2021. Mining closed high utility itemsets based on propositional satisfiability. *Data & Knowledge Engineering* 136 (2021).
[27] Joris Hulstijn and Jaap Gordijn. 2010. Risk Analysis for Inter-organizational Controls. In *12th International Conference on Enterprise Information Systems, ICEIS 2010*. Springer.
[28] IEEE Task Force on Process Mining. 2011. Process Mining Manifesto. In *Proc. BPM Workshops (Lecture Notes in Business Information Processing, Vol. 99)*. Springer, 169–194.
[29] IEEE Task Force on Process Mining. 2013. *XES Standard Definition*. Technical Report. http://www.xes-standard.org/xesstandarddefinition.

[30] Said Jabbour, Fatima Ezzahra El Mazouri, and Lakhdar Sais. 2018. Mining negatives association rules using constraints. *Procedia Computer Science* 127 (2018), 481–488.
[31] Said Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. 2020. SAT-based models for overlapping community detection in networks. *Computing* 102, 5 (2020), 1275–1299.
[32] Daniel Jackson. 2002. Alloy: A New Technology for Software Modelling. In *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*. 20.
[33] Martin Käppel, Lars Ackermann, Stefan Schönig, and Stefan Jablonski. 2021. Language-independent look-ahead for checking multi-perspective declarative process models. *Software and Systems Modeling* 20, 5 (2021), 1379–1401.
[34] Geetika T. Lakshmanan, Szabolcs Rozsnyai, and Fei Wang. 2013. Investigating Clinical Care Pathways Correlated with Outcomes. In *BPM (Lecture Notes in Computer Science, Vol. 8094)*. Springer, 323–338.
[35] Volodymyr Leno, Marlon Dumas, and Fabrizio Maria Maggi. 2018. Correlating Activation and Target Conditions in Data-Aware Declarative Process Discovery. In *BPM*, Vol. 11080. 176–193.
[36] David Lo, Siau-Cheng Khoo, and Chao Liu. 2007. Efficient mining of iterative patterns for software specification discovery. In *KDD*. ACM, 460–469.
[37] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. 2011. Monitoring Business Process Compliance Using Compliance Rule Graphs. In *OTM Conferences (1)*. 82–99.
[38] Fabrizio Maria Maggi. 2014. Discovering Metric Temporal Business Constraints from Event Logs. In *Perspectives in Business Informatics Research - 13th International Conference, BIR 2014, Lund, Sweden, September 22-24, 2014. Proceedings*. 261–275.
[39] Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. 2013. Discovering Data-Aware Declarative Process Models from Event Logs. In *BPM*. 81–96.
[40] Fabrizio Maria Maggi, Marco Montali, and Ubaier Bhat. 2019. Compliance Monitoring of Multi-Perspective Declarative Process Models. In *23rd IEEE International Enterprise Distributed Object Computing Conference, EDOC 2019, Paris, France, October 28-31, 2019*. 151–160.
[41] Fabrizio Maria Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. 2013. Analyzing Vessel Behavior Using Process Mining. In *Situation Awareness with Systems of Systems*. 133–148.
[42] Fabrizio Maria Maggi and Michael Westergaard. 2014. Using Timed Automata for *a Priori* Warnings and Planning for Timed Declarative Process Models. *Int. J. Cooperative Inf. Syst.* 23, 1 (2014). https://doi.org/10.1142/S0218843014400036
[43] Felix Mannhardt. 2016. Sepsis Cases - Event Log. (12 2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
[44] Andrea Marrella and Yves Lespérance. 2013. Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In *Enterprise, Business-Process and Information Systems Modeling*. Springer, 277–291.
[45] Andrea Marrella and Yves Lespérance. 2017. A planning approach to the automated synthesis of template-based process models. *Service Oriented Computing and Applications* 11 (2017), 367–392.
[46] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. 2011. A constraint-based language for declarative pattern discovery. In *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE, 1112–1119.
[47] Ikram Nekkache, Said Jabbour, Lakhdar Sais, and Nadjet Kamel. 2021. Towards a Compact SAT-Based Encoding of Itemset Mining Tasks. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 163–178.
[48] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Suriadi Suriadi. 2014. Mining Business Process Deviance: A Quest for Accuracy. In *OTM Conferences*, Vol. 8841. 436–445.
[49] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Suriadi Suriadi. 2016. Business Process Deviance Mining: Review and Evaluation. *CoRR* abs/1608.08252 (2016).
[50] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. 2007. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC*. 287–300.
[51] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. 2011. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In *BPM Workshops*. 383–394.
[52] Margus Räim, Claudio Di Ciccio, Fabrizio Maria Maggi, Massimo Mecella, and 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 - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*. 75–92.
[53] Hajo A. Reijers, Tijs Slaats, and Christian Stahl. 2013. Declarative Modeling—An Academic Dream or the Future for BPM? In *BPM 2013*. Vol. 8094. 307–322.
[54] A. Rozinat and W. M. P. van der Aalst. 2008. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems* 33, 1 (2008), 64–95.
[55] Stefan Schönig, Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. 2016. Discovery of Multi-perspective Declarative Process Models. In *Service-Oriented Computing - 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016,*

Proceedings. 87–103.
[56] Stefan Schulte, Dieter Schuller, Ralf Steinmetz, and Sven Abels. 2012. Plug-and-Play Virtual Factories. *IEEE Internet Computing* 16, 5 (2012), 78–82.
[57] Wil M. P. van der Aalst. 2016. *Process Mining - Data Science in Action, Second Edition*. Springer.
[58] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. 2012. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 2 (2012), 182–192.
[59] Miklos Vasarhelyi, Michael Alles, and Alexander Kogan. 2004. Principles of Analytic Monitoring for Continuous Assurance. *Journal of Emerging Technologies in Accounting* 1, 1 (2004), 1–21.
[60] H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2010. XES, XESame, and ProM 6. In *Information Systems Evolution - CAiSE Forum*, Vol. 72. 60–75.
[61] Michael Westergaard and Fabrizio Maria Maggi. 2012. Looking into the Future. Using Timed Automata to Provide a Priori Advice about Timed Declarative Process Models. In *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I*. 250–267.
[62] Stefan Zugal, Jakob Pinggera, and Barbara Weber. 2011. The Impact of Testcases on the Maintainability of Declarative Process Models. In *BMMDS/EMMSAD*. 163–177.