Efficient Compliance Computation in Probabilistic Declarative Specifications

Mario Alviano*, Antonio Ielo and Francesco Ricca

DeMaCS, University of Calabria, 87036 Rende (CS), Italy

Abstract

In this paper, we investigate the measurement of trace satisfaction probabilities within probabilistic Declare models, where Declare constraints are associated with probabilities. Each constraint, with a probability p, is independently included in a model with probability p or excluded with probability $1 - p$. This probabilistic framework creates multiple possible worlds, each corresponding to a specific selection of constraints, with the probability of each world calculated as the product of the probabilities of its included constraints. A trace can be satisfied by some of these worlds, and the probability that a trace is satisfied is the sum of the probabilities of all satisfying worlds. We develop techniques to compute this satisfaction probability by integrating a tool that determines trace satisfaction for crisp Declare models with an implementation of the inclusion-exclusion principle. Our preliminary experiments compare the performance of our prototype with and without the inclusion-exclusion principle, highlighting its impact on the efficiency and accuracy of the probability computations. The results demonstrate the potential of our approach in enhancing the analysis of probabilistic Declare models, providing a foundation for more sophisticated probabilistic reasoning in process mining.

Keywords

Process Mining, Declare, Probabilistic Reasoning

1. Introduction

The modeling and analysis of business processes are crucial for organizations seeking to improve efficiency, compliance, and adaptability in a dynamic environment [1]. Traditional approaches to process modeling often rely on deterministic frameworks where every aspect of the process is explicitly defined. However, real-world processes are frequently subject to variability and uncertainty, necessitating the incorporation of probabilistic elements into process models. Probabilistic Declare [2] is an extension of the Declare [3] modeling language that addresses this need by associating constraints with probabilities, thereby allowing for the representation of uncertainty in process execution.

Declare is a declarative process modeling language that allows the specification of constraints on the execution of activities within a process. Unlike imperative process modeling languages that prescribe a specific sequence of activities, Declare defines what must or must not happen, thereby offering greater flexibility. Constraints in Declare can represent various types of conditions, such as precedence (an activity must precede another), response (an activity must be followed by another), and exclusion (two activities cannot occur together). This flexibility makes Declare particularly suitable for complex, flexible, and dynamic processes.

In a probabilistic Declare model, each constraint is assigned a probability p representing the likelihood that the constraint is included in the model. This probabilistic nature acknowledges the inherent uncertainty and variability in real-world processes, where constraints might not always be strictly enforced or might only apply under certain conditions. The inclusion or exclusion of each constraint is assumed to be independent, leading to a combinatorial explosion of possible worlds—each corresponding to a unique subset of constraints. The probability of a particular world is the product of the probabilities of the constraints it includes. For a given trace (a sequence of executed activities), the probability that the trace is satisfied by the probabilistic Declare model is the sum of the probabilities of all worlds in which the trace satisfies the constraints. This probabilistic framework introduces significant challenges in measuring trace satisfaction probabilities due to the exponential number of possible worlds.

PLP 2024: 11th Workshop on Probabilistic Logic Programming, October, 2024, Dallas, USA.

*Corresponding author.

The primary objective of this paper is to develop and evaluate techniques for measuring the probability that a trace is satisfied by a probabilistic Declare model. To achive this objective, we use an existing logic-programming-based tool [4] capable of determining whether a trace satisfies a set of crisp Declare constraints and apply the inclusion-exclusion principle [5] to efficiently compute the satisfaction probabilities, mitigating the combinatorial complexity. By adapting tools for crisp Declare to work within the probabilistic framework, we ensure that the underlying logic for constraint satisfaction remains robust and accurate, and can evolve indepedenly from the probabilistic setting. Regarding the inclusion-exclusion principle, it is a mathematical technique used to calculate the probability of the union of multiple events. By applying this principle, we can systematically account for the overlaps between different sets of constraints, thus avoiding the need to explicitly enumerate all possible worlds. This significantly reduces the computational burden and improves the efficiency of probability calculations. Our preliminary experiment involves testing the prototype implementation on a set of example traces and probabilistic Declare models also belonging to classical declarative process mining benchmarks from the literature. The experiment compares the performance and accuracy of the prototype with and without the inclusion-exclusion principle. The results indicate that the inclusion-exclusion principle greatly enhances computational efficiency.

2. Preliminaries

2.1. Process Mining

A *process* is a sequence of interrelated activities performed to achieve specific goals. In stark contrast to projects, processes are recurrent and have well-defined inputs and outputs. Due to the fact that processes are repeated over time, understanding, optimizing and improving processes is an appealing problem of practical interest, as each future enactment of the process benefits of these improvements. In the case of a manufacturing process, this could mean a faster execution speed, requiring less resources, or being overall cheaper.

Process Mining [6, 7] emerges as an interdisciplinary research field, encompassing tools and techniques from computer science, formal methods, data science and business process management to study processes. The central objects of Process Mining techniques are the *event log* and the *process model*.

Event logs consist of all the activities that have been performed in order to execute a process, and can be naturally partitioned into *traces*, that is, group together all events produced by the same execution of the process. Process models, on the other hand, are mathematical abstractions that allow to study processes and their properties. Process models can be either described by a domain expert, or learned from event logs, which is referred to as performing the *process discovery* [8] task. Another classic task is the *conformance checking* [9] task, which amounts to compute whether a trace is compliant to a given process model or not. There exist different formalisms to express process models. In particular, there's a distinction between declarative process models and procedural (or imperative) process models [10]. Procedural models aim to explicitly describe compliant traces. The most common process models in Process Mining literature are variants of the Petri Net, an automaton-like computational structure which is procedural in nature. Declarative process models, on the other hand, consist usually of high-level specifications that *shrink* the space of valid traces, by forbidding invalid behavior. It is well-known in the process mining literature that, in practice, imperative process models and declarative process models are effective in different contexts [11]: imperative process easily model well-structured processes while declarative process models are more apt for loosely structured processes [12]. Among declarative process modeling languages, Declare [3] has been the most successful one. The Declare language consists of a set of templates based on a set of *temporal patterns* which frequently arise when writing specifications [13]. Each Declare template describes a temporal relationship between some activities occurring in the same process trace — and, implicitly, defining *what should not happen* in a given trace for it to be considered compliant to the template. Thus, a process model in Declare is a set of constraints, obtained by instantiating some templates with specific activities. This has enabled process mining practitioners to exploit formal techniques grounded in LTL_f to provide specifications for

processes, without the hassle and intricacies [14] of writing a complete LTL_f specification from scratch. Furthermore, it has also provided effective techniques to tackle Declare reasoning problems. Declare-based reasoning tasks are usually dealt either with ad-hoc procedures scanning each trace [15], regular expressions [12], or by reasoning on their corresponding automaton (as in the case of monitoring [16]), which is obtained from their corresponding LTL_f definition through established techniques [17].

Early approaches to Declare and Declare-like patterns are based on logic programming [18, 19, 20, 21]. Recent works also apply Answer Set Programming to tackle conformance checking and query checking of Declare models [4].

2.2. Linear Temporal Logic over Finites Traces

Linear Temporal Logic (LTL; [22]) is an extension of propositional logic where the addition of *temporal operators* enables to reason over an infinite sequence of *states* (finite sets of propositional symbols), called *trace*. Linear Temporal Logic over Finite Traces (LTL_f; [23]) is a finite variant of LTL, where traces are finite sequences. Let $\mathcal{A}$ be a finite set of propositional symbols. An LTL_f formula φ over $\mathcal{A}$ is defined according to the following grammar:

$$\varphi := \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \neg\varphi \mid a \mid \mathsf{F}\,\varphi \mid \mathsf{G}\,\varphi$$

where $a \in \mathcal{A}$. A trace is a finite sequence $\pi = \sigma_0 \dots \sigma_k$, where $\sigma_i \subseteq \mathcal{A}$ is the i-th state of π, denoted by $\pi(i)$ and $k+1$ is the trace length, denoted by $len(\pi)$. Given a trace π, an integer $0 \leq i < len(\pi)$ and formula φ, the satisfaction relation is denoted by $\pi, i \models \varphi$ (to be read "π satisfies φ at time i") and defined as follows:

- $\pi, i \models \top$ if $0 \leq i < len(\pi)$;
- $\pi, i \models a \in \mathcal{A}$ if $a \in \pi(i)$;
- $\pi, i \models \mathsf{X}\,\psi$ if $\pi, i+1 \models \psi$ and $i+1 < len(\pi)$;
- $\pi, i \models \phi \wedge \psi$ if $\pi, i \models \phi$ and $\pi, i \models \psi$;
- $\pi, i \models \neg\phi$ if it is not true that $\pi, i \models \phi$;
- $\pi, i \models \phi\,\mathsf{U}\,\psi$ if there exists $i \leq j < len(\pi)$ such that $\pi, j \models \psi$ and for all $i \leq k < j$ it holds that $\pi, k \models \phi$;
- $\pi, i \models \mathsf{F}\,\varphi$ if there exists $i \leq j \leq len(\pi) - 1$ such that $\pi, j \models \varphi$;
- $\pi, i \models \mathsf{G}\,\varphi$ if for all $i \leq j \leq len(\pi) - 1$ we have that $\pi, j \models \varphi$.

If $\pi, i \models \neg\phi$, we also write $\pi, i \not\models \phi$. Whenever $\pi, 0 \models \varphi$, we say that π is a *model* of φ. Classic propositional shorthands (e.g. $\rightarrow, \vee, \dots$) are defined as usual.

Example 1. *Consider the following formulae ϕ_1, ϕ_2, ϕ_3, ϕ_4, along with an informal explanation of their intended semantics:*

$$\phi_1 = (\neg a\ \mathsf{U}\ b) \vee \mathsf{G}\,\neg a \qquad \phi_2 = \mathsf{G}\,(b \rightarrow \mathsf{X}\,c)$$
$$\phi_3 = \mathsf{F}\,c \rightarrow \mathsf{F}\,a \qquad \phi_4 = c$$

ϕ_1 states that either a never happens ($\mathsf{G}\,\neg a$) or that it never happens before the first occurrence of b ($\neg a\ \mathsf{U}\ b$). ϕ_2 states that whenever b occurs, c will occur in the next state. ϕ_3 states that whenever c happens, a happens as well — without any constraint about their relative order in the trace. ϕ_4 states that c must occur in the first state of the trace. An example of model for $\phi_1 \wedge \phi_2 \wedge \phi_3$ would be the trace $\pi = \{b, c\} \cdot \{c, a\}$, while an example of violating trace would be $\pi' = \{c\}$, as c occurs but a does not (thus violating ϕ_3) or $\pi'' = \{b\} \cdot \{c, a\}$ (which violates ϕ_4).

2.3. Declare

LTL_f has been widely applied to domains where reasoning about finite executions *is more natural* [23] than the infinite traces counterpart of LTL. However, despite LTL_f being an expressive logic to write temporal specifications, it is seldom the case that LTL_f is *directly* applied, especially by people without a background in formal methods, due to intricacies and pitfalls [14] of writing correct declarative specifications. In particular, a template language known as Declare [3] has become particularly relevant in the *process mining* [7, 24] community. Although Declare has not originally been developed using LTL_f logic, a very successful stream of research is based on Declare-as-LTL_f formalization [25]. The success of Declare as process modeling language is due to the fact that, although it can be formally grounded in terms of LTL_f, each *template* that is part of the language has an intuitive meaning, that can be understood by non-experts in temporal logic. This enables process mining practitioners and domain experts to compose specifications as set of small and easily-understandable LTL_f formulae, without being logic experts, while getting all the benefits of the LTL_f formalization of Declare.

The Declare language is composed by a set of *templates*. Each template corresponds to an LTL_f definition, and can be "instantiated" into a *constraint* (an LTL_f formula) by specifying which activities the template relates to.

Example 2. *The formulae ϕ_1, ϕ_2, ϕ_3, ϕ_4 are respectively instances of the Precedence, Chain Response, Responded Existence and Init Declare templates, $\phi_1 \equiv$ Precedence(a, b), $\phi_2 \equiv$ Chain Response(b, c), $\phi_3 \equiv$ Responded Existence(a, c), and $\phi_4 \equiv$ Init(c).*

3. Probabilistic Declarative Specifications

One of the limits of declarative process specifications, not specifically related to Declare or LTL_f, is that a given process execution trace is either *compliant* or not to a given specification. Since typically declarative specifications are understood as a conjunction of constraints, this means that a single violated constraints causes the whole model to be violated. This does not allow to reason, handle and address *uncertainty*, which is intrinsic in process executions. Recently, *probabilistic extensions* of Declare and LTL_f have been proposed [26, 27, 28]. One limitation of these approaches it that the semantic of these probabilistic extensions interprets the *probability* associated to each constraint as its *expected support* on an event log. Recall that the support of a constraint over a log is defined as the percentage of traces satisfying the constraint contained in the log. This has counterintuitive side-effects, which are discussed in more detail in [2]. Most importantly, it is not possible to measure *how much* a *single trace* is compliant w.r.t. a specification. Authors of [2] tackle this issue by introducing the notion of *probabilistic declarative specifications*, which encompasses a probabilistic extension of Declare, whose semantics is inspired to the popular *distribution semantics* in probabilistic logic programming [29, 30]. Most importantly, [2] also introduces a notion of *compliance* of a trace w.r.t. a probabilistic declarative specification, that quantitatively measures the probability of a trace satisfying a declarative specification without referring to an event log.

The main contribution of this paper is to revise, refine and build upon ideas introduced in [2] to provide a definition of probabilistic declarative specification and trace compliance which is grounded in standard notions of discrete probability spaces. This yields an equivalent characterization of probabilistic declarative specifications, aligned with standard notions of probability, that results in a more efficient implementation of the compliance computation. We briefly recap definitions and concepts introduced in [2].

Traces & logs. Let $\mathcal{A}$ be a finite set of *activities*. A *process execution trace* (or just trace) is a string over $\mathcal{A}$. (In applications of LTL_f to declarative process mining, it is customary to assume that for each state $|\pi| = 1$, e.g. exactly one event occurs at each time-step. Hence, traces can be modeled as strings over $\mathcal{A}$ rather than over $2^{\mathcal{A}}$.) An *event log* is a multi-set of traces. Although declarative specifications are usually expressed in LTL_f, characterization herein introduced are more general than LTL_f. Thus, we rely on an abstract notion of *constraint* (similar to [31]).

Constraints & satisfaction. A *constraint* c is a function with signature $c : \mathcal{A}^* \mapsto \{\top, \bot\}$. Given a trace π, a constraint is *satisfied* if $c(\pi) = \top$ (also denoted by $\pi \models c$) and *violated* if $c(\pi) = \bot$ (also denoted by $\pi \not\models c$). We respectively say that the trace *satisfies* or *violates* the constraint.

Declarative specifications (DS). A DS (or simply *specification*) is a set of constraints. A trace π *violates* a specification $\mathcal{M} = \{c_1, \ldots, c_k\}$ if there exists $1 \leq i \leq k$ such that $\pi \not\models c_i$; it *complies* with a specification $\mathcal{M}$ if it does not violate it (i.e. if $\pi \models c_i$ for all $1 \leq i \leq k$). With a slight abuse of notation, we denote these scenarios by $\pi \not\models \mathcal{M}$ and $\pi \models \mathcal{M}$ respectively. Declarative specifications satisfy the following *monotonicity* property: If π violates a constraint c, then π violates any specification that includes c.

Probabilistic declarative specifications (PDS). The idea of PDS, as introduced in [2], is to associate to each constraint in a specification $\mathcal{M}$ a probability, to be understood as the *strength* (or *relevance*) of the constraint. The underlying idea, inspired from distribution semantics, is that a probabilistic specification defines a probability distribution over standard (non-probabilistic) declarative specifications, where constraints are independently sampled from each other. Constraints with probability 1 (or 0) are *certain*, *mandatory*, *crisp* and should always be satisfied (or ignored) in order for a trace to be compliant with the specification. On the other hand, true probabilistic constraints ($0 < p < 1$) are not *mandatory*, and a trace can violate them while still overrall satisfying the declarative specification. Thus, rather than *satisfying traces* and *violating traces*, authors of [2] propose a quantitative notion of *compliance*, the probability that a trace is accepted by a declarative specification sampled from a probabilistic declarative specification. The intution is that if a constraint gets associated to a *high* probability, being compliant to that constraint yields *higher* compliance to the overall specification.

Probability space. We ground the notion of PDS and compliance in terms of a discrete probability space over subsets of a declarative specification. The following provides an alternative characterization of PDSs as introduced in Definitions 5-10 of the original article [2].

Lemma 1. *Let U be a finite, nonempty set, $\sigma : U \mapsto [0, 1]$. Let $f : 2^U \mapsto [0, 1]$ be the function:*

$$f(S) = \prod_{x \in S} \sigma(x) \cdot \prod_{y \in U \setminus S} 1 - \sigma(y)$$

It holds that $\sum_{S \in 2^U} f(S) = 1$.

Proof. By induction on $n = |U|$. If $U = \{x_1\}$, then it has two subsets $\emptyset$ and U, with respectively $f(\emptyset) = 1 - \sigma(x_1)$ and $f(U) = \sigma(x_1)$, thus the property holds. Assume the property holds for $U = \{x_1, \ldots, x_n\}$. We need to prove it holds for $U' = U \cup \{x_{n+1}\}$.

$$\begin{aligned}
\sum_{S \in 2^{U'}} f(S) &= \sum_{\substack{S \in 2^{U'} : \\ x_{n+1} \in S}} f(S) + \sum_{\substack{S \in 2^{U'} : \\ x_{n+1} \notin S}} f(S) && (1)\\
&= \sum_{S \in 2^U} f(S \cup \{x_{n+1}\}) + f(S) && (2)\\
&= \sum_{S \in 2^U} \sigma(x_{n+1}) f(S) + (1 - \sigma(x_{n+1})) \cdot f(S) && (3)\\
&= \sum_{S \in 2^U} (\sigma(x_{n+1}) + 1 - \sigma(x_{n+1})) \cdot f(S) && (4)\\
&= \sum_{S \in 2^U} f(S) = 1 && (5)
\end{aligned}$$

□

Thus, f is a probability mass function, and the pair $(2^U, f)$ is a discrete probability space.

Definition 1 (Probabilistic Declarative Specification; PDS)**.** *A probabilistic declarative specification (PDS) is a pair* $(\mathcal{M}, \sigma)$ *where* $\mathcal{M}$ *is a set of constraints and* $\sigma : \mathcal{M} \mapsto [0, 1]$*. We call* $\sigma(c)$ *the relevance of the constraint* $c \in \mathcal{M}$*.*

Let $\mathcal{W} = 2^{\mathcal{M}}$ be the set of *worlds* associated to a PDS. By the previous lemma, $(\mathcal{W}, f)$ is a discrete probability space. With a slight abuse of notation, we will refer to the PDS and the underlying probability space as PDS. Intuitively, the outcome of the stochastic event is a standard declarative specification.

Let $C(c)$ be the event "the constraint c belongs to the declarative specification". We show that $\mathbb{P}(C(c)) = \sigma(c)$, where $\mathbb{P}(\cdot)$ denotes the probability function.

Lemma 2. *Let* $(\mathcal{M}, \sigma)$ *be a PDS. Let* $c \in \mathcal{M}$*. It holds that* $\mathbb{P}(C(c)) = \sigma(c)$*.*

Proof. A subset of $\mathcal{M}$ either contains c or not. Thus, $2^{\mathcal{M}} = \bigcup_{S \subseteq \mathcal{M} \setminus \{c\}} \{S, S \cup \{c\}\}$. We are interested in worlds that contain c, i.e. W of the form $\{c\} \cup S$, where $S \subseteq \mathcal{M} \setminus \{c\}$.

$$\mathbb{P}(W) = \sigma(c) \cdot \prod_{x \in S} \sigma(x) \cdot \prod_{y \in \mathcal{M} \setminus (S \cup \{c\})} 1 - \sigma(y)$$

By summing up over such worlds W, we obtain $\sigma(c)$, hence $\mathbb{P}(C(c)) = \sigma(c)$. □

Similar reasoning yields that $C(c)$ and $C(c')$, for $c \neq c'$, are independent events. This is not surprising: we can recognize f as the joint probability mass function of n independent Bernoulli variables. These facts, that follow from the underlying discrete probability space structure of $(\mathcal{M}, \sigma)$, indeed reconcile our definition of PDS with the one provided in [2]. Thus with a slight abuse of notation we can also talk about *probability* of a constraint c, although the probability space is defined over *worlds*.

Let π be a trace, $(\mathcal{M}, \sigma)$ a PDS. For each specification $W \subseteq \mathcal{M}$, either $\pi \models W$ or $\pi \not\models W$; thus π effectively behaves like a binary random variable $\Pi : 2^{\mathcal{M}} \mapsto \{0, 1\}$, where $\Pi(W) = 1$ if and only if $\pi \models W$. With a slight abuse of notation, we will use π both to denote the string over $\mathcal{A}$ and its associated random variable Π.

Compliance [2] is understood as a measure for "how likely it is for π to satisfy $\mathcal{M}$". It is defined in [2] as the sum of probabilities of worlds W such that $\pi \models W$. By treating traces as binary random variables, compliance naturally emerges as the expected value of the random variable π:

$$\begin{aligned} \mathsf{compliance}((\mathcal{M}, \sigma), \pi) = \mathbb{E}(\pi) &= 0 \cdot \mathbb{P}(\pi = 0) + 1 \cdot \mathbb{P}(\pi = 1) && (6) \\ &= \mathbb{P}\Big(\bigcup_{\substack{W \subseteq \mathcal{M}: \\ \pi \models W}} W\Big) && (7) \\ &= 1 - \mathbb{P}\Big(\bigcup_{\substack{W \subseteq \mathcal{M}: \\ \pi \not\models W}} W\Big) && (8) \end{aligned}$$

To compute $\mathsf{compliance}(\mathcal{M}, \pi)$, we need to reason about composite events. First, we establish that:

1 If $\sigma(c) = 1$ and $\pi \not\models c$, then the probability of a satisfying world for π is zero;
2 If $\sigma(c) = 1$ and $\pi \models c$, then c does not affect the overall compliance, $\mathsf{compliance}(\mathcal{M}, \pi) = \mathsf{compliance}(\mathcal{M} \setminus \{c\}, \pi)$;

These facts easily follow from the probability associated to each world. We provide a numeric example of this fact, which will serve as a running example through the rest of the section. To ease computations, in the rest of the sections whenever z is a real number in $[0, 1]$ we denote by z' the value $z' = 1 - z$.

Example 3 (Compliance is unaffected by crisp constraints)**.** *Consider the specification* $\mathcal{M} = \{\phi_1, \phi_2, \phi_3, \phi_4\}$ *of the previous example, with* $\sigma(\phi_1) = 1$*,* $\sigma(\phi_2) = p$*,* $\sigma(\phi_3) = q$*,* $\sigma(\phi_4) = k$*. This yields 16 possible worlds* $W_1, \ldots, W_{16}$*, with corresponding probability* $\mathbb{P}(W_i)$*:*

$$
\begin{aligned}
W_1 &= \{\} & \mathbb{P}(W_1) &= 0 \cdot p' \cdot q' \cdot k' & &= 0\\
W_2 &= \{\phi_1\} & \mathbb{P}(W_2) &= 1 \cdot p' \cdot q' \cdot k' & &= p'q'k'\\
W_3 &= \{\phi_2\} & \mathbb{P}(W_3) &= 0 \cdot p \cdot q' \cdot k' & &= 0\\
W_4 &= \{\phi_3\} & \mathbb{P}(W_4) &= 0 \cdot p' \cdot q \cdot k' & &= 0\\
W_5 &= \{\phi_4\} & \mathbb{P}(W_5) &= 0 \cdot p' \cdot q' \cdot k & &= 0\\
W_6 &= \{\phi_1, \phi_2\} & \mathbb{P}(W_6) &= 1 \cdot p \cdot q' \cdot k' & &= pq'k'\\
W_7 &= \{\phi_1, \phi_3\} & \mathbb{P}(W_7) &= 1 \cdot p' \cdot q \cdot k' & &= p'qk'\\
W_8 &= \{\phi_1, \phi_4\} & \mathbb{P}(W_8) &= 1 \cdot p' \cdot q' \cdot k & &= p'q'k\\
W_9 &= \{\phi_2, \phi_3\} & \mathbb{P}(W_9) &= 0 \cdot p \cdot q \cdot k' & &= 0\\
W_{10} &= \{\phi_2, \phi_4\} & \mathbb{P}(W_{10}) &= 0 \cdot p \cdot q' \cdot k & &= 0\\
W_{11} &= \{\phi_3, \phi_4\} & \mathbb{P}(W_{11}) &= 0 \cdot p' \cdot q \cdot k & &= 0\\
W_{12} &= \{\phi_2, \phi_3, \phi_4\} & \mathbb{P}(W_{12}) &= 0 \cdot p \cdot q \cdot k & &= 0\\
W_{13} &= \{\phi_1, \phi_2, \phi_3\} & \mathbb{P}(W_{13}) &= 1 \cdot p \cdot q \cdot k' & &= pqk'\\
W_{14} &= \{\phi_1, \phi_2, \phi_4\} & \mathbb{P}(W_{14}) &= 1 \cdot p \cdot q' \cdot k & &= pq'k\\
W_{15} &= \{\phi_1, \phi_3, \phi_4\} & \mathbb{P}(W_{15}) &= 1 \cdot p' \cdot q \cdot k & &= p'qk\\
W_{16} &= \{\phi_1, \phi_2, \phi_3, \phi_4\} & \mathbb{P}(W_{16}) &= 1 \cdot p \cdot q \cdot k & &= pqk
\end{aligned}
$$

The only worlds with a non-null probability are W_i, $i \in \{2, 6, 7, 8, 13, 14, 15, 16\}$, and we can easily observe the bijection between this set of worlds and the power set of $\{\phi_2, \phi_3, \phi_4\}$ (e.g., the specification obtained by discarding crisp constraints):

$$
\begin{aligned}
W'_1 &= \{\} & \mathbb{P}(W'_1) &= p'q'k' & &= \mathbb{P}(W_2)\\
W'_2 &= \{\phi_2\} & \mathbb{P}(W'_2) &= pq'k' & &= \mathbb{P}(W_6)\\
W'_3 &= \{\phi_3\} & \mathbb{P}(W'_3) &= p'qk' & &= \mathbb{P}(W_7)\\
W'_4 &= \{\phi_4\} & \mathbb{P}(W'_4) &= p'q'k & &= \mathbb{P}(W_8)\\
W'_5 &= \{\phi_2, \phi_3\} & \mathbb{P}(W'_5) &= pqk' & &= \mathbb{P}(W_{13})\\
W'_6 &= \{\phi_2, \phi_4\} & \mathbb{P}(W'_6) &= pq'k & &= \mathbb{P}(W_{14})\\
W'_7 &= \{\phi_3, \phi_4\} & \mathbb{P}(W'_7) &= p'qk & &= \mathbb{P}(W_{15})\\
W'_8 &= \{\phi_2, \phi_3, \phi_4\} & \mathbb{P}(W'_8) &= pqk & &= \mathbb{P}(W_{16})
\end{aligned}
$$

Crisp constraints with a null probability have a similar behavior.

Example 4 (Compliance computation - Enumerating Worlds)**.** *Consider a trace $\pi = \{b, c, a\}$. It satisfies the constraints ϕ_1, ϕ_3 and ϕ_4. Thus, compatible worlds are all the ones that do not contain ϕ_2, namely W_i, with $i \in I = \{1, 2, 4, 5, 7, 8, 11, 15\}$. Computing $\sum_{i \in I} \mathbb{P}(W_i)$ according to worlds' probabilities in Example 3 yields $p'q'k' + p'qk' + p'q'k + p'qk = p'(q'k' + qk' + q'k + qk) = p'$ (ignoring null terms).*

Thus, since we have shown that compliance of a trace π is not affected by crisp constraints in $\mathcal{M}$, in the rest of the section we assume without loss of generality that $(\mathcal{M}, \sigma)$ consists solely of probabilistic constraints (e.g., for all c, $0 < \sigma(c) < 1$). Furthermore, without loss of generality, we assume that constraints in $\mathcal{M}$ are indexed in such a way that $\pi \not\models c_i$ if $1 \leq i \leq k$ and $\pi \models c_i$ if $k < i \leq n$, for some $1 \leq k \leq |\mathcal{M}|$. We define the composite events $V(i) = C(c_i)$, worlds that contain the violating constraint c_i; $\mathsf{VIO} = \bigcup_{1 \leq i \leq k} V(i)$ the event that corresponds to worlds violating at least one constraint, thus yielding worlds for which π is not compliant. Similarly, we define $S(i)$ as the worlds that contain the satisfied constraint c_i *and none of the violating constraints*, thus $S(i) = C(i) \cap \overline{\mathsf{VIO}}$, with $\mathsf{SAT} = \bigcup_{k+1 \leq i \leq n} S(i) = \bigcup_{k+1 \leq i \leq n} C(i) \cap \overline{VIO}$. Furthermore, the $null$ event is the event that includes no satisficing constraints.

Example 5. *Consider the PDS* $(\{\phi_1, \phi_2, \phi_3, \phi_4\}, \sigma)$ *and the trace* $\pi = \{b, c, a\}$ *as by Example 4. First, we would ignore the crisp constraint* ϕ_1*, and re-index* $\mathcal{M}$ *as above, obtaining* $\mathcal{M}' = \{c_1, c_2, c_3\}$ *where* $c_1 = \phi_2$*,* $c_2 = \phi_3$ *and* $c_3 = \phi_4$*. In this case, we have* $k = 1$*. The event* $V(1)$ *is the set of worlds that include* ϕ_2*. Referring to Example 3, that is worlds* $\{W_3, W_6, W_9, W_{10}, W_{12}, W_{13}, W_{14}, W_{16}\}$ *. The event* $S(2)$ *are the set of worlds that do not contain* ϕ_2 *and include* ϕ_3*, namely* $\{W_4, W_7, W_{11}, W_{15}\}$ *and* $S(3)$ *by analogous reasoning is the set of worlds that do not contain* ϕ_2 *and include* ϕ_4*, namely* $\{W_5, W_8, W_{11}, W_{15}\}$*.* $S(2)$ *and* $S(3)$ *are not disjoint events since they both include the worlds* W_{11} *and* W_{15}*.*

Since VIO and SAT are defined as the union of non-disjoint events, their probabilities can be computed by applying the inclusion-exclusion principle [5]. In the rest of the section, to simplify the notation, we denote $C(c_i)$ by $C(i)$. In particular:

$$\mathbb{P}(\mathsf{VIO}) = \mathbb{P}(\bigcup_{1\leq i\leq k} V(i)) \tag{9}$$

$$\mathbb{P}(\mathsf{SAT}) = \mathbb{P}(\bigcup_{k+1\leq i\leq n} S(i)) \tag{10}$$

$$= \mathbb{P}(\bigcup_{k+1\leq i\leq n} C(i) \cap \overline{\mathsf{VIO}}) \tag{11}$$

$$= \mathbb{P}(\bigcup_{k+1\leq i\leq n} C(i) \cap \overline{(\bigcup_{1\leq i\leq k} V(i))}) \tag{12}$$

$$= \mathbb{P}(\bigcup_{k+1\leq i\leq n} C(i) \cap (\bigcap_{1\leq i\leq k} \overline{V(i)})) \tag{13}$$

$$= \mathbb{P}(\bigcap_{1\leq i\leq k} \overline{V(i)}) \cdot \mathbb{P}(\bigcup_{k+1\leq i\leq n} C(i)) \tag{14}$$

$$= (\prod_{1\leq i\leq k} 1 - \mathbb{P}(V(i))) \cdot \mathbb{P}(\bigcup_{k+1\leq i\leq n} C(i)) \tag{15}$$

The definition of VIO and SAT events rely on the monotonicity assumption of the semantics underlying the constraints in the specification $\mathcal{M}$. Thus, we can compute compliance by applying the inclusion-exclusion principle [5] either to the collections $\{C(i) : k+1 \leq i \leq n\}$ or $\{C(i) : 1 \leq i \leq k\}$:

$$\mathbb{P}\left(\bigcup_{i=1}^{n} C(i)\right) = \sum_{k=1}^{n} \left((-1)^{k-1} \sum_{\substack{I\subseteq\{1,\ldots,n\} \\ |I|=k}} \mathbb{P}\left(\bigcap_{j\in I} C(j)\right) \right) \tag{16}$$

$$\mathbb{P}\left(\bigcup_{i=1}^{n} C(i)\right) = \sum_{k=1}^{n} \left((-1)^{k-1} \sum_{\substack{I\subseteq\{1,\ldots,n\} \\ |I|=k}} \prod_{j\in I} \mathbb{P}(C(j)) \right) \tag{17}$$

$$\mathbb{P}\left(\bigcup_{i=1}^{n} C(i)\right) = \sum_{k=1}^{n} \left((-1)^{k-1} \sum_{\substack{I\subseteq\{1,\ldots,n\} \\ |I|=k}} \prod_{j\in I} \sigma(j) \right) \tag{18}$$

Computing such probabilities boils down to the probability of independent events, which are input values provided in σ for the PDS $(\mathcal{M}, \sigma)$ (as it follows from Lemma 2). We can choose either one of the sets, eventually complementing the result. In practice, due to the exponential number of subsets to be considered, we prefer the one with the least cardinality (e.g., compute based on $\{C(i) : 1 \leq i \leq k\}$ if π has less violated constraints in $\mathcal{M}$ than satisficing ones; viceversa, $\{C(i) : k+1 \leq i \leq n\}$ if there are less satisficing constraints than violating ones).

Algorithm 1 Computing compliance

1: **procedure** ComputeCompliance(π, $(\mathcal{M}, \sigma)$)
2: **Arguments:**
3: π: An execution trace
4: $(\mathcal{M}, \sigma)$: A PDS over set of constraints $\mathcal{M}$, where $\sigma(c)$ is the probability of $c \in \mathcal{M}$
5: **Return value:**
6: Compliance of π wrt $(\mathcal{M}, \sigma)$
7:
8: VIOLATED $\leftarrow \{c \in \mathcal{M} : \pi \not\models c\}$
9: **if** $\exists c : c \in$ VIOLATED **and** $\sigma(c) = 1$ **then**
10: **return** 0
11: **end if**
12: SATISFIED $\leftarrow \{c \in \mathcal{M} \setminus \text{VIOLATED} : \sigma(c) < 1\}$
13: **if** |VIOLATED| $\leq$ |SATISFIED| **then**
14: **return** $1-$ InclusionExclusion(VIO, σ)
15: **else**
16: $p_{\text{no\_vio}} \leftarrow \prod_{c \in \text{VIOLATED}} 1 - \sigma(c)$
17: $p_{\text{no\_sat}} \leftarrow \prod_{c \in \text{SATISFIED}} 1 - \sigma(c)$
18: **return** $p_{\text{no\_vio}} \cdot (\text{InclusionExclusion}(\text{SATISFIED}, \sigma) + p_{\text{no\_sat}})$
19: **end if**
20: **end procedure**

Algorithm 2 Applying inclusion-exclusion principle

1: **procedure** InclusionExclusion($\mathcal{C}$, σ)
2: **Arguments:**
3: $\mathcal{C}$: A set of constraints
4: σ: A function that maps each constraint to its probability
5: **Return value:**
6: p: Probability of the set $\mathcal{C}$
7:
8: $p \leftarrow 0$
9: **for** $k \leftarrow 1$ to $|\mathcal{C}|$ **do**
10: **for each** E **in** $\{S \in 2^{\mathcal{C}} : |S| = k\}$ **do**
11: $p_E \leftarrow \prod_{c \in E} \sigma(c)$
12: **if** k is odd **then**
13: $p \leftarrow p + p_E$
14: **else**
15: $p \leftarrow p - p_E$
16: **end if**
17: **end for**
18: **end for**
19: **return** p
20: **end procedure**

This suggests a straightforward 2-phase technique to compute compliance of a trace wrt a PDS, shown as Algorithm 1, where InclusionExclusion is a procedure (reported as Algorithm 2) that iterates over subsets of a finite set and computes probabilities according to the above considerations. Specifically, the first phase comprises only line 8 of Algorithm 1 and uses an external tool (as for example the one presented in [4]) to identify all the violated constraints and store them in the set VIOLATED. The external tool is not further used afterward, in the second phase of the proposed procedure, which starts

by checking the presence of crisp constraints in VIOLATED (lines 9–10). If all the violated constraints are probabilistic, then the set SATISFIED of satisfied probabilistic constraints is computed (line 12) and inclusion-exclusion is applied on the smaller among the two sets of constraints (lines 13–19). The quantities $p_{\text{no\_vio}}$ and $p_{\text{no\_sat}}$ (lines-16–17) denote respectively the probability of a world not containing any violated constraint, and the probability of a world not containing any *satisfied* constraint — recall that the empty specification trivially accepts all traces.

Example 6 (Computing compliance - Inclusion Exclusion)**.** *In the same setting as Example 4, the satisficing non-crisp constraints are ϕ_3 and ϕ_4. Since we have a single violated constraint and two satisfied non-crisp constraints, the algorithm would apply the inclusion-exclusion computation over the singleton set of constraint $\{\phi_2\}$, yielding $\sigma(\phi_2) = p$, and return its complement $1 - p = p'$, matching results obtained by enumrating worlds in Example 4. Similarly, applying the algorithm on the set of satisficing constraints $\{\phi_3, \phi_4\}$ would yield the following quantities:*

$$p_{no\_vio} = (1 - \sigma(\phi_2)) = p'$$
$$p_{no\_sat} = (1 - \sigma(\phi_3))(1 - \sigma(\phi_4)) = q'k'$$

applying the inclusion-exclusion principle over the set $\{\phi_3, \phi_4\}$ yields the probability:

$$P = \sigma(\phi_3) + \sigma(\phi_4) - \sigma(\phi_3)\sigma(\phi_4) = q + k - qk$$

Hence, returning the value:

$$\begin{aligned} p_{no\_vio}(p_{no\_sat} + P) &= p'(q'k' + q + k - qk) \\ &= (1 - p)((1 - q)(1 - k) + q + k - qk) \\ &= (1 - p)((1 - k - q + qk + q + k - qk)) \qquad = (1 - p) \end{aligned}$$

Also in this case, this matches the result obtained by worlds enumeration.

4. Experiments

We perform some experiments to assess scalability and feasibility of our approach. Code for our prototype and to reproduce these experiments is available in a public repository[1].

4.1. Inclusion-exclusion scalability

By applying the inclusion-exclusion principle, according to the previous section, computing compliance of a trace wrt a PDS is exponential in the number $\min(|\mathsf{VIO}|, |\mathsf{SAT}|)$. Hereinafter, the number $\min(|\mathsf{VIO}|, |\mathsf{SAT}|)$ is referred to as the *number of effective constraints*. Computing compliance does not depend altogether on the nature and underlying semantics of the constraints to be checked against the trace, nor on the length of the trace. Similarly, the size (number of constraints) in the PDS does not affect complexity, but merely provides an upper bound on the number of effective constraints (in the worst case, the number of effective constraints is half of the number of probabilistic constraints).

Thus, to assess scalability of this approach, we measure time to compute compliance as the number of effective constraints increases. In particular, we analyze runtimes of compliance checking varying the number of effective constraints from 1 to 30, sampling 5 PDSs for each number, and 10 traces for each PDS. This yields 50 compliance measurements for each number of effective constraints. Table 1 reports the average runtime and the standard deviation for number of constraints handled within the timeout of 120s (i.e., from 21 to 27). We can observe that runtime doubles with the increment of the number of effective constraints, which is expected due to inclusion-exclusion being applied. However,

[1] https://www.github.com/ainnoot/pds-compliance

# of Effective Constraints	Average Runtime	Standard Deviation
21	1.39149	0.03111
22	2.93617	0.06001
23	6.01178	0.13515
24	12.97994	1.57132
25	25.84984	0.62080
26	54.67553	1.20554
27	109.37013	5.09600

Table 1
Average runtimes and standard deviation for compliance checking over a PDS with a given number of effective constraints. All measurements are in seconds. We omit statistics with less than 20 effective constraints as they result in less than 0.5 seconds of runtime.

it is important to note that the number of effective constraints is, in the worst case, half of the number of probabilistic constraints. It turns out that an enumeration of possible worlds has to deal with 2^n subsets of constraints, while our approach only consider $2^{\frac{n}{2}}$ subsets in the worst case. [2] We leave the comparison with the original implementation provided in [2] as a future work, but we observe here that [2] reports times in the order of 10^4 seconds of computation to process models with 14 crisp constraints and 7 probabilistic constraints, while our implementation can handle up to 54 probabilistic constraints in less than 2 minutes (120 seconds), with crisp constraints not affecting times at all; these times regard only compliance computation, and crisp conformance checking has to be performed separately, with times in the order of minutes.

4.2. Use on a real-world log

We also make some preliminary tests of our approach on real-world logs. As an example, we compute compliance for traces of the *Sepsis event log*, a well-known process mining event log dealing with healthcare data. First, we extract a Declare model $\mathcal{M}$ using MINERful [12] default parameters, interpreting the support of each constraint as its associated probability. Then, we use an ASP-based tool [4] to perform conformance checking of $\mathcal{M}$ over each trace of $\mathcal{L}$. The resulting model $\mathcal{M}$ contains 36 constraints. Upon projecting the input traces onto the constraints (e.g., computing the sets SAT and VIO for each trace), we obtain 28 unique traces. Thus, we are to perform 28 compliance checking tasks on a PDS of at most 18 effective constraints. This is well within the capacity of our approach. Indeed, the whole process takes about 2 seconds on a standard laptop, with roughly 1 second spent parsing input and performing crisp conformance checking, and the rest of time computing compliance of the 28 traces.

5. Conclusion

Probabilistic extensions of LTL_f [26, 27, 28] and LTL_f-based pattern languages are increasingly gaining attraction in process mining applications. A recent work by [2] proposes a novel probabilistic semantic for declarative specifications, inspired by distribution semantic from probabilistic logic programming. This approach disentangles the probability associated to a constraint in the declarative specification from the notion of *expected relative frequency* observed in an event log, which is the approach of other probabilistic extensions of LTL_f and Declare. In this paper, we propose an alternative approach to address compliance computation, based on standard discrete probability notions and combinatorial principles, which has the potential to significantly improve the performance of the logic programming-based approach introduced in [2].

[2] Intuitively, if $\frac{n}{2}$ constraints are satisfied (thus, $\frac{n}{2}$ violated) we are forced to apply inclusion-exclusion over $\frac{n}{2}$ items.

Acknowledgments

This work was partially supported by Italian Ministry of University and Research (MUR) under PRIN project PRODE "Probabilistic declarative process mining", CUP H53D23003420006, under PRIN project PINPOINT "exPlaInable kNowledge-aware PrOcess INTelligence", CUP H23C22000280006, under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by Italian Ministry of Health (MSAL) under POS projects CAL. HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); and by the LAIA lab (part of the SILA labs). Mario Alviano and Francesco Ricca are members of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[1] P. Trkman, The critical success factors of business process management, Int. J. Inf. Manag. 30 (2010) 125–134. URL: https://doi.org/10.1016/j.ijinfomgt.2009.07.003.

[2] M. Vespa, E. Bellodi, F. Chesani, D. Loreti, P. Mello, E. Lamma, A. Ciampolini, Probabilistic compliance in declarative process mining, in: Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024), volume 3779 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 11–22.

[3] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for loosely-structured processes, in: EDOC, IEEE Computer Society, 2007, pp. 287–300.

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

[5] S. S. Sane, The inclusion-exclusion principle, Hindustan Book Agency, Gurgaon, 2013, pp. 57–79. URL: https://doi.org/10.1007/978-93-86279-55-2_4. doi:10.1007/978-93-86279-55-2_4.

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

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

[8] W. M. P. van der Aalst, Foundations of process discovery, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 37–75. URL: https://doi.org/10.1007/978-3-031-08848-3_2.

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

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

[11] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113. URL: https://doi.org/10.1007/s00450-009-0057-9.

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

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

[14] B. Greenman, S. Saarinen, T. Nelson, S. Krishnamurthi, Little tricky logic: Misconceptions in the understanding of LTL, Art Sci. Eng. Program. 7 (2023).

[15] I. Donadello, F. Riva, F. M. Maggi, A. Shikhizada, Declare4py: A python library for declarative process mining, in: BPM (PhD/Demos), volume 3216 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 117–121.

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

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

Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021, AAAI Press, 2021, pp. 122–130. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15954.
[18] E. Bellodi, F. Riguzzi, E. Lamma, Statistical relational learning for workflow mining, Intell. Data Anal. 20 (2016) 515–541.
[19] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic declarative process mining, in: KSEM, volume 6291 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 292–303.
[20] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Exploiting inductive logic programming techniques for declarative process mining, Trans. Petri Nets Other Model. Concurr. 2 (2009) 278–295.
[21] E. Lamma, P. Mello, F. Riguzzi, S. Storari, Applying inductive logic programming to process mining, in: ILP, volume 4894 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 132–146.
[22] A. Pnueli, The temporal logic of programs, in: FOCS, IEEE Computer Society, 1977, pp. 46–57.
[23] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, IJCAI/AAAI, 2013, pp. 854–860.
[24] W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022.
[25] C. D. Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 108–152.
[26] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Inf. Syst. 109 (2022) 102033.
[27] F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic conformance checking based on declarative process models, in: CAiSE Forum, volume 386 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 86–99.
[28] F. M. Maggi, M. Montali, R. Peñaloza, Temporal logics over finite traces with uncertainty, in: AAAI, AAAI Press, 2020, pp. 10218–10225.
[29] T. Sato, A statistical learning method for logic programs with distribution semantics, in: ICLP, MIT Press, 1995, pp. 715–729.
[30] E. Bellodi, The distribution semantics in probabilistic logic programming and probabilistic description logics: a survey, Intelligenza Artificiale 17 (2023) 143–156.
[31] V. Fionda, A. Ielo, F. Ricca, Logic-based composition of business process models, in: KR, 2023, pp. 272–281.

A. Online Resources

We provide an open-source prototype which implements the techniques herein described, which can be found at the following public repository.