ARTICLE IN PRESS

Information Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Probabilistic declarative process mining

Anti Alman[a], Fabrizio Maria Maggi[b,*], Marco Montali[b,*], Rafael Peñaloza[c]

[a] University of Tartu, Estonia
[b] Free University of Bozen-Bolzano, Italy
[c] University of Milano-Bicocca, Milano, Italy

ARTICLE INFO

Article history:
Received 1 February 2021
Received in revised form 19 December 2021
Accepted 23 March 2022
Available online xxxx
Recommended by Manfred Reichert

Keywords:
Declarative processes
Probabilistic temporal reasoning
Probabilistic process discovery
Probabilistic monitoring
Probabilistic conformance checking

ABSTRACT

In a variety of application domains, (business) processes are intrinsically uncertain. Surprisingly, only very few languages and techniques in BPM consider uncertainty as a first-class citizen. This is also the case in declarative processes, which typically require that process executions satisfy all the elicited process constraints. We counteract this limitation by introducing the notion of probabilistic process constraint. We show how to characterize the semantics of probabilistic process constraints through the interplay of time and probability, and how it is possible to reason over such constraints by loosely coupling temporal and probabilistic reasoning. We then rely on this approach to redefine several key process mining tasks in the light of uncertainty. First, we discuss how probabilistic constraints can be discovered from event data by employing, off-the-shelf, existing algorithms for declarative process discovery. Second, we study how to carry out monitoring, obtaining a setting where a monitored partial trace may be in multiple monitoring states at the same time, though with different probabilities. Third, we handle conformance checking both at the trace and event log level, in the latter case providing a notion of earth mover's distance that suits with our context. All the presented techniques have been implemented in proof-of-concept prototypes.

1. Introduction

Temporal process constraints have been extensively adopted to declaratively capture the acceptable courses of execution in operational processes [1–4]. In particular, the *Declare* constraint-based process modeling language [2,5] has been introduced as a front-end language to specify process constraints based on linear temporal logic over finite traces (LTL$_f$) [6]. While conventional process models requires to explicitly account for all valid courses of executions, constraint-based approaches implicitly characterize them as all the traces that satisfy all modeled constraints. For example, a Declare model may capture an order-to-cash scenario by indicating that whenever an order is accepted, it must be eventually paid by the customer, and that shipment (picking one and only one of the available shipment modalities) can only occur upon a prior payment. The model then separates the (infinitely) many different traces where the order is properly paid and shipped by conforming to all constraints, from the nonconforming traces where at least one such constraints is violated.

In this scenario, as customary in all existing constraint-based approaches for process modeling, the constraints contained in the model are *certain*: they are *all* expected to hold in every conforming execution. This view is too restrictive when one wants to capture commonly recurring patterns, such as:

- *Common behaviors and best practices*, captured as constraints that should hold in the majority, but not necessary all cases. As an example, in the order-to-cash scenario one could express that orders are shipped via truck in at least 90% of the cases.
- *Outlier and exceptional behaviors*, in the form of constraints that hold in a very few, but still conforming, cases. For example, one could express that an order is shipped via car in no more than 1% of the cases;
- *Partially controllable behaviors*, involving activities that are not all controlled by the organization orchestrating the process; in the order-to-cash scenario, payments are executed by external customers, and one could express that it is known that whenever an order is accepted, a payment is performed by the customer in 8 cases out of 10 (which implicitly indicates that in the remaining 2, not payment is issued).

Uncertainty is intrinsically present also when process constraints are discovered from event data, as they may hold only in a

* Corresponding authors.
E-mail addresses: anti.alman@ut.ee (A. Alman), maggi@inf.unibz.it (F.M. Maggi), montali@inf.unibz.it (M. Montali), rafael.penaloza@unimib.it (R. Peñaloza).

https://doi.org/10.1016/j.is.2022.102033
0306-4379/© 2022 Elsevier Ltd. All rights reserved.

(more or less large) share of the traces contained in the log, which in turn contains only a sample of all cases that could be observed in reality. This explains why contemporary Declare discovery techniques support retaining constraints in the discovered model even when such constraints are violated in some cases [7–10]. However, once discovered, such constraints are indistinguishable from constraints that hold in all the traces, leading to potential inconsistencies [11]. Also in this setting, the issue is that uncertainty needs to be explicitly accounted for at the model level.

For all these reasons, it is surprising that only few process mining approaches incorporate uncertainty as a first-class citizen, even in the case where the process is modeled using conventional, imperative formalisms (such as variants of Petri nets) [12]. In the context of Declare or, more generally, of processes declaratively captured in LTL$_f$, no probabilistic, suitable extension existed until recently [13]. In [13], a probabilistic version of LTL$_f$ has been in fact thoroughly studied for the first time. A fragment of that logic, isolated and studied in the same paper, turned out to provide a natural basis for lifting Declare to an uncertain setting, leading to the *ProbDeclare* framework [14]. In [14], ProbDeclare is introduced as a framework where constraints are uncertain. Uncertainty, in turn, is characterized through a (frequentist) notion of probability based on the ratio of traces in a log that are expected to satisfy the constraint. A ProbDeclare constraint is hence declarative in two dimensions, as it at once declaratively expresses which executions traces are conforming, and how many of such conforming traces are expected to exist over the total.

In this paper, we thoroughly extend [14] along two directions. On the one hand, we provide a full account of the ProbDeclare framework, defining for the first time its formal semantics, and providing a number of examples that highlight how it works — considering the challenging interplay between the temporal and uncertainty dimensions. On the other hand, we develop ProbDeclare in the context of process mining [15]. To do so, we start from the frequentist interpretation of a ProbDeclare constraint: a constraint φ holds with probability p if, by considering all the traces contained in the log, the proportion of all traces that satisfy φ is p. In the example of partially controlled behavior above related to payments, the constraint is satisfied in a log if a ratio of 0.8 of the traces contained therein is so that if a n order acceptance event is present, then a consequent payment is also present. Notice that, being the constraint declarative, there are many different ways to satisfy the constraint, in turn contributing to the 0.8 ratio. For example, a trace where no order acceptance occurs, or a trace where order acceptance is followed by three payments, are both examples of conforming traces.

Based on this observation, we provide a comprehensive framework for *probabilistic declarative process mining*, tackling discovery, (prescriptive) monitoring, and conformance checking for probabilistic process constraints. Notably, the framework exploits as much as possible state-of-the-art techniques for certain constraints, and enriches them by suitably handling uncertainty.

This substantiates into a threefold contribution. First, we observe that probabilistic Declare constraints can be discovered off-the-shelf using already existing techniques for declarative process discovery [8–10,16,17], generating declarative process models that are, by design, consistent (in the probabilistic sense [13]). This overcomes at once the notorious consistency issues experienced when the discovered constraints are interpreted as all being certain [11].

Second, we study how to monitor probabilistic constraints, where constraints and their combinations may be in multiple monitoring states at the same time, though with different associated probabilities. This is based on the fact that a single ProbDeclare model gives rise to multiple sets of constraints (called scenarios), each with its own distinct probability, where each set fixes which model constraints are satisfied/violated therein. Specifically, we show how to lift existing automata-theoretic monitoring techniques to this more nuanced probabilistic setting.

Third, we show how (post-mortem) conformance checking [18] can be handled both at the trace and event log level, in the latter case providing, for the first time, a notion of *earth mover's distance* (EMD) that provides a numerical indicator on the degree of conformance of a log with respect to a ProbDeclare model. This complements EMD-based conformance checking approaches for (stochastic) Petri nets [19].

Notice that while results for ProbDeclare discovery and monitoring were already tackled in [14] in a preliminary form, and are extended here, conformance checking is a completely novel contribution. In addition, we provide here a proof-of concept implementation and experimental evaluation of all the introduced techniques.

The paper is structured as follows. After preliminary notions introduced in Section 2, we introduce the syntax and semantics of probabilistic constraints and ProbDeclare in Section 3. We then show in Section 4 how to reason on their interplay in logical and probabilistic terms. In Section 5, we discuss how ProbDeclare constraints can be discovered from event data using existing techniques. In Section 6, we tackle the problem of monitoring probabilistic constraints at runtime. In Section 7, we revisit conformance checking from a probabilistic angle, introducing an EMD-based measure for conformance, and discussing alternatives. In Section 8, we conduct an evaluation of the different presented techniques, using the BPIC2018 event log [20] as a basis. Section 9 discusses related work. Finally, Section 10 concludes the paper and spells out directions for future work.

2. Preliminaries

This section introduces the preliminary notions needed in the rest of the article. We start by fixing some standard notions and notation related to *multisets*. Given a set A, the *set of multisets* over A is the set of mappings of the form $m : A \to \mathbb{N}$. The set of all multisets over A is denoted by $A^{\oplus}$. Given a multiset $S \in A^{\oplus}$ over A and an element $a \in A$, $S(a) \in \mathbb{N}$ denotes the number of times a appears in S, called its *multiplicity*. Given $a \in A$ and $n \in \mathbb{N}$, we write $a^n \in S$ if $S(a) = n$. With a slight abuse of notation, we write $a \in S$ if $a^n \in S$ for some $n > 0$. The *cardinality* $|S|$ of a multiset S is the sum of the multiplicities of its elements: $|S| = \sum_{a \in S} S(a)$. We say that S is *finite* if its cardinality is finite.

We fix a finite alphabet Σ of *activities*, representing atomic execution steps in the process. A *(process) trace* τ over Σ is a finite sequence $\mathsf{a}_1 \dots \mathsf{a}_n$ of activities, where $\mathsf{a}_i \in \Sigma$ for $i \in \{1, \dots, n\}$. The *length* of trace τ is denoted by $length(\tau)$. We use the notation $\tau(i)$ to select the activity a_i present in position (also called instant) i of τ.

From a formal languages point of view, a trace is a word over a given finite alphabet Σ of propositions, which in our setting denote activities. Hence, we use Σ^* to denote the infinite set of all possible traces built from activities in Σ. A *log* over Σ is a finite multiset of traces over Σ^*; the cardinality of a log is defined as the number of traces stored therein.[1]

In the remainder of the section, we recall syntax and semantics of LTL over finite traces [6] and how it has been used to capture process constraints within declarative process modeling, with a particular emphasis on the Declare approach [2,5].

[1] From the theoretical point of view, our approach would work also in the case where the log is an infinite multiset of traces. For simplicity of exposition, in the article we do not consider this case.

2.1. LTL over finite traces

LTL over finite traces (LTL$_f$) [6] is the most widely employed temporal logic to express properties of (business) process executions. It has exactly the same syntax as standard LTL. However, as the name suggests, while LTL formulae are interpreted over infinite, recurring sequences, LTL$_f$ formulae are defined over finite (but possibly unbounded) ones.

Differently from [6], where LTL$_f$ is defined as usual over finite sequences of propositional assignments, we consider in this paper the simpler case where formulae are directly interpreted over the notion of process trace as defined above.

Definition 1. The set of LTL$_f$ formulae is recursively defined through the following grammar:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $a \in \Sigma$.

LTL$_f$ is built over atomic propositions (denoting activities in our setting) combined through boolean connectives and two temporal constructors: $\bigcirc$ ("next") and $\mathcal{U}$ ("until"). The semantics of the logic is defined by evaluating formulae over trace instants, which form a discrete measure of time.

Definition 2. Consider an LTL$_f$ formula φ, a trace τ, and a valid instant i of τ with $1 \leq i \leq length(\tau)$. We inductively define that *φ holds at instant i of trace τ*, written $\tau, i \models \varphi$, by:

- $\tau, i \models a$ for $a \in \Sigma$ iff $\tau(i) = a$;
- $\tau, i \models \neg\varphi$ iff $\tau, i \not\models \varphi$;
- $\tau, i \models \varphi_1 \vee \varphi_2$ iff $\tau, i \models \varphi_1$ or $\tau, i \models \varphi_2$;
- $\tau, i \models \bigcirc\varphi$ iff $i < length(\tau)$ and $\tau, i+1 \models \varphi$;
- $\tau, i \models \varphi_1 \mathcal{U} \varphi_2$ iff there exists some $j, i \leq j \leq length(\tau)$ such that $\tau, j \models \varphi_2$ and for every k such that $i \leq k < j$, we have $\tau, k \models \varphi_1$.

Intuitively, $\bigcirc$ denotes the *next state* operator, and $\bigcirc\varphi$ holds if there exists a next instant (i.e., the current instant does not correspond to the end of the trace), and in the next instant φ holds. Operator $\mathcal{U}$ instead is the *until* operator, and $\varphi_1 \mathcal{U} \varphi_2$ holds if φ_1 holds now and continues to hold until, in a future instant, φ_2 finally holds.

From these operators we can derive the usual boolean operators $\wedge$ and $\rightarrow$, the two formulae *true* and *false*, as well as additional temporal operators. We consider, in particular, the following three:

(eventually) $\Diamond\varphi := true \mathcal{U} \varphi$ is true, if there is a future state where φ holds;

(globally) $\Box\varphi := \neg\Diamond\neg\varphi$ is true, if now and in all future states φ holds;

(weak until) $\varphi_1 \mathcal{W} \varphi_2 := (\varphi_1 \mathcal{U} \varphi_2) \vee \Box\varphi_1$ relaxes the until operator by admitting the possibility that φ_2 never becomes true, in this case requiring that φ_1 holds globally.

We write $\tau \models \varphi$ as a shortcut notation for $\tau, 0 \models \varphi$. In this case, we say that τ *satisfies* φ (or that φ *is satisfied* by τ). In addition, we say that formula φ is *consistent* if there exists a trace τ such that $\tau \models \varphi$.

Example 1. The LTL$_f$ formula $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc})$ models that, whenever an order is closed, then it is eventually accepted. The "next" operator used is needed to guarantee that acceptance is made after at least one time instant. The structure of the formula follows what is called a *response* in Declare.

It is well known that every LTL$_f$ formula φ can be translated into a corresponding standard finite-state automaton $\mathcal{A}_\varphi$ which accepts exactly those finite traces that satisfy φ [6,21]. The complexity of reasoning with LTL$_f$ is the same as that of LTL, with satisfiability being PSPACE-complete. However, manipulation of finite-state automata performs much better in practice compared to the case of Büchi automata used when formulae are interpreted over infinite traces [22–26]. This is the main reason why LTL$_f$ has been extensively and successfully adopted within BPM to capture constraint-based, declarative processes, in particular providing the formal basis of *Declare* [21,23].

2.2. Declare

Declare is a constraint-based process modeling language based on LTL$_f$. Declare models a process by fixing a set of activities, and defining a set of temporal *(process) constraints* over them. Constraints are specified via pre-defined LTL$_f$ templates, which come with a corresponding graphical representation (see Table 1 for some prototypical Declare templates which we use in this paper). For the sake of generality, in this paper, we consider arbitrary LTL$_f$ formulae as constraints. However, in the examples, we consider formulae whose templates can be represented graphically in Declare.

Definition 3. A Declare model is a pair $\langle \Sigma, \mathcal{C} \rangle$, where Σ is a finite set of activities, and $\mathcal{C}$ is a finite set of LTL$_f$ formulae over Σ, called *constraints*.

Declare adopts a *crisp* interpretation of constraints: a trace satisfies a Declare model if it satisfies *all* constraints contained therein.

Definition 4. A trace τ *satisfies* a Declare model $\langle \Sigma, \mathcal{C} \rangle$ if $\tau \models \varphi$ holds for every $\varphi \in \mathcal{C}$. A Declare model M is *consistent* if there exists a trace τ that satisfies it.

Automata-based techniques for LTL$_f$ have been adopted in Declare to tackle fundamental tasks within the lifecycle of Declare processes, such as consistency checking [2,4], enactment and monitoring [2,21,27], as well as discovery support [10].

3. Probabilistic process constraints: Modeling and reasoning

By inspecting Definition 4, it is apparent that constraints are interpreted in an exact, *certain* way, as they must all be satisfied in every execution of the system. We now relax this assumption, and consider instead an uncertain framework where constraints may not necessarily be all satisfied.

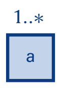
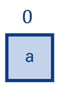
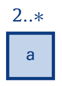
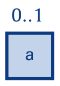
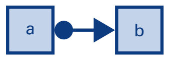
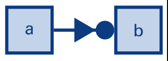
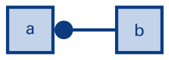
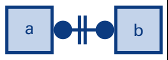
To do so, we need to tackle three problems, at increasing level of complexity. The first problem is to characterize how traces generated by the process can be interpreted probabilistically. Consider for example an order-to-cash process generating two alternative executions: one where the order is closed and then accepted, the other where the order is closed and then rejected.

Example 2. The log L_1 containing 90 repetitions of the first execution and 10 repetitions of the second is radically different from the log L_2 where the two executions appear equally often.

This shows that we need to move from an interpretation of traces as nondeterministic executions of the process, to one where their relative frequency is relevant, as it provides the footprint of the implicit stochastic behavior of the process.

The second problem pertains how to lift the semantics of a single process constraint to its probabilistic version. We use probabilities to express conditions on how likely it is for a constraint

Table 1
Some Declare templates, with their LTL_f and graphical representations.

TEMPLATE	NOTATION	TEMPLATE	NOTATION
existence(a) = $\Diamond \mathsf{a}$		absence(a) : $\neg$existence(a) = $\neg\Diamond \mathsf{a}$	
existence2(a) : $\Diamond(\mathsf{a} \wedge \bigcirc\Diamond \mathsf{a})$		absence2(a) : $\neg$existence2(a) = $\neg\Diamond(\mathsf{a} \wedge \bigcirc\Diamond \mathsf{a})$	
response(a, b) : $\Box(\mathsf{a} \rightarrow \bigcirc\Diamond \mathsf{b})$		precedence(a, b) : $\neg\mathsf{b}\mathcal{W}\mathsf{a}$	
resp-existence(a, b) : $\Diamond \mathsf{a} \rightarrow \Diamond \mathsf{b}$		not-coexistence(a, b) : $\neg(\Diamond \mathsf{a} \wedge \Diamond \mathsf{b})$	

to be satisfied in an arbitrarily selected execution of the process. As explained already, this allows one to model constraints that can be exceptionally violated, or outlying traces that are seldomly observed.

Example 3. Let C_1 be a probabilistic constraint indicating that an order is accepted with probability greater or equal than 0.7 (that is, 70% of the times or more). Considering the two logs in Example 2, we can see that L_1 satisfies the constraint (as traces containing payments exceed 70% of the total), whereas log L_2 does not, as in L_2 traces with payments cover 50% of the whole log.

The third problem emerges when multiple probabilistic constraints have to be considered at once, in the light of uncertainty. The intuition that a trace may satisfy only some of the them indicates that we need to account for multiple, alternative possible worlds (called scenarios hereafter), each indicating which constraints are satisfied and which not. The main question for scenarios is how likely they are or, more technically, what are the possible probabilities describing how likely it is to encounter each of them, given the probability conditions associated to the single constraints. These values may be impossible to determine, or may correspond to fixed numbers inducing a single discrete probability distribution over scenarios, or to (possibly infinite) sets of numbers describing a family of probability distributions compatible with the local probability conditions on constraints.

Example 4. Consider constraint C_1 from Example 3, together with another constraint C_2, expressing that with 0.5 probability the order is paid. This yields 4 possible scenarios: (1) C_1 and C_2 both hold (the order is accepted and paid), (2) C_1 holds and C_2 does not (the order is accepted, but not paid), (3) C_1 does not hold while C_2 does (the order is paid without being accepted), and (4) none of the two constraints hold (the order is not accepted nor paid). The probability of encountering one given scenario out of this four depends on how the two constraints relate to each other both in the way they constraint the process executions, and the way they constrain the probabilities of the executions satisfying (or violating) them. Considering the semantics of C_1 and C_2 and their associated probability conditions, we are in the case where there are infinitely many probability distributions describing how likely it is to encounter a process trace falling in each of the four scenarios. By combined reasoning on traces and probabilities, we could for example infer that at least 20% and at most 50% of the traces must belong to scenario (1) where the order is both accepted and paid.

In this section, we attack the first of these three problems in the order they have been introduced. We start by describing how event logs relate to stochastic languages, providing the basis for evaluating probabilistic constraint models. We then focus on single constraints, defining their shape and semantics. Before moving to sets of constraints, we recall the necessary background on the probabilistic temporal logic that provides the formal underpinnings of our approach. We finally move to sets of constraints, defining a probabilistic version of Declare, called ProbDeclare.

In Section 4, we then show how we can reason on the multiple probabilistic constraints present in a single ProbDeclare model.

3.1. Stochastic languages and event logs

To represent the traces generated by a stochastic process, we borrow the notion of *stochastic language* from [12]. Following the notation introduced in Section 2, we use Σ to denote a finite set of activities.

Definition 5. A *stochastic language* over Σ is a function $\rho : \Sigma^* \rightarrow [0, 1]$ mapping each trace over Σ onto a corresponding probability, so that

$$\sum_{\tau \in \Sigma^*} \rho(\tau) = 1 \tag{1}$$

A stochastic language ρ is *finite* if $|\{\tau \in \Sigma^* \mid \rho(\tau) > 0\}|$ is finite, that is, ρ contains finitely many traces with non-zero probability.

As argued in [12], a log can be seen as a finite stochastic language by interpreting trace multiplicities as frequencies. Specifically, the probability of a trace is computed as the ratio of the trace multiplicity and the total number of traces in the log.

Definition 6. The *stochastic language induced by a log* L over Σ is the finite stochastic language ρ_L such that for every trace $\tau \in L$, we have $\rho_L(\tau) = \frac{L(\tau)}{|L|}$.

Conversely, a finite stochastic language can be seen as a log by transforming probabilities into multiplicities.

Definition 7. The *log induced by a finite stochastic language* ρ over Σ is the multiset L_ρ such that for every trace $\tau \in \Sigma^*$, we have $L_\rho(\tau) = \rho(\tau) \cdot |L|$.

Note in particular that any trace whose probability is zero in ρ will not appear in the log L_ρ.

Example 5. Consider the following traces over $\Sigma = \{\mathsf{close}, \mathsf{acc}, \mathsf{nop}\}$:

$\tau_1 = \langle \mathsf{close}, \mathsf{acc} \rangle$
$\tau_2 = \langle \mathsf{close}, \mathsf{acc}, \mathsf{close}, \mathsf{nop}, \mathsf{acc} \rangle$
$\tau_3 = \langle \mathsf{close}, \mathsf{acc}, \mathsf{close}, \mathsf{nop} \rangle$ $\tau_4 = \langle \mathsf{close}, \mathsf{nop} \rangle$

Log $L = [\tau_1^{50}, \tau_2^{30}, \tau_3^{10}, \tau_4^{10}]$ induces the stochastic language ρ_L defined as follows: (i) $\rho(\tau_1) = 0.5$ (ii) $\rho(\tau_2) = 0.3$ (iii) $\rho(\tau_3) = 0.1$ (iv)$\rho(\tau_4) = 0.1$ (v) ρ is 0 for any other trace in Σ^*.

3.2. Probabilistic process constraints

We now extend LTL$_f$ process constraints to their *probabilistic* version. We do so by declaratively expressing conditions on the conforming process executions, and on their acceptable probabilities.

Definition 8. A *probabilistic constraint* over Σ is a triple $\langle \varphi, \bowtie, p\rangle$, where:

- φ, the *process condition*, is an LTL$_f$ formula over Σ;
- $\bowtie \in \{=, \neq, \leq, \geq, <, >\}$ is the *probability operator*;
- p, the *probability reference value*, is a rational value in $[0, 1]$.

Together, $\bowtie$ and p form the *probability condition* $\bowtie\ p$ of the constraint.

We use compact notation $\langle \varphi, p\rangle$ for the probabilistic constraint $\langle \varphi, =, p\rangle$.

Example 6. Consider the log L from Example 5, and the Declare constraint $\texttt{response}(\mathsf{close}, \mathsf{acc}) = \Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc})$. One can directly check that this constraint does not hold in the entire log, as it is violated by traces τ_3 and τ_4: such traces indeed contain at least one occurrence of close (at instant 3 for τ_3, and at instant 1 for τ_4) that is not followed by any occurrence of acc. We can then define probabilistic variants of the constraint, tolerating the possibility that their process condition could indeed be violated:

- $\langle \Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc}), 0.8\rangle$ indicates that $\texttt{response}(\mathsf{close}, \mathsf{acc})$ holds with a probability equal to 0.8;
- $\langle \Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc}), <, 0.5\rangle$ indicates that $\texttt{response}(\mathsf{close}, \mathsf{acc})$ holds with a probability that is lower than 0.5.

The semantics of these constraints will be clarified soon (see Example 7).

The semantics of probabilistic constraints is defined in terms of stochastic languages; this directly accounts also for an alternative semantics based on event logs, thanks to Definition 6. The intuition is the following. A probabilistic constraint $C = \langle \varphi, \bowtie, p\rangle$ declaratively tackles two dimensions: the process dimension via its process condition φ, which separates conforming from non conforming traces, and the space of probabilities via its probability condition $\bowtie\ p$, indicating which probability masses are legitimate. Both aspects have to be considered when judging whether a stochastic language ρ satisfies C. On the one hand, φ carves out from ρ the possibly infinitely many traces from Σ^* that satisfy φ; on the other hand, $\bowtie\ p$ expresses a condition over the collective probability mass obtained from all such traces according to ρ, which is required to satisfy $\bowtie\ p$. When $\bowtie$ is the equality operator, then the probability mass is bound to be exactly p, but when other operators are employed, infinitely many different probability masses can be chosen to satisfy the probability condition.

From now on, we implicitly assume that all the introduced notions are defined over Σ and Σ^*.

Definition 9. A stochastic language ρ *satisfies* a probabilistic constraint $C = \langle \varphi, \bowtie, p\rangle$, written $\rho \models C$, if:

$$\sum_{\tau\in\Sigma^*, \tau\models\varphi} \rho(\tau) \bowtie p \qquad (2)$$

A log L satisfies C if the stochastic language ρ_L induced by L does so.

Notice that probabilistic constraints are semantically equivalent if their probability conditions are identical, and their process conditions are logically equivalent (which does not require that they should be syntactically identical). Hence, with a slight abuse of terminology, we use the term “constraint” (in singular form) to represent all constraints that have the same probability condition, and logically equivalent process conditions.

Example 7. Consider the log L from Example 5 and the two probabilistic constraints from Example 6. L satisfies the probabilistic constraint $C_{ca} = \langle \Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc}), 0.8\rangle$. In fact, the process condition $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc})$ of C_{ca} is satisfied[2] by traces τ_1 and τ_2, whose overall probability is $0.5 + 0.3 = 0.8$. Instead, L does not satisfy constraint $\langle \Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc}), <, 0.5\rangle$, since 0.8 is not less than 0.5.

Several key observations on the interplay between the process and probability conditions of a constraint can be extracted from Definition 9. First of all, if $C = \langle \varphi, \bowtie, p\rangle$ is logically inconsistent, that is, its LTL$_f$ process condition φ is inconsistent, then the only possibility to have a stochastic language that satisfies C is to have $0 \bowtie p$. This is needed because, by definition, an inconsistent LTL$_f$ formula has no satisfying trace, which in turn means that the collective probability of satisfying traces is 0.

On the other hand, if $C = \langle \varphi, 0\rangle$, i.e., the probability condition of C has the form $= 0$, then a satisfying stochastic language ρ must assign a zero probability to all traces satisfying φ; formally: for every trace $\tau \in \Sigma^*$, if $\tau \models \varphi$ then $\rho(\tau) = 0$. Conversely, a probabilistic constraint of the form $C = \langle \varphi, 1\rangle$ indicates that all traces having a non-zero probability in ρ must satisfy φ, and so C is a *crisp constraint* that corresponds to φ in the Declare sense. When interpreted over a log, C indeed requires that all traces in the log satisfy φ.

A further key observation concerns the effect of negation or, more generally, of the relationship between a constraint and its *dual*, which we define next. Since duality involves the inversion of the probability operator used therein, we first handle this specific aspect. Given a probability operator $\bowtie\ \in \{=, \neq, \leq, \geq, <, >\}$, its *comparison-inverted operator* $\mathsf{cinv}(\bowtie)$ is defined as follows: (i)$\mathsf{cinv}(=)$ gives =; (ii) $\mathsf{cinv}(\neq)$ gives $\neq$; (iii) $\mathsf{cinv}(>)$ gives $<$; (iv) $\mathsf{cinv}(<)$ gives $>$; (v) $\mathsf{cinv}(\geq)$ gives $\leq$; (vi) $\mathsf{cinv}(\leq)$ gives $\geq$. This closely resembles what happens when employing standard algebraic techniques to solve (in)equalities: whenever the two terms of an (in)equality are multiplied by a negative factor, if the (in)equality operator relating them is a comparison operator, it has to be suitably “flipped”.

We are now ready to define duality.

Definition 10. The *dual* of a probabilistic constraint $C = \langle \varphi, \bowtie, p\rangle$ is the probabilistic constraint $\overline{C} = \langle \neg\varphi, \mathsf{cinv}(\bowtie), 1 - p\rangle$.

As the definition shows, the dual constraint $\overline{C}$ of C is obtained by: (i) negating the process condition φ of C into $\neg\varphi$; (ii) inverting the probability operator $\bowtie$ of C into $\mathsf{cinv}(\bowtie)$; (iii) replacing the probability reference value p of C with $1 - p$.

Example 8. Consider constraint C_{ca} from Example 6. By recalling that

$$\neg\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc}) = \Diamond(\mathsf{close} \wedge \neg\bigcirc\Diamond\mathsf{acc})$$

we have that $\overline{C_{ca}} = \langle \Diamond(\mathsf{close} \wedge \neg\bigcirc\Diamond\mathsf{acc}), 0.2\rangle$ is its dual constraint. The constraint indicates that traces containing a close activity not followed by any consequent acc activity should collectively have a probability of 0.2.

[2] Recall that a $\texttt{response}$ constraint is satisfied if *every* execution of the source is followed by the execution of the target.

The semantics of constraint probabilities allows us to establish the following key relationships between probabilistic constraints and their duals.

Theorem 1. *A stochastic language ρ satisfies a probabilistic constraint C if and only if ρ satisfies the dual constraint $\overline{C}$.*

Proof. Let $C = \langle \varphi, \bowtie, p\rangle$. For every trace $\tau \in \Sigma^*$, we have that either $\tau \models \varphi$, or $\tau \not\models \varphi$. Consequently, recalling that a stochastic language defines a probability distribution over Σ^* (see Eq. (1)), and that $\tau \not\models \varphi$ if and only if $\tau \models \neg\varphi$, we get:

$$\sum_{\tau\in\Sigma^*,\tau\models\varphi} \rho(\tau) + \sum_{\tau\in\Sigma^*,\tau\models\neg\varphi} \rho(\tau) = 1$$

This rewrites into

$$\sum_{\tau\in\Sigma^*,\tau\models\varphi} \rho(\tau) = 1 - \sum_{\tau\in\Sigma^*,\tau\models\neg\varphi} \rho(\tau)$$

Since, by hypothesis, ρ satisfies C, we can apply the definition of probabilistic constraint satisfaction (see Eq. (2)), obtaining

$$\left(1 - \sum_{\tau\in\Sigma^*,\tau\models\neg\varphi} \rho(\tau)\right) \bowtie (p)$$

and, in turn:

$$\left(- \sum_{\tau\in\Sigma^*,\tau\models\neg\varphi} \rho(\tau)\right) \bowtie (p-1) \tag{3}$$

Upon multiplying the left and right terms by -1, we finally obtain

$$\left(\sum_{\tau\in\Sigma^*,\tau\models\neg\varphi} \rho(\tau)\right) \mathsf{cinv}(\bowtie)\ (1-p)$$

where the original operator $\bowtie$ must suitably be replaced by its corresponding comparison-inverted operator $\mathsf{cinv}(\bowtie)$; in fact, upon multiplying the two terms of (3) by -1, operators $=$ and $\neq$ stay unaltered, while operators $>$ and $\geq$ are respectively turned into $<$ and $\leq$ and vice-versa. By applying again the definition of probabilistic constraint satisfaction, this means that ρ satisfies $\langle \neg\varphi, \mathsf{cinv}(\bowtie), 1-p\rangle$, which indeed corresponds to $\overline{C}$. □

Theorem 1 witnesses that probabilistic constraints and their duals are semantically equivalent.

When probabilistic constraints are interpreted over logs, the semantics of probabilistic constraints, and in turn the result provided in Theorem 1, can be intuitively understood in terms of frequencies. In fact, a log L satisfies $C = \langle\varphi, p\rangle$ if a fraction of p traces in L satisfy φ, which in turn require that the complementary fraction $1-p$ violates φ (that is, satisfies $\neg\varphi$). With a different reading, by randomly picking a trace τ from L, we have that τ satisfies C with probability p, while τ violates C with probability $1-p$.

Example 9. Consider again constraint C_{ca} from Example 6. When interpreted over a log L, C_{ca} captures that in 80% of the traces from L it is true that, whenever an order is closed, then it is eventually accepted. This is equivalent to assert the dual statement that in 20% of the traces from L, the `response` is violated, i.e., there exists an instant where the order is closed in an instant that is not followed by an acceptance. Given an unknown trace τ, there is then 0.8 probability that τ satisfies the response formula $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{acc})$, and 0.2 that τ violates such a formula, that is, satisfies the negated formula $\Diamond(\mathsf{close} \wedge \neg\bigcirc\Diamond\mathsf{acc})$.

Example 10. The distinction between the `existence` and `absence` templates in Declare gets blurred when their probabilistic versions are considered. This is due to the fact that, from the logical perspective, they negate each other (cf. the first two lines of Table 1). Let us focus to the case where an exact probability mass is attached to an `existence` constraint. Then, the following holds:

$$\begin{aligned}\langle \texttt{existence}(\texttt{a}), p\rangle &= \langle \Diamond\texttt{a}, p\rangle = \langle \neg\Diamond\texttt{a}, 1-p\rangle \\ &= \langle \texttt{absence}(\texttt{a}), 1-p\rangle\end{aligned}$$

The same line of reasoning applies to the `existence2` and `absence2` templates. In our uncertain setting, all such templates predicate in fact on the *probability of (repeated) occurrence* of a given activity.

So far, we have been dealing with single probabilistic constraints. We next focus on sets of constraints. This is much more challenging as the presence of multiple probabilistic constraints establishes mutual relationships among them. Such mutual relationships pertain process and uncertainty dimensions, as well as their interplay. To understand the formal basis of this twofold interplay, we need to briefly recall the PLTL^0_f logic.

3.3. Interlude: the PLTL^0_f Logic

Before moving to a probabilistic extension of Declare where multiple probabilistic constraints are simultaneously present, we note that the probabilistic constraints introduced in Definition 8 are a syntactic variant of the notion of formula in the logic PLTL^0_f, a fragment of the recently introduced probabilistic temporal logic PLTL_f [13]. In a nutshell, a PLTL^0_f formula is a set of expressions of the form $\circledcirc_{\bowtie p}\varphi$, where φ, $\bowtie$, and p are defined as in probabilistic constraints (cf. Definition 8).

While the general semantics of PLTL_f is more complex, PLTL^0_f formulae are interpreted through a simple "tree-shaped" model consisting of one root node, which branches over a finite number of finite traces;[3] in other words, a PLTL^0_f model is a finite set of LTL_f traces, representing different possible evolutions of the system. To handle the uncertainty, this set of traces has an associated probability distribution; that is, each trace is assigned a probability value in [0,1] such that the sum of the probabilities of all branches (that is, of all traces) amounts to 1.

A PLTL^0_f model M satisfies the PLTL^0_f formula Φ iff for each expression $\circledcirc_{\bowtie p}\varphi$ in Φ the probability of all the traces that satisfy φ (in the classical LTL_f sense) is $\bowtie p$. This formula Φ is *consistent* iff there is a PLTL^0_f model that satisfies it.

An important property of PLTL^0_f, which was already shown in previous work [13], is that deciding consistency of a PLTL^0_f formula is PSPACE-complete. The basic idea is to construct, one at a time, the different *scenarios* induced by the formula: combinations of LTL_f formulae appearing in Φ or their negations. Intuitively, a scenario identifies the set of traces that satisfy the same subset of LTL_f formulae that form the overall PLTL^0_f formula. If there are n probabilistic expressions, there will be 2^n different scenarios, although some of them might not be consistent. The probabilistic constraints are then verified over all the possible (locally consistent) scenarios. That is, for each $\circledcirc_{\bowtie p}\varphi$ we require that the sum of the probabilities of all scenarios that include φ satisfy $\bowtie p$. These constraints can be verified through a system of linear constraints. The full details about this construction and its properties are provided in [13], and suitably reviewed in the context of this article in Section 4.

[3] Theoretically, a model could also contain infinitely many, finite-length branches. However, it would not be distinguished from a corresponding model that compactly represents the original one using only finitely many branches. We therefore stick to finite-branching models for simplicity.

3.4. Probabilistic declare

We now lift Declare to its probabilistic version ProbDeclare, where multiple probabilistic constraints have to be considered at once. Without loss of generality, we assume that no constraint has the form $\langle\varphi, 0\rangle$ (in that case, the constraint can in fact be equivalently reformulated as its dual $\langle\neg\varphi, 1\rangle$). For simplicity of treatment, we also separate crisp constraints from genuinely probabilistic constraints. This separation still allows to reason over constraints and their interplay in our framework, as a crisp constraint φ can be seen as its corresponding probabilistic version $\langle\varphi, 1\rangle$. At the same time, this distinction is practically relevant: while a given trace may or not satisfy genuinely probabilistic constraints, it must satisfy all the crisp ones. Hence, when constructing the different scenarios, we will only need to consider possible satisfaction of violation of probabilistic constraints, assuming that all the crisp ones are always satisfied.

Definition 11. A *ProbDeclare model* is a triple $\langle\Sigma, \mathcal{C}, \mathcal{P}\rangle$, where $\mathcal{C}$ is a finite set of LTL$_f$ formulae called *crisp constraints*, while $\mathcal{P}$ is a set of (genuinely) probabilistic constraints, each having a probability condition that is different from $= 1$ (otherwise, they would be listed in $\mathcal{C}$).

From now on, we always assume that, given a ProbDeclare model $\langle\Sigma, \mathcal{C}, \mathcal{P}\rangle$, the standard Declare model $\langle\Sigma, \mathcal{C}\rangle$ consisting only of the crisp constraints is consistent. This type of consistency can be checked using standard techniques [11].

The notion of satisfaction for ProbDeclare models resembles that of standard Declare models, but now we have to consider a stochastic language (or a log) instead of a single trace. This, in turn, radically changes the resulting framework.

Definition 12. A stochastic language ρ *satisfies* a ProbDeclare model $\langle\Sigma, \mathcal{C}, \mathcal{P}\rangle$ if:

- for every crisp constraint $\varphi \in \mathcal{C}$ and every trace $\tau \in \Sigma^*$ with non-zero probability (that is, such that $\rho(\tau) > 0$), we have $\tau \models \varphi$;
- for every probabilistic constraint $C = \langle\varphi, \bowtie, p\rangle \in \mathcal{P}$, $\rho \models C$ as in Definition 9, that is, we have $\sum_{\tau\in\Sigma^*,\tau\models\varphi} \rho(\tau) \bowtie p$.

Satisfying multiple constraints at once introduces hidden dependencies among them, not only for what concerns their process dimension [4,5], but also when it comes to their probability conditions. On the one hand, all the local probability conditions attached to single constraints induce a set of global conditions on the acceptable probability distributions over Σ^*. On the other hand, the same trace may satisfy the process condition of multiple probabilistic constraints, thus influencing the probabilities of all such constraints, probabilities that are in turn required to satisfy the corresponding constraint probability conditions.

All in all, depending on the probabilistic constraints at hand, there may be one, many, or no satisfying stochastic language(s).

Definition 13. A ProbDeclare model is *consistent* if there exists at least one finite stochastic language (or, equivalently, one log) that satisfies it.

Example 11. Consider a ProbDeclare model containing a single probabilistic constraint, indicating that in at least 10% of the traces, the order is canceled and paid: $\langle\varphi_{pc}, \geq, 0.1\rangle$, with $\varphi_{pc} = \Diamond\mathsf{pay} \wedge \Diamond\mathsf{cancel}$. This model has infinitely many satisfying logs, varying in terms of traces and frequencies. All such logs have to ensure that at least 10% of the traces are so that they satisfy $\Diamond\mathsf{pay} \wedge \Diamond\mathsf{cancel}$. Two sample logs satisfying the constraint are, in this light:

- $[\langle\mathsf{close}, \mathsf{acc}, \mathsf{pay}, \mathsf{cancel}\rangle^5]$ (100% traces satisfying) φ_{pc};
- $[\langle\mathsf{close}, \mathsf{ref}, \mathsf{pay}, \mathsf{cancel}\rangle^5, \langle\mathsf{close}, \mathsf{acc}, \mathsf{pay}, \mathsf{cancel}\rangle^10,$ $\langle\mathsf{close}, \mathsf{ref}\rangle^{15}, \langle\mathsf{close}, \mathsf{acc}\rangle^{20}, \langle\mathsf{close}, \mathsf{acc}, pay\rangle^{50}]$ (15% traces satisfying) φ_{pc}.

Consider now a different ProbDeclare model containing the previous constraint and also a second constraint indicating that order cancelations rarely occur, in particular in no more than 5% of the traces: $\langle\varphi_c, \leq, 0.05\rangle$, with $\varphi_c = \Diamond\mathsf{cancel}$. We can see that the resulting ProbDeclare model is inconsistent. In fact, φ_{pc} logically implies φ_c, and consequently, every trace satisfying φ_{pc} also satisfies φ_c. Since φ_{pc} must be satisfied in at least 10% of the traces of a satisfying log, then φ_c is satisfied with that, or a higher, ratio. This clashes with the probability condition indicating that φ_c cannot be satisfied with a larger ratio than 5%.

One may wonder what is the complexity of checking consistency of a ProbDeclare model. It turns out that, thanks to the close correspondence between the PLTL^0_f logic and ProbDeclare, we can import in our setting the complexity bounds obtained for PLTL^0_f in [13]. Interestingly, recalling that many different reasoning tasks can be reduced to consistency, these complexity bounds indicate that reasoning in ProbDeclare yields the same worst-case complexity as that of reasoning in the (non-probabilistic) LTL$_f$ setting.

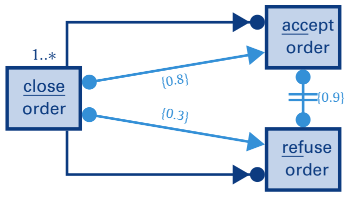
Theorem 2. *Checking consistency of ProbDeclare models is* PSPACE-*complete.*

Proof. Consider the definition of ProbDeclare model (Definition 11), and the PLTL^0_f logic recalled in Section 3.3. We encode a ProbDeclare model $M = \langle\Sigma, \mathcal{C}, \mathcal{P}\rangle$ intoPLTL^0_f formula Φ_M whose expressions encode the constraints in $\mathcal{C}$ and $\mathcal{P}$ as follows: for every crisp constraint $\psi \in \mathcal{C}$, we build the PLTL^0_f expression $\circledcirc_{=1}\psi$, and for every probabilistic constraint $\langle\varphi, \bowtie, p\rangle$, we build the PLTL$^0_f$ expression $\circledcirc_{\bowtie p}\varphi$. As mentioned earlier PLTL$_f$ (and thus PLTL^0_f) formulae are interpreted over finite trees, where nodes are propositional assignments, and branches carry probabilities, with the condition that the sum of such probabilities is 1. A finite stochastic language ρ can be straightforwardly represented as a tree of this form as follows: every trace τ in ρ such that $\rho(\tau) > 0$ becomes a branch of the corresponding tree, with probability $\rho(\tau)$ attached to it.

We then get that M is consistent if and only if Φ_M is satisfiable. Checking satisfiability of Φ_M is a PSPACE-complete task [13, Theorem 18]. □

Theorem 2 does not yet provide a concrete technique to actually carry out reasoning and, more generally, understand how different probabilistic constraints interact with each other. We close this section with an example that highlights the intricacies of such an interaction. In the next section we will systematically attack these aspects by proposing a provably correct, operational technique based on the notion of scenario, recalled in Section 3.3 for PLTL^0_f.

Example 12. Consider the following ProbDeclare model, where crisp constraints are shown in dark blue, and probabilistic constraints are in light blue with their corresponding exact probability reference values (the operator is always implicitly =).



The model expresses that each order is at some point closed, and, whenever this happens, there is probability 0.8 that it will be eventually accepted, and probability 0.3 that it will be eventually refused. Due to the two `precedence` constraints, accepting/rejecting an order can only occur if the order was closed. Since the sum of these probability masses exceeds 1, a fraction of traces will contain both an acceptance and a rejection. This declaratively captures the state of affairs in which a previous decision made on a closed order is later reversed. On the other hand, there is a sensible amount of traces where the order will be eventually accepted, but not refused, given the fact that the probability reference value of the `response` constraint connecting close to ref is only 0.3. In 90% of the cases, it is asserted that acceptance and rejection are mutually exclusive; if this would be a crisp constraint, it would conflict with the need of having a fraction of traces where the order is accepted and refused.

4. Reasoning over multiple probabilistic constraints

Is the model in Example 12 consistent? What is the probability of encountering a trace where an order is closed, and then both refused and accepted? To answer these questions, we need to understand which alternative possible process executions are implicitly described by a ProbDeclare model, and what are their respective, legitimate probabilities — lifting uncertainty from the local level of constraints to the global level where all constraints are taken into account at once.

As we have seen for PLTL$_f^0$ in Section 3.3, these alternative process executions arise from the fact that a valid execution may, in principle, satisfy some probabilistic constraints, and violate others, and the set of satisfied/violated constraints may be different from that of another, valid execution. This yields multiple possible worlds, which we call *(constraint) scenarios* consistently with [13]. Each scenario fixes which probabilistic constraints are satisfied; this also determines that the other probabilistic constraints are violated. A scenario, then, is nothing else than a declarative, finite description of the (possibly infinite) set of traces that all share the properties of satisfying/violating the probabilistic constraints, as dictated by the scenario.

In this section, we show how we can formally characterize, and reason upon, scenarios and their probabilities. This provides the foundational basis for the process mining tasks described in the rest of the article. Specifically, we proceed according to the following steps. We first introduce constraint scenarios. Then we show how scenarios can be logically and probabilistically characterized, finally introducing an operational technique to provide combined reasoning on scenarios and their probability distributions.

4.1. Constraint scenarios

We now proceed, step by step, to highlight the different facets of scenarios, starting from their definition. To simplify our technical treatment, from now on we fix an ordering over the probabilistic constraints in a ProbDeclare model, thus representing them as a tuple instead of a set.

Consider a ProbDeclare model with n probabilistic constraints. In principle, a model of this form implicitly yields 2^n possible scenarios. Each scenario picks which process conditions from the probabilistic constraints are satisfied, and which are violated. In this light, a scenario implicitly describes all traces that satisfy the process conditions selected by the scenario.

Definition 14. A *scenario* of a ProbDeclare model $M = \langle \Sigma, \mathcal{C}, \langle \varphi_1, p_1 \rangle, \dots, \langle \varphi_n, p_n \rangle \rangle$ is a total function $\sigma : \{1, \dots, n\} \to \{0, 1\}$, where $\sigma(i) = 1$ indicates that the LTL$_f$ process condition φ_i is satisfied in the scenario, while $\sigma(i) = 0$ indicates that φ_i is violated in the scenario (that is, $\neg\varphi_i$ is satisfied therein). As a compact, explicit notation, we denote σ as $S^M_{\sigma(1)\cdots\sigma(n)}$, or simply $S_{\sigma(1)\cdots\sigma(n)}$ when M is clear from the context. We also employ notation S^M_k or S_k, where k is a decimal number whose binary encoding coincides with $\sigma(1)\cdots\sigma(n)$.

Example 13. Fig. 1 builds on the ProbDeclare model introduced in Example 12, indicating its induced scenarios. The model contains 6 constraints, three crisp and three probabilistic. Circled numbers represent the ordering of such constraints. Since we have 3 probabilistic constraints, $2^3 = 8$ possible constraint scenarios are induced, each enforcing the satisfaction of the three crisp constraints, and deciding on the satisfaction or violation of the three constraints `response`(close, acc), `response`(close, ref), and `not-coexistence`(acc, ref).

The resulting scenarios are reported in the same figure, using the naming conventions introduced before, in agreement with the constraint ordering. For example, scenario S_{101} is the scenario that satisfies `response`(close, acc) and `not-coexistence`(acc, ref), but violates `response`(close, ref).

4.2. Logical characterization and consistency of scenarios

As we already pointed out, each scenario provides a canonical representation for all the (possibly infinitely many) traces that all agree on the satisfaction/violation of constraints as indicated by the scenario. Three questions immediately arise: (i) how does one check to which scenario(s) a trace belongs? (ii) Can a trace belong to multiple scenarios? (iii) Are all scenarios meaningful, or should we discard some of them?

To answer such questions, we provide a logical characterization of scenarios. First and foremost, we introduce a characteristic LTL$_f$ formula for a scenario: a trace belongs to a scenario if and only if the trace satisfies the characteristic formula of the scenario.

Definition 15. Let $M = \langle \Sigma, \mathcal{C}, \langle \langle \varphi_1, \bowtie_1, p_1 \rangle, \dots, \langle \varphi_n, \bowtie_n, p_n \rangle \rangle \rangle$ be ProbDeclare model. The *characteristic formula* induced by a scenario $S^M_{b_1\cdots b_n}$ over M, compactly called $S^M_{b_1\cdots b_n}$-formula, is the LTL$_f$ formula

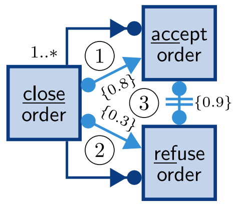
$$\Phi(S^M_{b_1\cdots b_n}) = \bigwedge_{\psi \in \mathcal{C}} \psi \wedge \bigwedge_{i \in \{1,\dots,n\}} \begin{cases} \varphi_i & \text{if } b_i = 1 \\ \neg\varphi_i & \text{if } b_i = 0 \end{cases}$$

Definition 16. A trace τ *belongs to scenario* $S^M_{b_1\cdots b_n}$ if $\tau \models \Phi(S^M_{b_1\cdots b_n})$. Scenario $S^M_{b_1\cdots b_n}$ is *consistent* if there is at least one trace that belongs to it.

Consistency of scenarios correspond to the usual notion of satisfiability in LTL$_f$. An inconsistent scenario can be dropped, as no trace can belong to it.

Example 14. We continue Example 13 by focusing on the logical characterization, and consistency, of the 8 scenarios introduced there.

The characteristic formulae of the different scenarios are built by conjoining the LTL$_f$ formulae of the crisp constraints and those



	①	②	③	CONSISTENT?
S_{000}	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})$	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{ref})$	$\Diamond\text{acc} \wedge \Diamond\text{ref}$	no
S_{001}	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})$	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{ref})$	$\neg(\Diamond\text{acc} \wedge \Diamond\text{ref})$	**yes**
S_{010}	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})$	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{ref})$	$\Diamond\text{acc} \wedge \Diamond\text{ref}$	no
S_{011}	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})$	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{ref})$	$\neg(\Diamond\text{acc} \wedge \Diamond\text{ref})$	**yes**
S_{100}	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{ref})$	$\Diamond\text{acc} \wedge \Diamond\text{ref}$	no
S_{101}	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$	$\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{ref})$	$\neg(\Diamond\text{acc} \wedge \Diamond\text{ref})$	**yes**
S_{110}	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{ref})$	$\Diamond\text{acc} \wedge \Diamond\text{ref}$	**yes**
S_{111}	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$	$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{ref})$	$\neg(\Diamond\text{acc} \wedge \Diamond\text{ref})$	no

Fig. 1. A ProbDeclare model, with 8 constraint scenarios, out of which only 4 are consistent. Recall that each scenario induces a formula that does not simply conjoin the positive/negated variants of the probabilistic constraints, but includes also the conjunction of the formulae for crisp constraints.

of the probabilistic constraints, for the latter deciding, one by one, whether to keep the formula in its positive or negated form. For example, considering scenario S_{101}, we have $\Psi(S_{101}) =$

$$\begin{aligned}&\texttt{existence}(\text{close}) \wedge \texttt{precedence}(\text{close}, \text{acc})\\ &\wedge \texttt{precedence}(\text{close}, \text{ref})\\ &\wedge \texttt{response}(\text{close}, \text{acc}) \wedge \neg\texttt{response}(\text{close}, \text{ref})\\ &\wedge \texttt{not-coexistence}(\text{acc}, \text{ref})\end{aligned}$$

which, in turn, is the LTL$_f$ formula

$$\begin{aligned}&(\Diamond\text{close}) \wedge ((\neg\text{acc})\,\mathcal{W}\,\text{close}) \wedge ((\neg\text{ref})\,\mathcal{W}\,\text{close})\\ &\wedge (\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})) \wedge (\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{ref}))\\ &\wedge (\neg(\Diamond\text{acc} \wedge \Diamond\text{ref}))\end{aligned}$$

Checking whether this LTL$_f$ formula is satisfiable tells us whether scenario S_{101} is consistent or not. More generally, as indicated in Fig. 1 only 4 scenarios are actually consistent.

S_{101} is consistent: a trace witnessing consistency is the one in which an order is closed and then accepted. Notice that there are infinitely many other traces that belong to this scenario. For example, one may repeat the acceptance an arbitrary amount of times, still leading to a trace that belongs to S_{101}.

Instead, S_{111} is *not* consistent. In fact:

- it requires that the order is closed (due to the crisp 1..∗ constraint on close);
- consequently, the order is eventually accepted and refused, due to the two response constraints attached to close, which in this scenario must be both satisfied;
- however, the presence of both an acceptance and a refusal violates the `not-coexistence` constraint linking such two activities, contradicting the requirement that also this constraint must be satisfied in this scenario.

All in all, we get 4 consistent scenarios:

- S_{001}, where an order must be closed and later not accepted nor refused;
- S_{011}, where an order must be closed and later refused (and not accepted);
- S_{101}, where an order must be closed and later accepted (and not refused);
- S_{110}, where an order must be closed and later accepted and refused.

To close with the logical characterization of scenarios, we have that scenarios partition the set of all traces in Σ^* that satisfy the crisp constraints.

Theorem 3. *Given a ProbDeclare model M. For every trace $\tau \in \Sigma^*$ that satisfies all crisp constraints in M, we have that τ belongs to one and only one scenario of M.*

Proof. Let $M = \langle \Sigma, \mathcal{C}, \langle\langle\varphi_1, \bowtie_1, p_1\rangle, \ldots, \langle\varphi_n, \bowtie_n, p_n\rangle\rangle\rangle$. For an LTL$_f$ formula φ, we either have $\tau \models \varphi$ or $\tau \models \neg\varphi$. We then proceed as follows. Let $b_1 \cdots b_n$ be a sequence of n bits. For every $i \in \{1, \ldots, n\}$, set $b_i = 1$ if $\tau \models \varphi_i$, $b_i = 0$ otherwise. By definition, τ then belongs to $S^M_{b_1 \cdots b_n}$. $\square$

4.3. *Probabilistic characterization of scenarios and combined reasoning*

We now move to the probabilistic characterization of scenarios. This is extremely important, as it tells us what are the relative, acceptable cumulative frequencies of traces belonging to each scenario. Once this information is incorporated, scenarios provide a canonical representation of all the possible logs/finite stochastic languages that satisfy the ProbDeclare model at hand. These stochastic languages may vary in terms of probability distributions for two reasons. First, probabilistic constraints may come with probability conditions that accept multiple, possibly infinitely many actual probability masses. Second, from the logical point of view each constraint may have multiple, possibly infinitely many distinct traces satisfying it. In this case, the probability conditions would not dictate a specific probability mass for each such traces, but just condition their overall probability mass they get collectively. By using the terminology of probability theory, this means that scenarios yield a so-called *credal set* of probability distributions.

To obtain the legitimate probability distributions over scenarios, three initial observations are in place. A first observation, which shows how the logical and probabilistic characterizations of scenarios interact with each other, is that if a scenario is inconsistent, it must forcefully be associated to a probability mass of 0 - considering that it has no satisfying trace. A second observation is that the allowed probability masses for consistent scenarios have to be chosen so as to respect the probability condition of each probabilistic constraint that is satisfied in that scenario. However this cannot be naively done by considering each scenario in isolation. In fact, a probabilistic constraint may be satisfied in multiple distinct consistent scenarios, consequently requiring that its probability condition is respected by the cumulative probability masses of all such scenarios.

Interestingly, when characterizing the allowed probability masses for such scenarios, it may very well turn out that a consistent scenario gets probability 0. An even more extreme situation may arise, namely the one where it may not be at all possible to find such probability masses. This indicates that ProbDeclare models may be inconsistent due to the impossibility of extracting a probability distribution from the probability conditions of its probabilistic constraints, considering that inconsistent scenarios must have 0 probability.

Example 15. Consider the four consistent scenarios of Fig. 1 and Example 14. We provide some intuitive observations on probabilistic constraints and scenarios, which will then be systematically handled in the remainder of the section.

Three out of these four scenarios, namely S_{001}, S_{011}, and S_{101}, indicate that the `not coexistence` constraint relating acc and ref is satisfied. This means that their overall, combined probability mass must be in line with the one assigned to the constraint, namely the exact value 0.9. The fourth consistent scenario is S_{110}, which is the only consistent scenario where the `not coexistence` constraint is false. Considering that the sum of the probability masses associated to all consistent scenarios must be 1, we have that S_{110} must be then associated to a probability mass of $1-0.9 = 0.1$. This is in line with the probability reference values assigned to the two probabilistic `response` constraints: the fact that their probabilities sum up to 1.1 indicates that there must be traces satisfying both of them with a *minimum* ratio of $1.1 - 1 = 0.1$.

Two scenarios, namely S_{101} and S_{110}, indicate that `response`(close, acc) is true. The probability condition attached to this constraint is $= 0.8$, and corresponds to the cumulative probability mass obtained by summing the probability mass of such two scenarios. Since, by what observed before, the probability of S_{110} is 0.1, this implies that the probability of S_{101} is 0.7.

By a similar line of reasoning, applied to constraint $\langle$`response`(close, acc), 0.3$\rangle$, we can infer that the probability mass of S_{011} is actually 0.2. Since the probability masses of the three scenarios S_{101}, S_{110}, and S_{011} saturate 1, we finally infer that scenario S_{001} cannot have any satisfying trace, as its probability mass is 0. This does not happen due to logical inconsistency, but comes from the interplay between the logical characterization of scenarios (which selects four consistent scenarios out of the total) and the probabilistic characterization of such scenarios as induced by the probability conditions attached to the constraints that are satisfied therein.

To systematically characterize the possible probabilities masses associated to scenarios, we reconstruct the approach of PLTL_f^0 [13, Theorem 17] in our setting. In particular, to compute the possible distributions of probability masses associated to consistent scenarios, we set up a system of inequalities whose solutions constitute all the probability distributions that are compatible with the logical and probabilistic characterization of the probabilistic constraints in the ProbDeclare model of interest. To do so, we associate each scenario to a probability variable, keeping the same naming convention. For example, the probability mass of scenario S_{001} is represented by variable x_{001}. For $M = \langle \Sigma, \mathcal{C}, \langle\langle\varphi_1, \bowtie_1, p_1\rangle, \ldots, \langle\varphi_n, \bowtie_n, p_n\rangle\rangle\rangle$, we construct the system $\mathcal{L}_M$ of inequalities using probability variables x_i, with i ranging from 0 to $2^n - 1$ (in binary format):

$$x_i \geq 0 \quad 0 \leq i < 2^n \tag{4}$$

$$\left(\sum_{i=0}^{2^n-1} x_i\right) = 1$$

$$\left(\sum_{\substack{i\in\{0,\ldots,2^n-1\},\\ j\text{th position of } i \text{ is } 1}} x_i\right) \bowtie_j p_j \quad 0 \leq j < n \tag{5}$$

$$x_i = 0 \quad 0 \leq i < 2^n, \text{scenario } S_i \text{ is inconsistent} \tag{6}$$

The first two lines guarantee that variables x_i indeed form a probability distribution, being all non-negative and collectively summing up to 1. The schema of inequalities captured in Eq. (5) verifies the probability associated to each probabilistic constraint in M, lifting local probability conditions to global conditions over scenarios. Specifically, one inequality per probabilistic constraint $\langle\varphi_j, \bowtie_j, p_j\rangle$ in M is generated. The (in)equality ensures that the collective sum of probability masses attached to all scenarios where that constraint is true, should all yield a resulting probability mass that satisfies condition $\bowtie_j p_j$. Finally, the schema of inequalities captured in Eq. (6) is where logical and probabilistic reasoning are connected: for every inconsistent scenario, the inequality states that its probability mass is 0. This guarantees that only consistent scenarios may receive a positive probability mass in $\mathcal{L}_M$.

All in all, we get that a ProbDeclare model M is consistent if and only if $\mathcal{L}_M$ admits a solution (i.e., admits at least one probability distribution over scenarios). Even more, solving $\mathcal{L}_M$ corresponds to verifying whether it is possible to find a log whose traces can be assigned to the different scenarios, ensuring that all the constraint probability conditions are respected by the ratios of traces in each scenario. Checking whether $\mathcal{L}_M$ admits a solution can be done in PSPACE in the size of M, if we calculate the size of M as the length of the LTL_f formulae appearing in its crisp and probabilistic constraints [13].

Example 16. Consider the ProbDeclare model M containing two probabilistic constraints:

1. `existence`(close) = $\Diamond$close with probability $= 0.1$;
2. `response`(close,acc) = $\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$ with probability $= 0.8$.

When interpreted over a log, M indicates that only 10% of the traces contain that the order is closed, and that 80% of the traces are so that, whenever an order is closed, it is eventually accepted. This model is inconsistent. Intuitively, the fact that in 80% of the traces, whenever an order is closed, it is eventually accepted, is equivalent to say that, in 20% of the traces, we violate such a `response` constraint, i.e., we have that an order is closed but then not accepted. All such traces satisfy the `existence` constraint over the close activity, and, consequently, the probability of such a constraint must be at least 0.2. However, this is contradicted by the first constraint of M, which imposes that such a probability is 0.1.

We now show how this is detected formally. M yields 4 constraint scenarios:

$$\begin{aligned}
S_{00} &= \{\neg\Diamond\text{close}, \Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})\}\\
S_{01} &= \{\neg\Diamond\text{close}, \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})\}\\
S_{10} &= \{\Diamond\text{close}, \Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})\}\\
S_{11} &= \{\Diamond\text{close}, \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})\}
\end{aligned}$$

Scenario S_{00} is inconsistent: it requires and forbids that the order is closed. The other scenarios are instead all consistent. Hence, $\mathcal{L}_M$ is[4]:

$$\begin{array}{ccccccccc}
x_{00} & + & x_{01} & + & x_{10} & + & x_{11} & = & 1\\
 & & & & x_{10} & + & x_{11} & = & 0.1\\
 & & x_{01} & & & + & x_{11} & = & 0.8\\
x_{00} & & & & & & & = & 0
\end{array}$$

This system yields $x_{10} = 0.2$, $x_{01} = 0.9$, and $x_{11} = -0.1$. This is an inconsistent probability assignment, and witnesses that it is not possible to define a probability distribution over scenarios that agrees with the local probability conditions imposed by the two probabilistic constraints.

[4] We omit, for compactness, the inequalities of type (4) indicating that each variable must be ≥ 0.

If a ProbDeclare model M is consistent, then $\mathcal{L}_M$ has at least one solution, and it may have infinitely many, possibly requiring the different probability masses for a scenario to be scattered in different subintervals of $[0, 1]$. Hence we need a way to extract some practical, finitely representable information about such different solutions. We do so through the notion of scenario probability box, storing bounds on the probability masses that scenarios of M can get assigned to.

Definition 17. The *scenario probability box* $\mathbb{P}_M$ of a consistent ProbDeclare model M is a total function mapping every scenario of M into a closed interval contained in $[0, 1]$, such that for every scenario S_i of M, we have $\mathbb{P}_M(S_i) = [inf_i, sup_i]$ if and only if:

- inf_i is the solution of the optimization problem that minimizes x_i over $\mathcal{L}_M$;
- sup_i is the solution of the optimization problem that maximizes x_i over $\mathcal{L}_M$.

Note that this definition does not (nor it intends to) claim that for every value $p \in \mathbb{P}_M(S_i)$ in the probability box, there is a model that assigns that probability to S_i. The probability box provides only the extreme values as an approximate way to understand the behavior of these scenarios: it excludes some values *a priori*, and can be refined as values for other scenarios are picked, by updating the system of inequalities. How much the interval singled out by a probability box is informative depends on the actual space of solutions for the model at hand.

Obviously, whenever $\mathcal{L}_M$ has a unique solution, we can avoid invoking the optimization problems, and directly assign to each scenario a probability box whose extreme values coincide to the probability mass from the solution. In that case, the probability boxes provide full information, and do not require anymore to look into $\mathcal{L}_M$. This is not the case when multiple solutions exist.

The scenario probability box $\mathbb{P}_M$ of a consistent ProbDeclare model M, together with the system $\mathcal{L}_M$, can be used to:

- *Extract probability mass distributions over scenarios* that agree with M. This is done by picking, for every scenario S_i, one probability mass value within $\mathbb{P}_M(S_i)$, in such a way that all the chosen values actually constitute a solution for $\mathcal{L}_M$ (which can be checked in linear time).
- *Check whether a finite stochastic language ρ satisfies M*. To do so, we proceed as follows. For every scenario S_i of M, we compute the probability mass $P_\rho(S_i)$ induced by ρ on S_i as:

$$P_\rho(S_i) = \sum_{\tau \in \Sigma^*,\ \rho(\tau)>0,\ \tau \text{ belongs to } S_i} \rho(\tau)$$

  We then verify that the so-computed probability masses provide an actual probability distribution, which can be checked by feeding them into $\mathcal{L}_M$ and checking whether the so-obtained ground inequalities are all satisfied.

Example 17. Consider the ProbDeclare model in Fig. 1. We now substantiate the intuitive reasoning described in Example 15 with a systematic computation of scenario probabilities. The system of inequalities for the model is so that $x_{000} = x_{010} = x_{100} = x_{111} = 0$, since the corresponding scenarios are all inconsistent. For the consistent scenarios, we instead get the following equalities, once the variables above are removed (being them all equal to 0):

$$\begin{array}{ccccccccc} x_{001} & + & x_{011} & + & x_{101} & + & x_{110} & = & 1 \\ & & & & x_{101} & + & x_{110} & = & 0.8 \\ & & x_{011} & & & + & x_{110} & = & 0.3 \\ x_{001} & + & x_{011} & + & x_{101} & & & = & 0.9 \end{array}$$

It is easy to see that this system of equations admits only one solution: $x_{001} = 0, x_{011} = 0.2, x_{101} = 0.7, x_{110} = 0.1$. This solution witnesses that scenario S_{001} has zero-probability, and that the most likely scenario, holding in 70% of cases, is actually S_{101}, namely the one where after the order is closed, it is eventually accepted, and not refused. In addition, the solution tells us that there are other two unlikely scenarios: the first, holding in 20% of cases, is the one where, after the order is closed, it is eventually refused (and not accepted); the second, holding in 10% of cases, is the one where a closed order is accepted and refused.

We finish this section with an example of ProbDeclare model whose corresponding system of inequalities admits infinitely many solutions.

Example 18. Consider the ProbDeclare model in Fig. 2. It comes with 4 constraint scenarios, obtained from the two process conditions precedence(sign,close) = $\neg$close $\mathcal{W}$ sign and response (close, sign) = $\Box$(close $\rightarrow$ $\bigcirc\Diamond$sign), as well as their respective negated formulae $\neg$sign $\mathcal{U}$ close and $\Diamond$(close $\wedge$ $\neg\bigcirc\Diamond$sign). All such scenarios are consistent, and hence the resulting system of inequalities is:

$$x_{00} \geq 0 \quad x_{01} \geq 0 \quad x_{10} \geq 0 \quad x_{11} \geq 0$$

$$\begin{array}{ccccccccc} x_{00} & + & x_{01} & + & x_{10} & + & x_{11} & = & 1 \\ & & & & x_{10} & + & x_{11} & = & 0.8 \\ & & x_{01} & & & + & x_{11} & = & 0.1 \end{array}$$

This system admits multiple solutions. In fact, by calculating the minimum and maximum values for the 4 variables, we get the following scenario probability box:

- scenario S_{00}, where the order is closed but consent is not signed, gets probability interval $[0, 0.1]$;
- scenario S_{01}, where the order is closed and consent is signed afterward, gets probability interval $[0, 0.1]$;
- scenario S_{10}, where the order is closed after having signed consent, gets probability interval $[0.7, 0.8]$;
- scenario S_{11}, where the order is closed and consent is signed at least twice (once before, and once afterward), gets probability interval $[0.1, 0.2]$.

Since the probability box involves intervals of limited size, it provides a good, approximated view over the actual probability distributions that can be faithfully employed instead of solving the system of inequalities whenever needed.

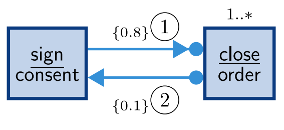
5. Discovering ProbDeclare models from event logs

We now show that ProbDeclare models can be discovered from event data using, off-the-shelf, already existing techniques, with a quite interesting property: that the discovered ProbDeclare model is always guaranteed to be consistent (in the probabilistic sense of Definition 13).

A variety of different algorithms has been devised to discover Declare models from event data [7–10,16]. In general, the vast majority of these algorithms adopts the following approach to discovery:

- Candidate formulae are generated by analyzing the activities contained in the log.
- For each formula, its *support* is computed as the fraction of traces in the log where the constraint holds.
- Candidate formulae are filtered, retaining only those whose support exceeds a given threshold.
- Further filters are applied, for example considering aspects such as redundancy, interestingness, and vacuity [7,11,28].

In this pipeline, the notion of support is typically formalized as follows.



	1	2	CONSISTENT?
S_{00}	¬sign $\mathcal{U}$ close	$\Diamond$(close $\wedge$ ¬$\bigcirc\Diamond$sign)	yes
S_{01}	¬sign $\mathcal{U}$ close	$\Box$(close $\rightarrow$ $\bigcirc\Diamond$sign)	yes
S_{10}	¬close $\mathcal{W}$ sign	$\Diamond$(close $\wedge$ ¬$\bigcirc\Diamond$sign)	yes
S_{11}	¬close $\mathcal{W}$ sign	$\Box$(close $\rightarrow$ $\bigcirc\Diamond$sign)	yes

Fig. 2. A ProbDeclare model and its 4 constraint scenarios.

Definition 18. The *support* of an LTL$_f$ formula φ in an event log L is

$$supp_L(\varphi) = \frac{\sum_{\tau\in L, \tau\models\varphi} L(\tau)}{|L|}$$

To obtain a meaningful Declare model in output, there is one additional, crucial catch: the formulae that pass all the steps of the pipeline may result in an overall inconsistent model. The reason is that formulae with a high support strictly less than 1 may actually conflict with each other [11,29]; this is not recognized by the model, which does not keep nor use any information related to support. Fixing these potential inconsistencies calls then for additional post-processing techniques [11].

5.1. Support as uncertainty

Differently from Declare, in ProbDeclare we can interpret support as uncertainty, using it to fine-tune the constraint probability. This leads to explicitly maintain support-related information at the model level, with a well-defined semantics and reasoning capabilities. The most straightforward way to use support is to adjust the discovery pipeline described above as follows:

- each discovered formula with support 1 can be retained as a crisp constraint;
- each discovered formula φ with support $p < 1$ can be retained as a probabilistic constraint of the form $\langle\varphi, p\rangle$ (that is, with probability condition $= p$).

Example 19. Consider $L = [\langle$close, acc$\rangle^7$, $\langle$close, ref$\rangle^2$, $\langle$close, acc, ref$\rangle^1]$, capturing the evolution of 10 orders, 7 of which have been closed and then accepted, 2 of which have been closed and then refused, and 1 of which has been closed, then accepted, then refused. The support of constraint `response`(close,acc) is $8/10 = 0.8$, witnessing that 8 traces satisfy such a constraint, whereas 2 violate it. This corresponds exactly to the interpretation of probability 0.8 for the probabilistic `response`(close,acc) constraint in Fig. 1. More generally, the entire ProbDeclare model of Example 12 can be discovered from L.

It turns out that a ProbDeclare model discovered in this way enjoys the key property of being consistent, no matter which subset of the crisp and probabilistic constraints is retained. Consequently, we do not need to apply any post-processing step tailored to guarantee consistency, while we can carry out combined reasoning on the discovered constraints and their support.

Theorem 4. *Let L be a log over Σ, and $M = \langle\Sigma, \mathcal{C}, \mathcal{P}\rangle$ be a ProbDeclare model. If:*

- *for every crisp constraint $\psi \in \mathcal{C}$, we have $supp_L(\psi) = 1$, and*
- *for every probabilistic constraint $\langle\varphi, \bowtie, p\rangle \in \mathcal{P}$, we have that $\bowtie$ is the equality operator and that $p = supp_L(\varphi)$,*

then M is consistent.

Proof. Let M be the ProbDeclare model as in the theorem. We show that consistency is witnessed by L itself, in the sense that the stochastic language ρ_L induced by L has the property of satisfying M. To satisfy M, L must satisfy every crisp and probabilistic constraint therein. The case of crisp constraints is straightforward.

As for probabilistic constraints, let $C = \langle\varphi, supp_L(\varphi)\rangle \in \mathcal{P}$ be a probabilistic constraint in M. By Definition 9, to satisfy C the stochastic language ρ_L must satisfy the following equation: $\sum_{\tau\in\Sigma^*, \tau\models\varphi} \rho(\tau) = supp_L(\varphi)$.

On the other hand, since each τ contributes to the left-hand sum if and only if $\rho(\tau) > 0$, which in turn happens if and only if $\tau \in L$, we can rewrite the left-hand side into: $\sum_{\tau\in L, \tau\models\varphi} \rho(\tau) = supp_L(\varphi)$.

By Definition 6, we know that $\rho(\tau) = \frac{L(\tau)}{|L|}$, thus obtaining:

$$\sum_{\tau\in L, \tau\models\varphi} \rho(\tau) = \sum_{\tau\in L, \tau\models\varphi} \frac{L(\tau)}{|L|} = \frac{\sum_{\tau\in L, \tau\models\varphi} L(\tau)}{|L|}$$

which, by Definition 18, coincides with $supp_L(\varphi)$. □

Thanks to the result of this theorem, probabilistic constraints can be discovered in a *purely local* way, having the guarantee that they will never conflict with each other. However, the theorem does not give any hint on how many discoverable constraints have to be inserted in the discovered model, nor which are more interesting than others. Hence, in principle one could apply the brute-force approach of [7], or the discovery algorithms in [8–10] by setting 0 as minimum support threshold, thus generating *all* possible constraints regardless of their support. Afterward, in the spirit of steps 3 and 4 above, filters applied to single constraints or considering the non-local interplay of multiple constraints can be applied to keep only the most interesting ones.

5.2. Discovery of Relaxed ProbDeclare Models

In the previous section, we have used support as the *exact* reference probability value of the discovered constraints. We close the section by showing three different relaxations of ProbDeclare discovery, where support is used in a less constraining way. Being relaxations of the original discovery technique, they all continue to satisfy Theorem 4, and are hence consistent by design.

The first relaxation explores the notion of constraint duality (cf. Definition 10). Duality of probabilistic constraints tells that a probabilistic constraint with very low probability reference value p (read: a very low support during discovery) is definitely of interest, since its dual, equivalent constraint has a very high probability reference value $1 - p$ (read: a very high support). Consequently, if $\chi \in [0, 1]$ is the minimum support threshold used in step 3 of the discovery pipeline listed above, then a variant of the discovery algorithm could employ a modified filter of the following form: formula φ is retained if $supp_L(\varphi) \geq \chi$ or $supp_L(\varphi) \leq 1 - \chi$.

Example 20. Consider again the log in Example 19. By setting a minimum threshold $\chi = 0.7$ for support, all crisp and probabilistic constraints shown in Example 12 would be retained, as their probability/support values are either greater or equal than 0.7, or less or equal than 0.3. By setting $\chi = 0.85$, instead, only the crisp constraints and the `not-coexistence` probabilistic constraint would be kept.

The two second relaxations arise from the observation that discovering a probabilistic constraint of the form $\langle \varphi, supp_L(\varphi)\rangle$ is very sensitive to changes in the log, as the constraint requires the ratio of traces in the log satisfying φ to be *exactly* the value of the support.

A first way of relaxing this is to introduce a *confidence interval* $\xi \in [0, 1]$ for probability conditions, and use it to replace $\langle \varphi, supp_L(\varphi)\rangle$ with the following two probabilistic constraints:

- $\langle \varphi, \geq, p_1\rangle$, with $p_1 = max\{0, supp_L(\varphi) - \xi/2\}$;
- $\langle \varphi, \leq, p_2\rangle$, with $p_2 = min\{1, supp_L(\varphi) + \xi/2\}$.

These two constraints can be seen as a single probability constraint whose probability condition indicates that the probability mass of the constraint belongs to an interval of size ξ centered around $supp_L(\varphi)$. Suitable choices for ξ can be used to balance generalization and precision.

A second possibility is to relax the probabilistic constraint even more, by not considering its actual support, but only the fact that its support is, by construction, higher than the minimum support threshold χ. This leads to replace the original constraint $\langle \varphi, supp_L(\varphi)\rangle$ by $\langle \varphi, \geq, \chi\rangle$. If the "dual variant" of the algorithm is employed, then this replacement is used whenever $supp_L(\varphi) \geq \chi$, whereas in the case where $supp_L(\varphi) \leq 1 - \chi$, the original constraint $\langle \varphi, supp_L(\varphi)\rangle$ must be replaced by $\langle \varphi, \leq, 1 - \chi\rangle$.

There are a number of open questions that arise from this new approach to discovery and its three variants, such as for example:

- how to characterize under/over-fitting considering the presence of probabilities, and the fact that discovery algorithms do not target full LTL$_f$, but focus on specific formula templates that natively introduce some form of generalization;
- how to reinterpret key notions such as vacuity and interestingness in the presence of probabilities;
- how to fine-tune the mining of probabilities by observing evolving logs and the variations they induce in their stochastic interpretation.

This is left as future investigation, based on the framework outlined in this article.

6. Monitoring probabilistic constraints

In Section 4, we presented a framework to reason on ProbDeclare models through scenarios (Definition 14), and their probability boxes (Definition 17). We now describe how this framework can be made operational into a probabilistic monitoring technique.

We consider two types of monitoring: prefix monitoring, which simply checks whether the currently monitored execution satisfies a ProbDeclare model; and full monitoring, where the monitor also considers all possible future evolutions of the current trace, in turn providing mechanisms for the early detection of violations [21,27,30].

In the technical treatment provided next, we call in several places the verification problem, which checks whether a trace τ satisfies an LTL$_f$ formula φ. Operationally, this task can be handled through the construction of the finite-state automaton $\mathcal{A}_\varphi$, and determinizing it if needed [21,27]. We can then use the resulting deterministic automaton to check whether τ satisfies φ incrementally as new activity executions are detected.

6.1. Prefix monitoring

A very direct form of monitoring consists in checking whether a partial trace, that is, the prefix of a full trace whose continuation is yet to be determined, satisfies a given ProbDeclare model $M = \langle \Sigma, \mathcal{C}, \mathcal{P}\rangle$. This amounts to a *probabilistic version of conformance checking*, which can be directly tackled by operationalizing the technique used in the proof of Theorem 3 to compute the scenario to which a trace belongs.

Let τ be the trace of a monitored execution. Our goal is to check whether τ satisfies M and, if so, to which scenario it belongs, in turn using this information to return an interesting feedback on how M classifies τ. In the next procedure, recall that $\mathbb{P}_M(S)$ is the probability box of scenario S, namely an interval delimiting the minimum and maximum probability masses that can be associated to S when defining a probability distribution over scenarios. We proceed as follows:

- We separately check τ against each crisp constraint in $\mathcal{C}$, using its local automaton. This amounts to check whether the automaton accepts τ. If there exists a constraint $\psi \in \mathcal{C}$ such that $\tau \not\models \psi$, we return VIOLATION, together with ψ as a witness.
- Following the strategy adopted in the proof of Theorem 3, we use $\mathcal{A}_\varphi$ to separately check τ against the constraint formula φ of each probabilistic constraint in $\mathcal{P}$. We use the so-obtained verdicts to calculate the scenario S_i to which τ belongs.
- If $\mathbb{P}_M(S_i) = [0, 0]$, then we return VIOLATION, together with S_i and its zero probability as a witness.
- Else, we return CONFORMING, together with S_i and its probability interval $\mathbb{P}_M(S_i)$ as a feedback. On the one hand, S_i gives an indication about how M classifies τ, and which other traces would be classified identically. On the other hand, $\mathbb{P}_M(S_i)$ gives an indication on whether τ represents a common or an outlier behavior, thus coupling conformance with an estimation of the degree of "conformism" of τ. Notably, the larger the probability interval given by $\mathbb{P}_M(S_i)$, the less confident (and less precise) this indication is.

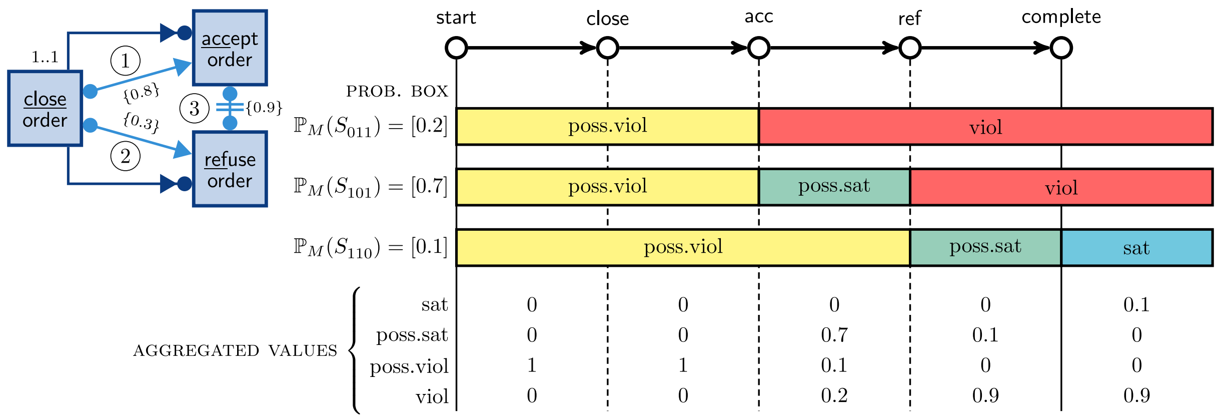
Recall that the probability boxes $\mathbb{P}_M$ do not express that all probabilities in this interval are possible, but it contains all the possible values. The result of this method tells us that the probability of observing a trace of this kind is within $\mathbb{P}_M(S_i)$, but makes no further claims about this value.

Prefix monitoring can be performed very efficiently, but comes with a main limitation: it does not reason on the possible future continuations of the current trace, and so cannot provide any insight on how the monitoring judgment may change once the trace is extended.

6.2. Full monitoring

We now show how prefix monitoring can be further developed into full monitoring of prefixes and their possible continuations in our probabilistic setting. In this case, we cannot consider the constraints in isolation anymore, but must reason at the level of scenarios.

As a preliminary, pre-processing step, we discard all inconsistent scenarios, along with all scenarios with zero-probability. For each consistent, non-zero-probability scenario S_i, we compute its characteristic formula $\Phi(S_i)$ as in Definition 15. Since this formula is in LTL$_f$, we compute its automaton $\mathcal{A}_{\Phi}(S_i)$ and determinize it. Then, we decorate this automaton turning it into a so-called colored automaton, through the well-known monitor construction techniques from the literature [21,27]. We call the resulting automaton a *scenario monitor*. For an LTL$_f$ formula φ over a set Σ of activities, and a partial trace τ representing an ongoing process execution, a colored automaton outputs one of the four following truth values, in agreement with the RV-LTL semantics for runtime verification [31]:



sat	0	0	0	0	0.1
poss.sat	0	0	0.7	0.1	0
poss.viol	1	1	0.1	0	0
viol	0	0	0.2	0.9	0.9

Fig. 3. Result computed by monitoring the ProbDeclare model on the top left against the trace ⟨close, acc, ref⟩, which conforms to the outlier constraint scenario where the two `response` constraints are satisfied, while the `not-coexistence` one is violated.

- τ *(permanently) satisfies* φ, if φ is currently satisfied ($\tau \models \varphi$), and φ stays satisfied no matter how the execution continues, that is, for every possible continuation trace τ' over Σ, we have $\tau \cdot \tau' \models \varphi$ (the $\cdot$ operator denotes the concatenation of two traces);
- τ *possibly satisfies* φ, if φ is currently satisfied ($\tau \models \varphi$), but φ may become violated in the future, that is, there exists a continuation trace τ' over Σ such that $\tau \cdot \tau' \not\models \varphi$;
- τ *possibly violates* φ, if φ is currently violated ($\tau \not\models \varphi$), but φ may become satisfied in the future, that is, there exists a continuation trace τ' over Σ such that $\tau \cdot \tau' \models \varphi$;
- τ *(permanently) violates* φ, if φ is currently violated ($\tau \not\models \varphi$), and φ stays violated no matter how the execution continues, that is, for every possible continuation trace τ' over Σ, we have $\tau \cdot \tau' \not\models \varphi$.

At runtime, we track the evolution of a running trace by delivering its activity occurrences to *all* the scenario monitors in parallel, fetching the truth values they produce as output. As pointed out in Section 6.1, at runtime we do not know to which scenario the current trace will belong to once the trace is extended with new activity occurrences. The overall feedback can then be generically interpreted as a sort of "superposition" of monitoring states. In addition, we can exploit in combination scenarios, probability intervals, and monitoring states to return a meaningful feedback. We comment on some of these advanced capabilities next.

For every partial trace, at most one scenario can turn out to be permanently or possibly satisfied. This is a direct consequence of Theorem 3. In fact, if the execution would be completed now, that scenario would turn out to be permanently satisfied, witnessing that the observed trace belongs to it (and thus, by Theorem 3, does not belong to any other scenario). Call this scenario S_{ok}.

- If S_{ok} is permanently satisfied, the verdict is irrevocable, and also implies that all other scenarios are and will for sure be permanently violated. This in turn witnesses that no matter how the execution continues, the resulting trace will necessarily belong to S_{ok}. We can then return CONFORMING, together with S_{ok} and its probability interval $\mathbb{P}_M(S_{ok})$ (exactly like in the case of prefix monitoring).
- If S_{ok} is temporarily satisfied, the verdict may instead change as the execution unfolds, but would collapse to the previous case if the execution terminates, which is communicated to the monitors by a special *complete* event.

In contrast to the cases of permanent and temporary satisfaction, multiple scenarios may be at the same time temporarily or permanently violated. For this reason, we need to aggregate the probability intervals of all those scenarios associated to the same monitoring outcome, to have an indication of the overall probability associated with that outcome. This is done by lifting the notion of scenario probability box to the case of sets of scenarios. In this case, the collective probability box is obtained by computing the two endpoints are via two optimization problems over $\mathcal{L}_M$, respectively minimizing and maximizing the sum of probability variables associated to the scenarios in the set. Just as for the probability boxes for individual scenarios, these bounds do not express that every value within the interval is a possible probability value, but does express that any value beyond them is impossible.

For temporarily violated scenarios, we return the aggregated probability interval computed as explained before. For permanently violated scenarios, we can go beyond that. On the one hand, we notice that the probability interval computed for permanently violated scenarios can never shrink over time, but only remain unaltered or grow. In fact, new scenarios may become permanently violated, but those that are already permanently violated will stay so forever. Consequently, a high aggregated probability mass associated to permanent violation can be interpreted as a clear indication that the monitored trace will turn out to be either a conforming outlier, or not conforming at all.

Another way to interpret the aggregated contribution of permanently violated scenarios is as in terms of posterior probability. All the scenarios that are permanently violated can in fact be considered *impossible* by the probabilistic model, in turn calling for rescaling the probability intervals attached to the other scenarios one the permanently violated ones are excluded. For any probability distribution in our model, if p_{ts}, p_{tv}, and p_{pv} represent the probabilities of the scenarios that are temporarily satisfied, temporarily violated, and permanently violated, respectively, as described above, we update the former two to: $p'_{ts} := p_{ts}/(1-p_{pv})$ and $p'_{tv} := p_{tv}/(1-p_{pv})$. Likewise for probability intervals.

Example 21. Consider the ProbDeclare model in Fig. 1 with its three consistent scenarios with non-zero probability (the contribution of scenario S_{001} is in fact irrelevant). Fig. 3 shows the result produced when monitoring a trace that at some point appears to belong to the most likely scenario, but in the end turns out to conform to the least likely one.

We briefly comment on the evolution of the trace and its corresponding monitors. When the trace starts, all scenarios are possibly violated, as they expect close to be executed. There is in this situation no uncertainty about which monitoring state has to be returned. Once close is executed, the situation stays unaltered, as both `response` constraints now require a consequent execution of acc and/or ref. The monitor returns a different picture when the third activity is processed, namely acc. In fact, scenario S_{011} becomes permanently violated, as the combination of the `existence`(close) and the negated version of `response`(close,acc) imposed therein that no occurrence of acc would happen. Differently, scenario S_{101} becomes possibly satisfied, as the `response`(close,acc) constraint is now satisfied. The satisfaction is possible as we cannot guarantee that the `not coexistence`(acc,ref) constraint, which must be true in S_{101}, will stay so. If the execution would stop now, then we would have that the trace collected so far actually belongs to scenario S_{101} (as it would become the only permanently satisfied). This would indicate that the monitored execution belongs to the most likely scenario, encountered in 70% of the cases.

However, the situation changes as the execution unfolds: `not coexistence`(acc,ref) becomes permanently violated when ref is later executed, in turn causing S_{101} to become permanently violated as well. Upon the execution of this activity, scenario S_{110} becomes possibly satisfied: it requires both `response` constraints to be satisfied and the `not coexistence`(acc,ref) one to be violated, which is indeed the case. Satisfaction is only possible as `existence`(close) may be violated upon a further execution of close. At this stage, it has become certain that the monitored trace does not belong to S_{011} nor S_{101}, as they are both permanently violated. There are in fact two possibilities: either the trace belongs to S_{110}, or becomes non conforming (and, thus, not part of any scenario). Since the execution terminates and the trace gets completed, the former is the case: the final trace belongs to S_{110}, a rare scenario that is seen in 10% of the cases.

From the image, we can also clearly see that the trace consisting only of a close activity would be judged as non-conforming, as it would violate all the monitored scenarios.

7. Probabilistic conformance checking

As a last process mining task, we revisit conformance checking. Probabilistic conformance checking of a single trace with respect to a ProbDeclare model can be fully tackled using the prefix monitoring technique introduced in Section 6.1. In this section, we concentrate on a different approach: conformance checking of a whole log with respect to a ProbDeclare model, in the spirit of an ongoing line of research that is investigating this problem by adopting procedural, probabilistic models based on variants of stochastic Petri nets [19,32–36].

In particular, we take inspiration from the approach in [19], which studies how the well-established notion of Wasserstein distance [37] (also called *earth mover's distance* or EMD for short) can be adapted to measure the distance between a log and a stochastic Petri net.

In general terms, given two discrete distributions of elements, the EMD between these distributions is computed by combining two distinct distances:

- an *element distance* measuring the pairwise distance between elements from the first distribution and elements from the second distribution, considering the elements as such, and not the probability mass they carry;
- a *reallocation distance* measuring the "effort" required to move the probability mass carried by an element from the first distribution to an element of the second distribution, with the overall goal of transforming one distribution into the other.

A key issue arising when applying EMD in the context of conformance checking, is to decide what are the elements to be compared, on the log side and on the model side. In [19], this is done:

- on the log side by transforming the input log into a stochastic language (with the approach that we reconstruct in Definition 6), and by considering as elements the traces of the stochastic language with their probability masses;
- on the model side, by explicitly enumerating a finite portion of all the probabilistic traces from the stochastic Petri net.

While similar in spirit, our approach comes with a radical difference with respect to that in [12]: while they explicitly consider traces as elements, we implicitly fold traces to be compared into scenarios. This is based on the observation that a ProbDeclare model is not able to classify traces at a more granular level than that of scenarios. At the same time, it brings the advantage that both on the log and on the model side we have to consider *boundedly many elements* (that is, scenarios), without the need of approximating or truncating the elements. If one is interested to inject into the approach a more fine-grained analysis of the log at the single trace level with ad-hoc distance measures, alternative approaches have to be investigated. We comment on this in Section 7.4.

In the remainder of the section, we first discuss how to measure the distance between two scenarios. We then describe how a log can be seen as a probability distribution over scenarios. Finally, we combine these notions, introducing a notion of EMD for ProbDeclare.

7.1. Scenario distance

Consider two scenarios of the same ProbDeclare model M. We measure their distance by applying the most intuitive approach, that is, on the basis of how many constraints they disagree on. Given the binary representation of the two scenarios, this corresponds to the number of flips that, component-wise, have to be applied so as to transform one scenario into the other. Let $\oplus$ denote the exclusive or (XOR) of two bits. Then, we can formally capture this intuition as follows.

Definition 19. Let $S_{b_1\cdots b_n}$ and $S_{d_1\cdots d_n}$ be two scenarios with the same number of components. Then the *(normalized) bit-flipping distance* between $S_{b_1\cdots b_n}$ and $S_{d_1\cdots d_n}$ is a number in [0, 1] defined as follows:

$$fd(S_{b_1\cdots b_n}, S_{d_1\cdots d_n}) = \frac{\sum_{i=1}^{n}(b_i \oplus d_i)}{n}$$

Example 22. Consider two scenarios S_{011} and S_{101}. We have:

- $fd(S_{011}, S_{011}) = 0$;
- $fd(S_{011}, S_{101}) = fd(S_{101}, S_{011}) = 2/3$.

7.2. Log-induced probability distributions

As indicated before, instead of unfolding the model into traces, we want to fold the log into scenarios. The idea is to compute the probability distribution that the traces in the log, together with their respective frequencies, induce over the scenarios of the model of interest.

This simply amounts to collect, scenario by scenario, the overall relative frequency of all traces that belong to a scenario, and use it to assign a probability to that scenario.

Definition 20. Let L be an event log, and M be a ProbDeclare model. The *probability distribution induced by L over M* is the function $\mathbb{F}_L^M$ that maps scenarios of M into values from [0, 1] as follows: for every scenario S_i of M, we have that

$$\mathbb{F}_L^M(S_i) = \frac{\sum_{\tau\in L,\ \tau \text{ belongs to } S_i} L(\tau)}{|L|}$$

It is easy to prove that $\mathbb{F}_L^M$ is indeed a probability distribution.

Example 23. Consider again the ProbDeclare model M in Fig. 1, and the log $L = [\langle \text{close}, \text{acc}\rangle^1, \langle \text{close}, \text{ref}\rangle^4, \langle \text{close}, \text{acc}, \text{ref}\rangle^2, \langle \text{close}, \text{ref}, \text{acc}\rangle^3]$. The trace $\langle \text{close}, \text{acc}\rangle$ is the only one in L that belongs to scenario S_{101}, consequently giving $\mathbb{F}_L^M(S_{101}) = 1/10 = 0.1$. Similarly, the trace $\langle \text{close}, \text{ref}\rangle$ is the only one belonging to the scenario S_{011}, consequently giving $\mathbb{F}_L^M(S_{011}) = 4/10 = 0.4$. Finally, the two remaining traces belong to S_{110}, as they represent two different ways to change decision about a closed order. Hence, we obtain the probability $\mathbb{F}_L^M(S_{110}) = (2+3)/10 = 0.5$.

7.3. Earth mover's distance for ProbDeclare

We are now ready to define our notion of EMD. To do so, we create a system of inequalities that expresses constraints on the amount of probability mass that has to be reallocated, in order to agree with one of the probability distributions associated to scenarios, and to the probability distribution induced by the log over such scenarios. In doing so, there are two main aspects to consider: first, that there may be multiple (possibly, infinitely many) probability distributions induced by the ProbDeclare model over its scenarios; second, that there are in principle several different ways to reallocate probability masses. Hence, we seek for the optimal reallocation strategy, minimizing at once over the probability distributions induced by the model, and over the different reallocation values for scenarios. Minimization is defined over the overall reallocation cost, where each reallocation from a scenario to another scenario is weighted by the bit-flipping distance between the two scenarios.

Specifically, given a ProbDeclare model M with n probabilistic constraints, and given a log L, we use variables $x_0, \ldots, x_{2^n-1}$ to indicate the probabilities assigned to the scenarios of M, compatibly with the probability box of M; in addition, we use variable $r_{i,j}$ to indicate the reallocation of the probability induced by L over scenario S_i to the probability x_i assigned to scenario S_j compatibly with the system of inequality $\mathcal{L}_M$. We then set up the system of inequalities $\mathcal{R}_{M,L}$ as follows:

$$\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^n-1} fd(S_i, S_j)\cdot r_{i,j} = cost \tag{7}$$

$$\sum_{j=0}^{2^n-1} r_{i,j} = \mathbb{F}_L^M(S_i) \qquad 0 \le i < 2^n \tag{8}$$

$$\sum_{i=0}^{2^n-1} r_{i,j} = x_j \qquad 0 \le j < 2^n \tag{9}$$

$$\mathcal{L}_M \tag{10}$$

Line (7) introduces the overall cost incurred in moving the probability masses, defined as the linear combination of each reallocation multiplied by the bit-flipping distance between the involved scenarios. Line (8) expresses the constraints on reallocations induced by the log, indicating that, for each scenario S_i, the overall reallocation variables involving S_i must collectively yield exactly the probability mass induced by L on that scenario. Similarly, line (9) expresses the constraints on reallocations induced by the model, indicating that, for each scenario S_i, the overall reallocation variables involving S_i must collectively yield exactly the probability mass assigned to that scenario compatibly with the probability distributions induced by M over scenarios. The next line (10) expresses precisely such a compatibility, including in the system of inequalities those in $\mathcal{L}_M$, which capture how the constraint probability conditions from M induce corresponding constraints on the probability masses of the different scenarios.

We are now finally ready to define the overall notion of distance.

Definition 21. The *earth mover's distance (EMD)* between a ProbDeclare model M and a log L is:

$$EMD(M, L) = 1 - cost^*,$$

where $cost^*$ is the solution of the optimization problem that minimizes variable $cost$ in the system of inequalities $\mathcal{R}_{M,L}$.

Obviously, when actually computing the earth mover's distance we can immediately remove from the system of inequalities all those entries that refer to scenarios with zero probability, as they would not contribute at all to the distance. In addition, in case $\mathcal{L}_M$ assigns a single probability mass to each scenario, we can directly use such values in place of the x_j variables, completely omitting Eqs. (10).

Example 24. Consider the ProbDeclare model M in Fig. 1, and the log L from Example 23. We now compute the EMD between them. Even before entering into the actual calculation, we can already observe that only three scenarios are relevant for considerations, namely S_{011}, S_{101}, and S_{110}, since they are the only ones associated to a non-zero probability by $\mathcal{L}_M$. In addition, we can immediately estimate that the distance will be quite sensible: from the model side, $\mathcal{L}_M$ indicates S_{101}, S_{011}, and S_{110} by decreasing likelihood, whereas from the log side, $\mathbb{F}_L^M$ indicates the completely reversed ranking.

To instantiate the system of inequalities $\mathcal{R}_{M,L}$, we can follow the intuition provided in the following two tables.

bit-flipping distances

	S_{011}	S_{101}	S_{110}
S_{011}	0	2/3	2/3
S_{101}	2/3	0	2/3
S_{110}	2/3	2/3	0

reallocations

	x_{011}	x_{101}	x_{110}	
$\mathbb{F}_L^M(S_{011})$	$r_{011,011}$	$r_{011,101}$	$r_{011,110}$	0.4
$\mathbb{F}_L^M(S_{101})$	$r_{101,011}$	$r_{101,101}$	$r_{101,110}$	0.1
$\mathbb{F}_L^M(S_{110})$	$r_{110,011}$	$r_{110,101}$	$r_{110,110}$	0.5
	0.2	0.7	0.1	

We obtain:

$$\begin{array}{rcrcrcl}
0\cdot r_{011,011} & + & 2/3\cdot r_{011,101} & + & 2/3\cdot r_{011,110} & & \\
+\ 2/3\cdot r_{101,011} & + & 0\cdot r_{101,101} & + & 2/3\cdot r_{101,110} & & \\
+\ 2/3\cdot r_{110,011} & + & 2/3\cdot r_{110,101} & + & 0\cdot r_{110,110} & = & cost \\
r_{011,011} & + & r_{011,101} & + & r_{011,110} & = & 0.4 \\
r_{101,011} & + & r_{101,101} & + & r_{101,110} & = & 0.1 \\
r_{110,011} & + & r_{110,101} & + & r_{110,110} & = & 0.5 \\
r_{011,011} & + & r_{101,011} & + & r_{110,011} & = & 0.2 \\
r_{011,101} & + & r_{101,101} & + & r_{110,101} & = & 0.7 \\
r_{011,110} & + & r_{101,110} & + & r_{110,110} & = & 0.1
\end{array}$$

By minimizing $cost$, we get solution 0.4. Hence:

$$EMD(M, L) = 1 - 0.4 = 0.6$$

7.4. An alternative EMD based on alignments

The EMD here presented relies on the observation that, when comparing a trace and an LTL$_f$ formula, what matters is whether the satisfies or violates the formula, not "how" and "why" this is the case. In our approach, this declarative approach leads to blurring the distinction between traces and scenarios. In fact, a scenario only distinguishes those traces that belong from those that do not, and does not provide any further finer-grained considerations on how violating traces differ to each other. This has the effect that two radically different logs may lead to the same EMD value when compared to a ProbDeclare model M if the traces they contain, albeit different, relate in the same way to the scenarios induced by M.

An alternative approach is to go beyond this coarse-grained, boolean approach, moving toward a more refined approach where, in case of violation, some estimation about the entity/degree of violation is used. This is commonly captured, in the conformance checking literature, by the notion of *alignment*, extensively studied in the literature [18] also in the case of declarative process models based on Declare [38]. For example, the EMD distance studied in [19] indeed combines reallocation distances on probabilities with alignment distances between each log trace and the stochastic Petri net.

In a nutshell, alignments provide an indication about the difference between an observed trace and a model trace, typically employing variants of the Levenshtein distance, e.g., counting how many insertions and deletions have to be applied to make the two traces equal. The alignment distance between an observed trace and a whole model is then computed as the smallest possible distance between the observed trace and the model traces.

In the context of our EMD, instead of relating a trace τ to a scenario S by first obtaining the scenario S' to which τ belongs, then comparing S to S' via the notion of bit-flipping distance (cf. Definition 19), one could employ the alignment distance between τ and S instead. This can be done, e.g., by computing the deterministic automaton $\mathcal{A}_{\Phi_S}$ for the characteristic formula of S (cf. Definition 15), then using the technique in [38]. Notice that this notion of distance is not a number in $[0, 1]$. However, we can easily normalize it against the worst possible alignment between τ of S, which is one where τ has nothing in common with the traces belonging to S; if this happens, then the alignment distance is $|\tau| + m_S$, where m_S is the length of the shortest trace (in this case, the best to select) belonging to S.

Example 25. Consider the trace $\tau = \langle \mathsf{close}, \mathsf{acc} \rangle$ from Example 23, and the three relevant scenarios S_{011}, S_{101}, and S_{110} from Fig. 1 and Example 24. Recall that τ belongs to scenario S_{101}. Hence, the bit-flipping distances are 0 with S_{101} itself, and 2/3 with both S_{011} and S_{110}.

Consider instead the alignment distance between τ and the three scenarios. Since τ belongs to S_{101}, the alignment distance is in this case 0, as τ gets compared to itself. In the case of S_{110}, instead, the (model) trace from S_{110} that is closest to τ is $\tau' = \langle \mathsf{close}, \mathsf{acc}, \mathsf{ref} \rangle$; this yields an alignment distance of 1, considering that by adding ref at the end of τ we get τ'. To normalize this distance, we note that the shortest trace length for S_{110} is actually 3 (being τ' one of the shortest traces in S_{110}). Considering that the length of τ is 2, we then get a distance of 1/5. Finally, in the case S_{011}, we have that the closest trace in S_{011} is $\tau'' = \langle \mathsf{close}, \mathsf{ref} \rangle$, which is also the shortest one. The alignment distance between τ and τ'' is 2, as one needs to remove acc and insert ref to turn τ into τ''. Hence, the normalized distance is 2/4.

Apart from the fact that the distance values are numerically different when moving from flipping to alignment distance, what is more important to notice is that while the bit-flipping distance considers scenarios S_{011} and S_{110} as being equally distant from τ, this is not the case for the alignment distance.

All in all, if one is interested in differentiating traces that violate a scenario based on "how close" they are to traces belonging to that scenario, the bit-flipping distance-based EMD can be replaced with a different EMD, where the element distance used therein relies on the normalized alignment distance instead of the (scenario-based) bit-flipping distance. An in-depth study of this aspect is matter of future work.

8. Evaluation

In the following, we present the evaluation results for the three main approaches introduced in this paper (process discovery, conformance checking, and monitoring). All parts of the evaluation are based on the BPIC2018 event log [20], which is a real life event log pertaining to the process of handling applications for EU direct payments for German farmers from the European Agricultural Guarantee Fund. For our analysis, we split the log into 3 sub-logs consisting of all the cases with initial event occurring in 2015, 2016, and 2017, respectively. The general statistics for these 3 sub-logs are shown in Table 2. We decided to split the log in this way, because a specific part of the process was redesigned starting from 2016 and another redesign of the same part followed starting from 2017. In particular, in 2016, the document Parcel document was replaced by the document Geo Parcel Document, and in 2017, the document Department Control Parcels document was also replaced by Geo Parcel Document. During both redesigns, all the related activities also changed, thus allowing us to get different discovery results per year and to compare a model discovered from one sub-log with the other sub-logs and identify some discrepancies.

Table 2
General statistics for the BPIC2018 sub-logs.

	2015	2016	2017
Activities	152	145	102
Events	897 678	763 710	852 610
Cases	14 746	14 550	14 507
Variants	12 244	8 655	8 020

The evaluation is aimed at answering the following research questions:

- Process discovery

RQ1.1 How does the size of a discovered probabilistic model vary when changing the minimum probability allowed for the constraints in the model?
RQ1.2 How does the performance of the discovery task vary when changing the minimum probability allowed for the constraints in the model?

- Conformance checking

RQ2.1 How sensitive is the EMD measure for different constraint probability distributions in the reference model?
RQ2.2 How does the computation of the EMD measure scale when the size of the reference model increases?
RQ2.3 How does the computation of the EMD measure scale when the noise in the log increases?
RQ2.4 Is the EMD measure capable of detecting conformance/non-conformance in a real life event log?

- Monitoring

RQ3.1 How does the monitoring approach scale when the size of the reference model increases?
RQ3.2 Is the scaling of the monitoring approach effected by the ratio of consistent/inconsistent scenarios?

8.1. Process discovery

All process discovery tests were performed using the Declare Miner [10] in RuM [39]. The templates used for the discovery are the ones shown in Table 1. We considered only constraints over activities occurring in at least 10% of the cases, and we filtered out redundant [11] and vacuously satisfied [28] constraints.

Table 3 shows the number of discovered constraints for different values of minimum constraint support, i.e., the minimum percentage of cases in which a discovered constraint must be (non-vacuously) satisfied. In general, the number of discovered constraints increases when the minimum constraint support decreases. However, this is not always the case since, sometimes, decreasing the minimum constraint support implies a higher

Table 3
Process discovery results for cases starting in 2015, 2016, and 2017. Model sizes are given in number of discovered constraints. Process discovery times are given in milliseconds.

Min. Supp.	2015		2016		2017	
	Model Size	Time	Model Size	Time	Model Size	Time
100	20	136 568	44	132 482	48	93 229
90	107	141 607	90	131 715	88	95 678
80	106	143 252	86	130 673	91	96 318
70	110	140 688	86	130 043	91	95 737
60	116	141 079	91	130 523	89	95 529
50	126	143 743	100	132 659	88	94 283
40	141	142 667	105	132 954	97	95 468
30	141	144 505	107	131 449	99	95 679
20	150	146 350	115	131 993	106	95 401
10	195	150 678	143	136 280	115	97 577

Table 4
EMD measures at different levels of probability miss-matches between the reference model and the event log. All given constraints are always fulfilled in the given event log (probability of 1.0 in the event log), but the probabilities in the model are changed for each test. Conformance checking times are given in milliseconds.

Scenario	Constraint probability distribution					EMD	Time
	0%	25%	50%	75%	100%		
1					5	1	9 979
2				1	4	0.95	10 187
3				2	3	0.9	10 505
4				3	2	0.8	10 237
5				4	1	0.65	10 587
6				5		0.55	9 966
7			1	4		0.525	10 257
8			2	3		0.525	10 175
9			3	2		0.525	9 486
10			4	1		0.525	9 416
11			5			0.5	10 027
12		1	4			0.475	10 869
13		2	3			0.475	10 202
14		3	2			0.475	10 215
15		4	1			0.475	9 577
16		5				0.45	9 832
17	1	4				0.35	9 429
18	2	3				0.2	10 098
19	3	2				0.1	10 433
20	4	1				0.05	10 581
21	5					0	9 987

number of redundant constraints that can be removed (**RQ1.1**). Interestingly, the minimum constraint support represents, in the probabilistic context, the minimum probability of all the constraints discovered with that minimum constraint support value, and the support of every single discovered constraint can be used as the satisfaction probability of that constraint in the input log. In the same table, we also report the execution times in milliseconds, which clearly depend on the number of activities available in the sub-logs (the higher the number of activities, the higher the execution time). This is due to the fact that a higher number of activities means more candidate constraints to be checked in the discovery task (**RQ1.2**).

8.2. Conformance checking

Conformance checking was carried on with the prototype available at https://bitbucket.org/fmmaggi/probabilisticmonitor/src/master/. First, we evaluated the sensitivity of the EMD measure by analyzing how the value of this measure changes when changing the probabilities of the constraints in the reference model used in the conformance checking task. Second, we used constraints discovered from the 2015 sub-log of the BPIC2018 event log to evaluate the performance of the approach by checking their conformance against the 2016 and the 2017 sub-logs.

8.2.1. EMD sensitivity

We used, as reference model, a set of 5 constraints satisfied in all cases of the 2015 sub-log so that all potential scenarios derived from this set of constraints would be consistent. Consistency of scenarios is important in this case, because it allows us to modify the probabilities of the selected constraints manually without running the risk of creating an inconsistent model.[5]

Then, we manually set the probability of one constraint to 0.75 in the reference model, thus causing a miss-match between the model and the log (we always use the 2015 sub-log for these experiments). Then, we set two constraints to 0.75 and so on. This procedure gives us a better understanding of how the EMD measure reflects the probabilistic non-conformance between a model and an event log.

The results are shown in Table 4. Here, we can see that the endpoints of the conformance/non-conformance spectrum are reflected in the EMD measure quite well (tests 1–6 and 16–21). However, in the middle of the conformance/non-conformance spectrum multiple tests result in the same EMD measure (tests 7–10 and 11–15), despite the differences in constraint probability distributions. Therefore, we can conclude that the sensitivity of the EMD is maximum when the constraints in the reference model have very high or very low probabilities (**RQ2.1**).

8.2.2. EMD performance

Given the overall high number of constraints discovered from the BPIC2018 sub-logs and the resulting high number of potential scenarios,[6] we decided to select a smaller sub-set of constraints to use in the EMD performance evaluation. More specifically, we used the model discovered from the 2015 sub-log (with minimum constraint support of 90, corresponding to constraints with probability of at least 90%). This model was then compared against the 2016 sub-log, in order to identify relevant sub-sets of constraints to use. We identified two sets of 9 constraints that were the most fulfilled and the most violated ones (when considered as crisp constraints) respectively.[7]

These probabilistic models (defined by combining the discovered constraints with their support as probability) were then used to run the conformance checking task against the 2016 and the 2017 sub-logs. Furthermore, to estimate the scalability of the approach, we ran each test with an increasing number of constraints. The results are shown in Tables 5–6.

First of all, we notice that the constraints with the highest number of fulfillments in the 2016 sub-log have EMD values always equal to 1 when using both the 2016 and the 2017 sub-logs, whereas the constraints with the highest number of violations in the 2016 sub-log have lower EMD values for both the 2016 and the 2017 sub-logs. This means that the probabilities of the constraints are stable across all three sub-logs (2015, 2016, and 2017) for the first sub-set of constraints, whereas for the second sub-set of constraints the probabilistic model discovered from the 2015 sub-log is neither compliant with the 2016 sub-log nor with the 2017 sub-log. This scenario shows that the EMD-based conformance checking, when applied to a real life

[5] An example of a model that is inconsistent because of probabilities is exactly1(a) - 1.0 and absence2(a) - 0.5, because if exactly1(a) is always satisfied then absence2(a) must also be always satisfied and cannot therefore have a probability of 0.5.

[6] Number of potential scenarios to be checked for consistency is 2^n, where n is the number of probabilistic constraints.

[7] To do this we used the Declare Analyzer [40] in RuM.

Table 5
Constraints from 2015 (min. support 90) that are most fulfilled in 2016 compared against 2016 and 2017 sub-logs. Conformance checking times are given in milliseconds.

Model Size	Scenarios		Year 2016		Year 2017	
	Total	Consistent	EMD	Time	EMD	Time
1	2	2	1	7 559	1	7 393
2	4	4	1	8 013	1	7 984
3	8	8	1	7 733	1	8 234
4	16	13	1	7 890	1	8 504
5	32	21	1	9 199	1	9 410
6	64	34	1	9 584	1	10 729
7	128	55	1	10 113	1	12 336
8	256	89	1	15 451	1	20 498
9	512	131	1	121 639	1	138 779

Table 6
Constraints from 2015 (min. support 90) that are most violated in 2016 compared against 2016 and 2017 sub-logs. Conformance checking times are given in milliseconds.

Model Size	Scenarios		Year 2016		Year 2017	
	Total	Consistent	EMD	Time	EMD	Time
1	2	2	0.02	7694	0.01	7 767
2	4	4	0.025	7341	0.03	7 817
3	8	8	0.045	7711	0.04	8 144
4	16	16	0.06	8335	0.055	8 434
5	32	32	0.05	8609	0.04	8 769
6	64	64	0.04	9666	0.035	11 067
7	128	128	0.035	12 501	0.035	12 737
8	256	256	0.03	28 403	0.155	24 974
9	512	512	0.045	124 429	0.25	105 251

Table 7
Constraints from 2015 (min. support 90) that are fulfilled the most in 2016 monitored against first 100 traces in 2016 and 2017 sub-logs. Monitoring times are given in milliseconds.

Model Size	Processing times (year 2016)			Processing times (year 2017)		
	Model	Event avg.	Trace avg.	Model	Event avg.	Trace avg.
1	71	0.05	2.41	69	0.05	2.81
2	87	0.04	2.23	86	0.05	2.79
3	115	0.03	1.80	114	0.05	2.79
4	165	0.03	1.82	168	0.05	2.77
5	261	0.04	2.34	297	0.05	2.61
6	492	0.04	2.33	536	0.05	3.00
7	1 152	0.05	2.56	1 181	0.04	2.52
8	2 974	0.05	2.40	2 902	0.05	2.74
9	12 987	0.04	2.57	12 890	0.05	2.79

Table 8
Constraints from 2015 (min. support 90) that are violated the most in 2016 monitored against first 100 traces in 2016 and 2017 sub-logs. Monitoring times are given in milliseconds.

Model Size	Processing times (year 2016)			Processing times (year 2017)		
	Model	Event avg.	Trace avg.	Model	Event avg.	Trace avg.
1	84	0.06	3.37	70	0.05	2.67
2	85	2.74	140.71	83	2.58	146.76
3	115	2.86	146.87	114	2.68	152.49
4	180	2.98	153.16	178	2.81	159.48
5	313	3.14	161.71	312	3.28	186.13
6	671	3.47	178.38	672	3.60	204.45
7	1 581	4.68	240.46	1 782	4.98	282.40
8	4 929	7.68	394.70	4 867	7.81	443.32
9	16 405	18.00	923.02	16 491	18.34	1041.13

event log (with known misconformancies), can successfully detect both conformance and non-conformance in a probabilistic setting (**RQ2.4**).

We can also notice that the number of inconsistent scenarios strongly depends on the structure of the reference model. More specifically, if the reference model contains constraints that interact,[8] with each other (in this specific case, this happens for the model containing constraints with the highest number of fulfillments), then the model is also more likely to have inconsistent scenarios.

Finally, the EMD-based conformance checking has a reasonably good performance but a noticeable slowdown occurs after 8 constraints. As expected, the execution times increase when the size of the reference model increases (**RQ2.2**). However, the time performance is not affected by the amount of noise present in the log, i.e., by the amount of discrepancies between the model and the log. Indeed, the execution times needed for executing the conformance checking task when values of the EMD measure are high (most fulfilled constraints) and low (most violated constraints) are comparable (**RQ2.3**).

8.3. Monitoring

The evaluation of monitoring was performed analogously to the EMD performance evaluation. Exactly the same models were used (sets of 9 constraints discovered from 2015 that were most fulfilled, and most violated in 2016) and the performance was evaluated using both the 2016 and the 2017 sub-logs. For all the tests, we report the average event and trace processing times and also the model pre-processing time. The latter refers to computation steps that are performed before processing any of the events being monitored (e.g., checking the consistency of potential scenarios), and therefore is unaffected by the size of the event log itself. The results of the monitoring evaluation are shown in Tables 7–8.

Overall, events and traces are processed near-instantaneously, especially in the case of the most fulfilled constraints, while the model preprocessing times ramp up at around 8 or 9 constraints (**RQ3.1**). However, there is a noticeable difference in performance between the most fulfilled (Table 7) and the most violated (Table 8) constraints, with the latter being slower overall. As shown in Section 8.2.2, this is due to the fact that all potential scenarios in the set of the most violated constraints are consistent and therefore need to be considered during monitoring. Instead, the set of the most fulfilled constraints has significantly less consistent scenarios, leading to a better overall performance. This shows that the number of consistent scenarios (and therefore also the structure of the probabilistic reference model) has a noticeable effect on the monitoring performance (**RQ3.2**).

9. Related work

As we already mentioned in the introduction, it is surprising that only very few process mining approaches incorporate uncertainty as a first-class citizen. Recently, uncertainty has been considered in (procedural) process mining, mainly in the context of approaches for stochastic conformance checking [12,32–35].

In [12], the authors propose a conformance measure that considers the stochastic characteristics of both the event log and the process model. The measure is based on the EMD and measures the effort to transform the distributions of traces of the event log into the distribution of traces of the process model. We take inspiration from this contribution when defining our

[8] Consider, for example, two models, the first consisting of existence(a) and existence(b), and the second consisting of existence(a) and response(a,b). The constraints in the first model involve different activities and, therefore, do not 'interact' with each other, resulting in all potential scenarios being consistent. However, both constraints in the second model involve the same activity a, and therefore do 'interact' with each other, which may result in some potential scenarios being inconsistent. In particular, both existence(a) and response(a,b) cannot be simultaneously violated.

conformance measure based on the EMD that we use to check the conformance of an event log with respect to a ProbDeclare model.

In [32], the authors extend the standard precision and recall conformance measures between process models and event logs with the support of partially matching processes. In addition, they introduce the desired properties that conformance measures supporting partially matching processes should fulfill and show that the presented measures fulfill them.

In [33], the authors propose precision and recall conformance measures based on the notion of entropy of stochastic automata that are capable of quantifying frequent and rare deviations between an event log and a process model.

In [34], the authors present an entropic relevance measure for stochastic conformance checking, computed as the average number of bits required to compress each log trace, based on the structure and information about relative likelihoods provided by the model. The measure penalizes traces from the event log not captured by the model and traces described by the model but not present in the event log, thus addressing both precision and recall quality criteria at the same time.

Standard measures to describe the quality of a process specification automatically discovered from execution logs neither fulfill essential properties, such as monotonicity, nor can they handle infinite behavior. In [35], the authors address this research problem by introducing a framework for the definition of behavioral quotients. They prove that corresponding quotients guarantee desired properties that existing measures have failed to support. They use quotients for capturing precision and recall measures between a collection of recorded executions and a system specification.

All the mentioned approaches are limited to (procedural) conformance checking whereas, in this paper, we propose a holistic framework that allows us to interpret all the branches of process mining based on declarative models from the probabilistic point of view.

10. Conclusions

In this paper, we investigate the influence of probabilistic constraints on the state-of-the-art techniques for declarative process mining. In particular, we introduce a holistic framework providing a rigorous formalization of how the standard declarative process mining approaches can be interpreted when replacing standard, certain constraints with their probabilistic counterparts.

We first define the semantics of probabilistic constraints and show how probabilistic constraints can be used to naturally lift the Declare language to its probabilistic version ProbDeclare. We observe that probabilistic Declare constraints can be discovered off-the-shelf using already existing techniques for declarative process discovery. Then, we study how to monitor probabilistic constraints and we show how conformance checking can be handled providing a notion of earth mover's distance that can be used to represent the degree of conformance of a log with respect to a ProbDeclare model.

In the future, we would like to experiment the presented framework in practical case studies and deeply investigate the implications that probabilistic constraints bring in the context of process mining analysis and how the richer feedback they provide to the user can be used in different application domains. Of particular interest is to understand how the plethora of process mining techniques presented here (including the different choices for discovery and earth mover's distance) could be combined with a filtering layer on the produced results, toward returning a more intuitive output for end users.

We are also investigating the consequences that taking into consideration stochastic aspects brings in the context of the wide range of existing procedural process mining techniques.

Finally, we would like to investigate the interplay of the probabilistic perspective in the contexts of standard multi-perspective process mining frameworks where not only control flow is considered, but also metric time and data.

Declaration of competing interest

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

Acknowledgments

This work has been partially supported by the Estonian Research Council (project PRG1226), by the UNIBZ, Italy projects VERBA, WineID, QUEST, CAT, and by the CHIST-ERA, France 2014–2020 Project PACMEL.

References

[1] S.W. Sadiq, W. Sadiq, M.E. Orlowska, Pockets of flexibility in workflow specification, in: H.S. Kunii, S. Jajodia, A. Sølvberg (Eds.), Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001), in: LNCS, vol. 2224, Springer, 2001, pp. 513–526.

[2] M. Pesic, H. Schonenberg, W.M.P. van der Aalst, DECLARE: Full support for loosely-structured processes, in: Proceedings of the 11th IEEE International Enterprise Distributed Object Computin Conference (EDOC 2007), IEEE Computer Society, 2007.

[3] T.T. Hildebrandt, R.R. Mukkamala, Declarative event-based workflow as distributed Dynamic Condition Response Graphs, in: K. Honda, A. Mycroft (Eds.), Proceedings of the 3rd Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software (PLACES 2010), in: EPTCS, vol. 69, 2010, pp. 59–73.

[4] M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographies, ACM Trans. Web 4 (1) (2010) 3:1–3:62.

[5] M. Montali, Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach, in: LNBIP, vol. 56, Springer, 2010.

[6] G. De Giacomo, M.Y. Vardi, Linear Temporal Logic and Linear Dynamic Logic on finite traces, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), AAAI Press, 2013, pp. 854–860.

[7] M. Westergaard, C. Stahl, Leveraging super-scalarity and parallelism to provide fast declare mining without restrictions, in: M. Fauvet, B.F. van Dongen (Eds.), Proceedings of the BPM Demo sessions 2013, in: CEUR Workshop Proceedings, vol. 1021, CEUR-WS.org, 2013.

[8] C. Di Ciccio, M. Mecella, On the discovery of declarative control flows for artful processes, ACM Trans. Manag. Inf. Syst. 5 (4) (2015) 24:1–24:37.

[9] S. Schönig, A. Rogge-Solti, C. Cabanillas, S. Jablonski, J. Mendling, Efficient and customisable declarative process mining with SQL, in: Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), vol. 9694, Springer, 2016, pp. 290–305.

[10] F.M. Maggi, C. Di Ciccio, C. Di Francescomarino, T. Kala, Parallel algorithms for the automated discovery of declarative process models, Inf. Syst. 74 (Part) (2018) 136–152.

[11] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Syst. 64 (2017).

[12] S.J.J. Leemans, A.F. Syring, W.M.P. van der Aalst, Earth movers' stochastic conformance checking, in: T.T. Hildebrandt, B.F. van Dongen, M. Röglinger, J. Mendling (Eds.), Proceedings of the Business Process Management Forum 2019, in: LNBIP, vol. 360, Springer, 2019.

[13] F.M. Maggi, M. Montali, R. Peñaloza, Temporal logics over finite traces with uncertainty, in: Proceedings of the 34-th AAAI Conference on Artificial Intelligence (AAAI 2020), AAAI Press, 2020, pp. 10218–10225.

[14] F.M. Maggi, M. Montali, R. Peñaloza, A. Alman, Extending temporal business constraints with uncertainty, in: D. Fahland, C. Ghidini, J. Becker, M. Dumas (Eds.), Proceedings of the 18th International Conference on Business Process Management (BPM 2020), vol. 12168, Springer, 2020, pp. 35–54.

[15] W.M.P. van der Aalst, Process Mining - Data Science in Action, 2nd, Springer, 2016.

[16] E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Inducing declarative logic-based models from labeled traces, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), Proceedings of the 5th International Conference on Business Process Management (BPM 2007), in: LNCS, vol. 4714, Springer, 2007, pp. 344–359.

[17] C. Di Ciccio, F.M. Maggi, J. Mendling, Efficient discovery of target-branched declare constraints, Inf. Syst. 56 (2016) 258–283.

[18] J. Carmona, B.F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018.

[19] S.J.J. Leemans, W.M.P. van der Aalst, T. Brockhoff, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inf. Syst. 102 (2021) 101724.

[20] B. van Dongen, F. Borchert, BPI Challenge 2018, http://dx.doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972, 4TU.ResearchData URL https://data.4tu.nl/articles/dataset/BPI_Challenge_2018/12688355/1.

[21] G. De Giacomo, R. De 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.), Proceedings of the 12th International Conference on Business Process Management (BPM 2014), in: LNCS, vol. 8659, Springer, 2014, pp. 1–17.

[22] M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, E. Lamma, Abductive logic programming as an effective technology for the static verification of declarative business processes, Fundam. Inform. 102 (3–4) (2010) 325–361.

[23] M. Westergaard, Better algorithms for analyzing and enacting declarative workflow languages using LTL, in: S. Rinderle-Ma, F. Toumani, K. Wolf (Eds.), Proceedings of the 9th International Conference on Business Process Management (BPM 2011), in: LNCS, vol. 6896, Springer, 2011, pp. 83–98.

[24] S. Zhu, L. Tabajara, J. Li, G. Pu, M. Vardi, Symbolic LTLf synthesis, in: C. Sierra (Ed.), Proceedings of the 26-th International Joint Conference on Artificial Intelligence (IJCAI 2017), ijcai.org, 2017, pp. 1362–1369.

[25] S. Zhu, G. De Giacomo, G. Pu, M. Vardi, LTLf synthesis with fairness and stability assumptions, in: Proceedings of the 34-th AAAI Conference on Artificial Intelligence (AAAI 2020), AAAI Press, 2020, pp. 3088–3095.

[26] L.M. Tabajara, M.Y. Vardi, LTLF synthesis under partial observability: From theory to practice, in: J. Raskin, D. Bresolin (Eds.), Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2020), in: EPTCS, vol. 326, 2020, pp. 1–17.

[27] F.M. Maggi, M. Montali, M. Westergaard, W.M.P. van der Aalst, Monitoring business constraints with Linear Temporal Logic: An approach based on colored automata, in: S. Rinderle-Ma, F. Toumani, K. Wolf (Eds.), Proceedings of the 9th International Conference on Business Process Management (BPM 2011), in: LNCS, vol. 6896, Springer, 2011, pp. 132–147.

[28] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, On the relevance of a business constraint to an event log, Inf. Syst. 78 (2018).

[29] C. Di Ciccio, F.M. Maggi, M. Montali, J. Mendling, Ensuring model consistency in declarative process discovery, in: H.R. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), Proceedings of the 13th International Conference on Business Process Management (BPM 2015), in: LNCS, vol. 9253, Springer, 2015, pp. 144–159.

[30] F.M. Maggi, M. Westergaard, M. Montali, W.M.P. van der Aalst, Runtime verification of LTL-based declarative process models, in: S. Khurshid, K. Sen (Eds.), Proceedings of the 2nd International Conference on Runtime Verification (RV 2011), in: LNCS, vol. 7186, Springer, 2011, pp. 131–146.

[31] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, ACM Trans. Softw. Eng. Methodol. 20 (4) (2011) 14:1–14:64.

[32] A. Polyvyanyy, A.A. Kalenkova, Monotone conformance checking for partially matching designed and observed processes, in: Proceedings of the 1st International Conference on Process Mining (ICPM 2019), IEEE, 2019, pp. 81–88.

[33] S.J.J. Leemans, A. Polyvyanyy, Stochastic-aware conformance checking: An entropy-based approach, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Proceedings of the 32nd International Conference on Advanced Information Systems Engineering (CAiSE 2020), in: LNCS, vol. 12127, Springer, 2020, pp. 217–233.

[34] A. Polyvyanyy, A. Moffat, L. García-Bañuelos, An entropic relevance measure for stochastic conformance checking in process mining, in: B.F. van Dongen, M. Montali, M.T. Wynn (Eds.), 2nd International Conference on Process Mining (ICPM 2020), IEEE, 2020, pp. 97–104.

[35] A. Polyvyanyy, A. Solti, M. Weidlich, C. Di Ciccio, J. Mendling, Monotone precision and recall measures for comparing executions and specifications of dynamic systems, ACM Trans. Softw. Eng. Methodol. 29 (3) (2020) 17:1–17:41.

[36] G. Bergami, F.M. Maggi, M. Montali, R. Peñaloza, Probabilistic trace alignment, in: C. Di Ciccio, C. Di Francescomarino, P. Soffer (Eds.), Proceedings of the 3rd International Conference on Process Mining (ICPM 2021), IEEE, 2021, pp. 9–16.

[37] L. Rüschendorf, The wasserstein distance and approximation theorems, Probab. Theory Related Fields 70 (1) (1985) 117–129.

[38] M. de Leoni, F.M. Maggi, W.M.P. van der Aalst, An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Inf. Syst. 47 (2015) 258–277.

[39] A. Alman, C. Di Ciccio, D. Haas, F.M. Maggi, A. Nolte, Rule mining with RuM, in: B.F. van Dongen, M. Montali, M.T. Wynn (Eds.), Proceedings of the 2nd International Conference on Process Mining (ICPM 2020), IEEE, 2020, pp. 121–128.

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