Extending Temporal Business Constraints with Uncertainty

Fabrizio Maria Maggi[1(✉)], Marco Montali[1], Rafael Peñaloza[2], and Anti Alman[3]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
{maggi,montali}@inf.unibz.it
[2] University of Milano-Bicocca, Milan, Italy
rafael.penaloza@unimib.it
[3] University of Tartu, Tartu, Estonia
anti.alman@ut.ee

Abstract. Temporal business constraints have been extensively adopted to declaratively capture the acceptable courses of execution in a business process. However, traditionally, constraints are interpreted logically in a crisp way: a process execution trace conforms with a constraint model if all the constraints therein are satisfied. This is too restrictive when one wants to capture best practices, constraints involving uncontrollable activities, and exceptional but still conforming behaviors. This calls for the extension of business constraints with uncertainty. In this paper, we tackle this timely and important challenge, relying on recent results on probabilistic temporal logics over finite traces. Specifically, our contribution is threefold. First, we delve into the conceptual meaning of probabilistic constraints and their semantics. Second, we argue that probabilistic constraints can be discovered from event data using existing techniques for declarative process discovery. Third, 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 probabilities.

Keywords: Declarative process models · Temporal logics · Process mining · Probabilistic process monitoring · Probabilistic conformance checking

1 Introduction

A key functionality that any process-aware information system should support is that of *monitoring* [12]. Monitoring concerns the ability to verify at runtime whether an actual process execution conforms to a prescriptive business process model. This runtime form of conformance checking is instrumental to detect, and then suitably handle, deviations appearing in ongoing process instances [14].

A common way of representing monitoring requirements that capture the expected behavior prescribed by a process model is by using declarative, business constraints. Many studies demonstrated that, in several settings, business

D. Fahland et al. (Eds.): BPM 2020, LNCS 12168, pp. 35–54, 2020.
https://doi.org/10.1007/978-3-030-58666-9_3

constraints can be formalized in terms of temporal logic rules [19]. Within this paradigm, the *Declare* constraint-based process modeling language [21] has been introduced as a front-end language to specify business constraints based on Linear Temporal Logic over finite traces (LTL_f) [2]. The advantage of this approach is that the automata-theoretic characterization of LTL_f is based on standard, finite-state automata. These can be exploited to provide advanced monitoring facilities where the state of constraints is determined in a sophisticated way by combining the events collected at runtime with the possible, future continuations [1,16], in turn enabling the early detection of conflicting constraints [17].

In a variety of application domains, business constraints are inherently *uncertain*. This is clearly the case for constraints which: *(i)* capture best practices that have to be followed by default, that is, in most, but not necessarily all, cases; *(ii)* link controllable activities to activities that are under the responsibility of uncontrollable, external stakeholders; *(iii)* should hold in exceptional but still conforming courses of execution. Uncertainty is intrinsically present also when business constraints are discovered from event data. It is then very surprising that only very few approaches incorporate uncertainty as a first-class citizen. This is the case not just when the prescriptive behavior to be monitored is expressed as a set of business constraints, but also when a more conventional imperative approach is adopted [11].

It is well known that combining uncertainty with temporal logics is extremely challenging. This is due to the interplay of temporal operators and uncertainty, which becomes especially tricky considering that, usually, temporal logics are interpreted over infinite traces. The resulting, combined logics then come with semantic or syntactic restrictions (see, e.g., [8,20]). To tackle these issues, the probabilistic temporal logic over finite traces PLTL_f, and its fragment PLTL_f^0, have been recently proposed in [15]. Since these logics are defined over finite traces, they are the natural candidate to enrich existing constraint-based process modeling approaches with uncertainty.

In this paper, we indeed employ PLTL_f^0 to achieve this goal. Specifically, we exploit the fact that PLTL_f^0 handles time and probabilities in a way that naturally matches with the notion of conformance: a constraint φ holds with probability p if, by considering all the traces contained in a log, φ is satisfied by a fraction p of all the traces contained therein. Based on this observation, we provide a threefold contribution.

First, we exploit PLTL_f^0 to introduce *probabilistic constraints* and delve into their semantics and conceptual meaning; notably, our semantics is based on the already established notion of stochastic language [11]. We then show how probabilistic constraints can be used to naturally lift the Declare language to its probabilistic version ProbDeclare. Second, we observe that probabilistic Declare constraints can be discovered off-the-shelf using already existing techniques for declarative process discovery [7,9,13,22], with strong guarantees on the consistency of the generated models. In fact, the discovered constraints are for sure (probabilistically) consistent, without incurring in the notorious consistency issues experienced when the discovered constraints are interpreted in a

crisp way [4,5]. Third, 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 raise to multiple scenarios, each with its own distinct probability, where some of the constraints are expected to be satisfied, and the others to be violated. Specifically, we show how to lift existing automata-theoretic monitoring techniques to this more sophisticated probabilistic setting, and report on a proof-of-concept implementation of the resulting framework.

The paper is structured as follows. After preliminary notions introduced in Sect. 2, we introduce the syntax and semantics of probabilistic constraints in Sect. 3. In Sect. 4, we discuss how ProbDeclare constraints can be discovered from event data using existing techniques. In Sect. 5, we show how to monitor probabilistic constraints, and report on the corresponding implementation. In Sect. 6, we conclude the paper and spell out directions for future work.

2 Preliminaries

We consider a finite alphabet Σ of atomic activities. A *trace* τ over Σ is a finite sequence $a_1 \dots a_n$ of activities, where $a_i \in \Sigma$ for $i \in \{1, \dots, n\}$. The *length* of trace τ is denoted by $\mathit{length}(\tau)$. We use notation $\tau(i)$ to select the activity a_i present in position (also called instant) i of τ, and Σ^* for the (infinite) set of all possible traces over Σ. A *log* over Σ is a finite multiset of traces over Σ.

We recall next syntax and semantics of LTL_f [1,2], and its application in the context of Declare [18,21]. Consistently with the BPM literature, we make the simplifying assumption that formulae are evaluated on sequences where, at each point in time, only one proposition is true, matching the notion of trace defined above.

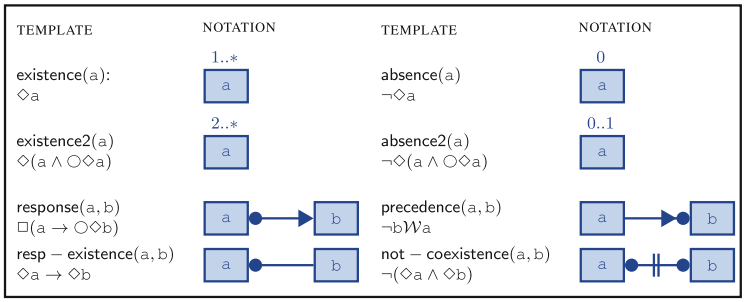
LTL Over Finite Traces. LTL_f has exactly the same syntax of standard LTL, but, differently from LTL, it interprets formulae over finite traces, as defined above. An LTL_f formula φ over Σ is built by extending propositional logic with temporal operators:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \qquad \text{where } a \in \Sigma.$$

A formula φ is evaluated over a trace τ in a valid instant i of τ, such that $1 \leq i \leq \mathit{length}(\tau)$. Specifically, we inductively define that φ *holds at instant* i *of* τ, written $\tau, i \models \varphi$, as:

- $\tau, i \models a$ 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 $i < \mathit{length}(\tau)$ and $\tau, i+1 \models \varphi$;
- $\tau, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff for some j such that $i \leq j \leq \mathit{length}(\tau)$, we have $\tau, j \models \varphi_2$ and for every k such that $i \leq k < j$, we have $\tau, k \models \varphi_1$.

Table 1. Some Declare templates, with their LTL$_f$ and graphical representations.



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 eventually, in a future instant, φ_2 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: *(i)* (eventually) $\Diamond\varphi = true\mathcal{U}\varphi$ is true, if there is a future state where φ holds; *(ii)* (globally) $\Box\varphi = \neg\Diamond\neg\varphi$ is true, if now and in all future states φ holds; *(iii)* (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 by requiring that φ_1 holds now and in all future states. We write $\tau \models \varphi$ as a shortcut notation for $\tau, 0 \models \varphi$, and say that formula φ is *satisfiable*, if there exists a trace τ such that $\tau \models \varphi$.

Example 1. The LTL$_f$ formula $\Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{accept})$ (called *response* in Declare) models that, whenever an order is closed, then it is eventually accepted. ◁

Every LTL$_f$ formula φ can be translated into a corresponding standard finite-state automaton $\mathcal{A}_\varphi$ that accepts all and only those finite traces that satisfy φ [1,2]. Although the complexity of reasoning with LTL$_f$ is the same as that of LTL, finite-state automata are much easier to manipulate in comparison with the Büchi automata used when formulae are interpreted over infinite traces. 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*.

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 constraints* over them, accepting every execution trace that satisfies all constraints. Constraints are specified via pre-defined LTL$_f$ templates, which come with a corresponding graphical representation (see Table 1 for the Declare templates 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. 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 [19,21], enactment and monitoring [1,16,21], and discovery support [13].

3 Probabilistic Constraints and ProbDeclare

We now lift LTL$_f$ constraints to their *probabilistic* version. As done in Sect. 2, we assume a fixed finite set Σ of activities.

Definition 1. *A* probabilistic constraint *over Σ is a triple $\langle \varphi, \bowtie, p \rangle$, where: (i) φ, the* constraint formula*, is an LTL$_f$ formula over Σ; (ii) $\bowtie \in \{=, \neq, \leq, \geq, <, >\}$ is the* probability operator*; (iii) p, the* constraint probability*, is a rational value in $[0, 1]$.* ◁

We use the compact notation $\langle \varphi, p \rangle$ for the probabilistic constraint $\langle \varphi, =, p \rangle$. A probabilistic constraint is interpreted over an event log, where traces have probabilities attached. Formally, we borrow the notion of *stochastic language* from [11].

Definition 2. *A* stochastic language *over Σ is a function $\rho : \Sigma^* \to [0, 1]$ that maps every trace over Σ onto a corresponding probability, so that $\sum_{\tau \in \Sigma^*} \rho(\tau) = 1$.* ◁

An event log can be easily turned into a corresponding stochastic language through normalization of the trace quantities, in particular by dividing the number of occurrences of each trace by the total number of traces in the log [11]. Similarly, a stochastic language can be turned into a corresponding event log by considering only the traces with non-zero probabilities.

Example 2. Consider the following traces over $\Sigma = \{\texttt{close}, \texttt{accept}, \texttt{nop}\}$: *(i)* $\tau_1 = \langle \texttt{close}, \texttt{accept} \rangle$; *(ii)* $\tau_2 = \langle \texttt{close}, \texttt{accept}, \texttt{close}, \texttt{nop}, \texttt{accept} \rangle$; *(iii)* $\tau_3 = \langle \texttt{close}, \texttt{accept}, \texttt{close}, \texttt{nop} \rangle$; *(iv)* $\tau_4 = \langle \texttt{close}, \texttt{nop} \rangle$. Log $\mathcal{L} = \{\tau_1^{50}, \tau_2^{30}, \tau_3^{10}, \tau_4^{10}\}$ corresponds to the stochastic language ρ 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^*. ◁

We say that a stochastic language ρ *satisfies* a probabilistic constraint $C = \langle \varphi, \bowtie, p \rangle$, written $\rho \models C$, iff $\sum_{\tau \in \Sigma^*, \tau \models \varphi} \rho(\tau) \bowtie p$. In other words, we first obtain all the traces that satisfy φ in the classical LTL$_f$ sense. We then use ρ to sum up the overall probability associated to such traces. We finally check whether the so-obtained number n is so that the comparison expression $n \bowtie p$ is true. Constraint $C = \langle \varphi, \bowtie, p \rangle$ is *plausible* if $p \neq 0$ and it is logically plausible, that is, $\rho \models C$ for some stochastic language ρ. This latter requirements simply means that φ is satisfiable in the classical LTL$_f$ sense.

Thanks to the correspondence between stochastic languages and event logs, we can define an analogous notion of satisfaction for event logs. With a slight abuse of notation, we use the same notation $\mathcal{L} \models C$ to indicate that event log $\mathcal{L}$ satisfies C. The resulting semantics naturally leads to interpret the constraint probability as a frequency, that is, as the fraction of conforming vs non-conforming traces contained in a log.

Example 3. The log $\mathcal{L}$ from Example 2 satisfies the probabilistic constraint $C_{ca} = \langle \Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{accept}), 0.8\rangle$. In fact, $\Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{accept})$ is satisfied[1] by traces τ_1 and τ_2, whose overall probability is $0.5 + 0.3 = 0.8$. ◁

This statistical interpretation of probabilities is central in the context of this paper, and leads to the following key observation: ρ *satisfies* $C = \langle \varphi, p\rangle$ iff it satisfies $\overline{C} = \langle \neg\varphi, 1 - p\rangle$. This reflects the intuition that, whenever φ holds in a fraction p of traces from an event log, then $\neg\varphi$ must hold in the complementary fraction $1 - p$ of traces from that log. Conversely, an unknown execution trace τ will satisfy φ with probability p, and will violate φ (i.e., satisfy $\neg\varphi$) with probability $1 - p$. This can be extended to the other probability operators in the natural way, taking into account that $\leq$ should be replaced by its dual $\geq$ (and vice-versa). Hence, we can interpret φ and $\neg\varphi$ as two alternative, *possible scenarios*, each coming with its own probability (respectively, p and $1 - p$). Whether such possible scenarios are indeed plausible depends, in turn, on their logical consistency (a plausible scenario must be logically satisfiable, that is, have at least one conforming trace) and associated probability (a plausible scenario must have a non-zero probability). A probabilistic constraint of the form $\langle \varphi, 1\rangle$ with φ satisfiable gives raise to a single possible world, where all traces in the log satisfy φ.

Example 4. Consider constraint C_{ca} from Example 3, modeling that, in 80% of the process traces, it is true that, whenever an order is closed, then it is eventually accepted. This is equivalent to assert that, in 20% of the traces, the response is violated, i.e., there exists an instant where the order is closed and not accepted afterward. Given an unknown trace τ, there is then 0.8 chance that τ will satisfy the response formula $\Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{accept})$, and 0.2 that τ will violate such a formula (i.e., satisfy its negation $\Diamond(\texttt{close} \wedge \neg\bigcirc\Diamond\texttt{accept})$). ◁

3.1 Probabilistic Declare

We now consider probabilistic declarative process models including multiple probabilistic constraints at once. We lift Declare to its probabilistic version ProbDeclare.

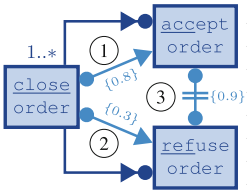
Definition 3. *A ProbDeclare model is a pair $\langle \Sigma, \mathcal{C}\rangle$, where Σ is a set of activities and $\mathcal{C}$ is a set of probabilistic constraints.* ◁

[1] Recall that a response constraint is satisfied if *every* execution of the source is followed by the execution of the target.

A stochastic language ρ over Σ satisfies a ProbDeclare model $\langle \Sigma, \mathcal{C} \rangle$ if it satisfies every probabilistic constraint $C \in \mathcal{C}$. It is interesting to note that, since $C = \langle \varphi, p \rangle$ and $\overline{C} = \langle \neg\varphi, 1 - p \rangle$ are equivalent, in ProbDeclare the distinction between existence and absence templates (cf. the first two lines of Table 1) gets blurred. In fact, $\langle \mathsf{existence}(\mathsf{a}), p \rangle$ corresponds to $\langle \Diamond a, p \rangle$. In turn, $\langle \Diamond a, p \rangle$ is semantically equivalent to $\langle \neg\Diamond a, 1-p \rangle$, which corresponds to $\langle \mathsf{absence}(a), 1-p \rangle$. The same line of reasoning applies to the existence2 and absence2 templates. All such constraints have in fact to be interpreted as the *probability of (repeated) occurrence* for a given activity.

Example 5. A small ProbDeclare model is shown on the left-hand side of Fig. 1, where only the equality operator is used for the various probabilities. Crisp constraints with probability 1 are shown in dark blue, and genuine probabilistic constraints are shown in light blue, with probability values attached. 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. Note that the sum of these probabilities exceeds 1, and, consequently, in a small fraction of traces, there will be an acceptance and also a rejection (capturing the fact that a previous decision on a closed order was subverted later on). 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 of the response constraint connecting `close order` to `refuse order` is only of 0.3. In 90% of the cases, it is asserted that acceptance and rejection are mutually exclusive. Finally, accepting/rejecting an order can only occur if the order was closed. ◁

We remark that ProbDeclare models and stochastic languages have a direct correspondence to the PLTL_f^0 logic and its interpretations (as defined in [15]). Specifically, a constraint of the form $\langle \varphi, \bowtie, p \rangle$ corresponds to the PLTL_f^0 formula $\circledcirc_{\bowtie p}\varphi$. PLTL_f^0 is a fragment of PLTL_f, also defined in [15]. Models of PLTL_f formulae are finite trees where nodes are propositional assignments, and edges carry probabilities, with the condition that the sum of the probabilities on the edges that depart from the same node add up to 1. A stochastic language ρ can then be easily represented as a PLTL_f model. This can be done by creating a tree where the root has as many outgoing edges as the number of traces in ρ. Each edge gets the probability that ρ associates to the corresponding trace. Then each edge continues into a single branch where nodes sequentially encode the events of the trace, and where edges all have probability 1. Due to this direct correspondence, we get that reasoning on ProbDeclare models (e.g., to check for satisfiability) can be carried out in PSpace, thus yielding the same complexity of LTL_f. This does not yet give a concrete technique to actually carry out reasoning and, more in general, understand how different probabilistic constraints and their probabilities interact with each other. This is answered in the next section, again taking advantage from the fact that, thanks to the correspondence with the PLTL_f framework in [15], all the techniques presented next are formally correct.



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

Fig. 1. A ProbDeclare model, with 8 constraint scenarios, out of which only 4 are logically plausible. Recall that each scenario implicitly contains also the three constraint formulae derived from the three constraints with probability 1.

3.2 Constraints Scenarios and Their Probabilities

Since a ProbDeclare model contains multiple probabilistic constraints, we have to consider that, probabilistically, a trace may satisfy or violate each of the constraints contained in the model, thus yielding multiple possible worlds, each one defining which constraints are satisfied, and which violated. E.g., in Fig. 1, we may have a trace containing `close order` followed by `accept order` and `refuse order`, thus violating the $\texttt{not} - \texttt{coexistence}$ constraint relating acceptance and refusal. This is indeed possible in 10% of the traces. More in general, consider a ProbDeclare model $M = \langle \Sigma, \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\}\rangle$. Each constraint formula φ_i is satisfied by a trace with probability p_i, and violated with probability $1 - p_i$. Hence, a model of this form implicitly yields, potentially, 2^n possible worlds resulting from all possible choices of which constraints formulae are satisfied, and which are violated (recall that violating a formula means satisfying its negation). We call such possible worlds *constraint scenarios*. The key point is to understand which scenarios are plausible, and with which overall probability, starting from the "local" probabilities attached to each single constraint. Overall, a set of constraint scenarios with their corresponding probabilities can be seen as a sort of *canonical stochastic language* that provides a uniform representation of all stochastic languages that satisfy the ProbDeclare model under study.

Example 6. If a constraint has probability 1, we do not need to consider the two alternatives, since every trace will need to satisfy its formula. An alternative way of reading this is to notice that the negated constraint would, in this case, have probability 0. Hence, to identify a scenario, we proceed as follows. We consider the $m \leq n$ constrains with probability different than 1, and fix an order over them. Then, a scenario is defined by a number between 0 and 2^{m-1}, whose corresponding binary representation defines which constraint formulae are satisfied, and which violated: specifically, for constraint formula φ_i of index i, if the bit in position $i-1$ is 1, then the scenario contains φ_i, if instead that bit is 0, then the scenario contains $\neg\varphi_i$. The overall formula describing a scenario is then simply the conjunction of all such formulae, together with all the formulae of constraints with probability 1. Clearly, each execution trace belongs to one and only one constraint scenario: it does so when it satisfies the conjunctive formula

associated to that scenario. We say that a scenario is *logically plausible*, if such a conjunctive LTL_f formula is satisfiable in the LTL_f sense: if it is not, then the scenario has to be discarded, since no trace will ever belong to it.

Figure 1 shows a ProbDeclare model with 6 constraints, three of which are crisp constraints with probability 1, while the other three are genuinely probabilistic. Circled numbers represent the ordering of such constraints. 8 possible constraint scenarios are induced, each enforcing the satisfaction of the three crisp constraints, while picking the satisfaction or violation of the three constraints $\mathsf{response}(\mathtt{close}, \mathtt{acc})$, $\mathsf{response}(\mathtt{close}, \mathtt{ref})$, and $\mathsf{not-coexistence}(\mathtt{acc}, \mathtt{ref})$. Logically speaking, we have to consider 6 different formulae: $\Box(\mathtt{close} \rightarrow \bigcirc\Diamond\mathtt{acc})$ and its negation $\Diamond(\mathtt{close} \wedge \neg\bigcirc\Diamond\mathtt{acc})$ (similarly for $\mathsf{response}(\mathtt{close},\mathtt{ref})$), as well as $\neg(\Diamond\mathtt{acc} \wedge \Diamond\mathtt{refuse})$ and its negation $\Diamond\mathtt{acc} \wedge \Diamond\mathtt{refuse}$. The resulting scenarios are reported in the same figure, using the naming conventions introduced before. E.g., scenario S_{101} is the scenario that satisfies $\mathsf{response}(\mathtt{close}, \mathtt{acc})$ and $\mathsf{not-coexistence}(\mathtt{acc}, \mathtt{ref})$, but violates $\mathsf{response}(\mathtt{close}, \mathtt{ref})$.

By checking the LTL_f satisfiability of the conjunction of the formulae entailed by a given scenario, we can see whether the scenario is logically plausible. In Fig. 1, only 4 scenarios are actually logically plausible. For example, S_{111} is *not* logically plausible. In fact, it requires that the order is closed (due to the crisp $1..*$ constraint on `close order`) and, consequently, that the order is eventually accepted and refused (due to the two response constraints attached to `close order`, which in this scenario must be both satisfied); however, the presence of both an acceptance and a refusal violates the $\mathsf{not-coexistence}$ constraint linking such two activities, contradicting the requirement that also this constraint must be satisfied in this scenario. S_{101} is logically plausible: it is satisfied by the trace where an order is closed and then accepted. All in all, we have 4 logically plausible scenarios: *(i)* S_{001}, where an order is closed and later not accepted nor refused; *(ii)* S_{011}, where an order is closed and later refused (and not accepted); *(iii)* S_{101}, where an order is closed and later accepted (and not refused); *(iv)* S_{110}, where an order is closed and later accepted and refused. ◁

While it is clear that a logically implausible scenario should correspond to probability 0, are all logically plausible scenarios really plausible when the actual probabilities are taken into account? By looking at Fig. 1, one can notice that scenario S_{001} is logically plausible: it describes traces where an order is closed but not accepted nor refused. As we will see, however, this cannot happen given the probabilities of 0.8 and 0.3 attached to $\mathsf{response}(\mathtt{close}, \mathtt{acc})$ and $\mathsf{response}(\mathtt{close}, \mathtt{ref})$. More in general, what is the probability of a constraint scenario, i.e., the fraction of traces in a log that belong to that scenario? Is it possible to assign probabilities to scenarios, while respecting the probabilities attached to the constraints? The latter question points out that a ProbDeclare model may be *unsatisfiable* (in a probabilistic sense), if there is no way to properly lift the probabilities attached to constraints to corresponding probabilities of the scenarios induced by those constraints. To answer these questions, we resort to the technique in [15]. We associate each scenario to a probability variable, keeping the

same naming convention. E.g., scenario S_{001} corresponds to variable x_{001}. More in general, for a ProbDeclare model $M = \langle \Sigma, \{\langle \varphi_1, \bowtie_1, p_1 \rangle, \ldots, \langle \varphi_n, \bowtie_n, p_n \rangle\}\rangle$, we construct the system $\mathcal{L}_M$ of inequalities using probability variables x_i, with i ranging from 0 to 2^n (in boolean):

$$\begin{array}{rll} x_i \geq & 0 & 0 \leq i < 2^n \\ \sum_{i=0}^{2^n-1} x_i = & 1 & \\ \sum_{j\text{th position is 1}} x_i \bowtie_j & p_j & 0 \leq j < n \\ x_i = & 0 & \text{if scenario } S_i \text{ is logically implausible} \end{array}$$

The first two lines guarantee that we assign a non-negative value to each variable, and that their sum is 1. We can see these assignments as probabilities, having the guarantee that all scenarios together cover the full probability spectrum. The third line verifies the probability associated to each constraint in M. In particular, it constructs one (in)equality per constraint $\langle \varphi_j, \bowtie_j, p_j \rangle$ in M, ensuring that all the variables that correspond to scenarios making φ_j true should all together yield a probability that is $\bowtie_j p_i$. The last line enforces that logically implausible scenarios get assigned probability 0. This shows how logical and probabilistic reasoning come together in $\mathcal{L}_M$.

We can use this system of inequalities to check whether a given ProbDeclare model is *satisfiable*: M is satisfiable if and only if $\mathcal{L}_M$ admits a solution. In fact, solving $\mathcal{L}_M$ corresponds to verifying whether the class of all possible traces can be divided in such a way that the proportions required by the probabilistic constraints in the different scenarios are satisfied. This, in turn, witnesses that there must be at least one logically plausible scenario that gets a non-zero probability. Checking whether $\mathcal{L}_M$ admits a solution can be done in PSPACE in the size of M, if we calculate the size as the length of the LTL_f formulae appearing therein [15].

Example 7. Consider the ProbDeclare model M containing two constraints:

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

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 order` 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:

$$S_{00} = \{\neg\Diamond\texttt{close}, \Diamond(\texttt{close} \wedge \neg\bigcirc\Diamond\texttt{acc})\} \quad S_{01} = \{\neg\Diamond\texttt{close}, \Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{acc})\}$$
$$S_{10} = \{\Diamond\texttt{close}, \Diamond(\texttt{close} \wedge \neg\bigcirc\Diamond\texttt{acc})\} \quad S_{11} = \{\Diamond\texttt{close}, \Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{acc})\}$$

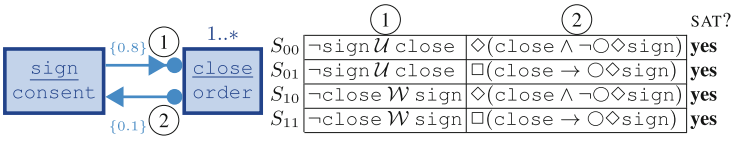


Fig. 2. A ProbDeclare model and its 4 constraint scenarios.

Scenario S_{00} is logically implausible: it requires and forbids that the order is closed; the other scenarios are instead all logically plausible. Hence, the equations of $\mathcal{L}_M$ are:

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

The equations yield $x_{10} = 0.2$, $x_{01} = 0.9$, and $x_{11} = -0.1$. This is an inconsistent probability assignment, and witnesses that it is not possible to properly assign suitable fractions of traces to the various constraint scenarios. ◁

When $\mathcal{L}_M$ is solvable, M is satisfiable. In addition, the solutions of $\mathcal{L}_M$ tell us what is the probability (or range of probabilities) for each constraint scenario. If a logically plausible scenario admits a probability that is strictly > 0, then it is actually *plausible* also in probabilistic terms. Contrariwise, a logically plausible scenario that gets assigned a probability that is forcefully 0 is actually *implausible*. This witnesses in fact that, due to the probabilities attached to the various constraints in M, the fraction of traces belonging to it must be 0.

Example 8. Consider the ProbDeclare model in Fig. 1. Its system of inequalities is so that $x_{000} = x_{010} = x_{100} = x_{111} = 0$, since the corresponding constraint scenarios are logically implausible. For the logically plausible scenarios, we instead get the following equalities, once the variables above are removed (being them all equal to 0):

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

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} is implausible, and that the most plausible 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 outlier 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. ◁

In general, the system $\mathcal{L}_M$ of inequalities for a ProbDeclare model M may have more than one solution. If this is the case, we can attach to each constraint scenario a probability interval, whose extreme values are calculated by minimizing and maximizing its corresponding variable over $\mathcal{L}_M$. Since these intervals are computed by analyzing each variable in isolation, not all the combinations of values residing in such intervals are actually consistent (which would entail yielding an overall probability of 1). Still, for sure these intervals contain probability values that are overall consistent, and, in addition, they provide a good indicator of which are the most (and less) plausible scenarios. We illustrate this in the next example.

Example 9. Consider the ProbDeclare model in Fig. 2. It comes with 4 constraint scenarios, obtained by considering the two constraint formulae precedence $(\texttt{sign},\texttt{close}) = \neg\texttt{close}\ \mathcal{W}\ \texttt{sign}$ and response$(\texttt{close},\texttt{sign}) = \Box(\texttt{close} \rightarrow \bigcirc\Diamond\texttt{sign})$, as well as their respective negated formulae $\neg\texttt{sign}\ \mathcal{U}\ \texttt{close}$ and $\Diamond(\texttt{close} \wedge \neg\bigcirc\Diamond\texttt{sign})$. All such scenarios are logically plausible, and hence the equations of the system are:

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

This system admits multiple solutions. In fact, by calculating the minimum and maximum values for the 4 variables, we get that: *(i)* scenario S_{00}, where the order is closed but consent is not signed, comes with probability interval $[0, 0.1]$; *(ii)* scenario S_{01}, where the order is closed and consent is signed afterward, comes with probability interval $[0, 0.1]$; *(iii)* scenario S_{10}, where the order is closed after having signed consent, comes with probability interval $[0.7, 0.8]$; *(iv)* scenario S_{11}, where the order is closed and consent is signed at least twice (once before, and once afterward), comes with probability interval $[0.1, 0.2]$. ◁

4 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 guarantee: that the discovered model is always consistent. We use the standard notation $[\cdot]$ for multisets, and use superscript numbers to identify the multiplicity of an element in the multiset.

A plethora of different algorithms have been devised to discover Declare models from event data [7,9,13,22]. In general, the vast majority of these algorithms adopt the following approach to discovery: (1) Candidate constraints are generated by analyzing the activities contained in the log. (2) For each constraint, its *support* is computed as the fraction of traces in the log where the constraint holds. (3) Candidate constraints are filtered, retaining only those whose support exceeds a given threshold. (4) Further filters (e.g., considering the "relevance" of a constraint [6]) are applied. (5) The overall model is checked for satisfiability,

operating with different strategies if it is not; this is necessary since constraints with high support, but less than 1, may actually conflict with each other [4,5]. In this procedure, the notion of support is formalized as follows.

Definition 4. *The* support *of an LTL$_f$ constraint φ in an event log $\mathcal{L} = [\tau_1, \ldots, \tau_n]$ is $supp_{\mathcal{L}}(\varphi) = \frac{|\mathcal{L}_\varphi|}{|\mathcal{L}|}$, where $\mathcal{L}_\varphi = [\tau \in \mathcal{L} \mid \tau \models \varphi]$.* ◁

We can adopt this approach off-the-shelf to discover ProbDeclare constraints: *we just use the constraint support as its associated probability, with operator* $=$. In other words, if φ is discovered with support p, we turn it into the probabilistic constraint $\langle \varphi, p \rangle$. When doing so, we can also relax step (3), e.g., to retain constraints with a very low support, implying that their negated versions have a very high support.

Example 10. Consider $\mathcal{L} = [\langle \texttt{close}, \texttt{acc} \rangle^7, \langle \texttt{close}, \texttt{ref} \rangle^2, \langle \texttt{close}, \texttt{acc}, \texttt{ref} \rangle^1]$, 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 in general, the entire ProbDeclare model of Fig. 1 can be discovered from $\mathcal{L}$. ◁

A second key observation is that once this procedure is used to discover ProbDeclare constraints, step (5) is unnecessary: the overall discovered model is in fact guaranteed to be satisfiable (in our probabilistic sense).

Theorem 1. *Let Σ be a set of activities, $\mathcal{L}$ an event log over Σ, and $\mathcal{C} = \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\}$ a set of probabilistic constraints, such that for each $i \in \{1, \ldots, n\}$, $p_i = supp_{\mathcal{L}}(\varphi_i)$. The ProbDeclare model $\langle \Sigma, \mathcal{C} \rangle$ is satisfiable.* ◁

Proof. Technically, $\langle \Sigma, \mathcal{C} \rangle$ is satisfiable if its corresponding PLTL$_f^0$ formula $\Phi := \{\circledcirc_{p_1} \varphi_1, \ldots, \circledcirc_{p_n} \varphi_n\}$ is satisfiable. To show this, we simply use $\mathcal{L}$ to build a model of Φ. For every set $I \subseteq \{1, \ldots, n\}$, let φ_I be the LTL$_f$ formula $\varphi_I := \bigwedge_{i \in I} \varphi_i \wedge \bigwedge_{i \notin I} \neg \varphi_i$, and let $\mathcal{L}_I$ be the sublog of $\mathcal{L}$ containing all the traces that satisfy φ_I. Note that the sublogs $\mathcal{L}_I$ form a partition of $\mathcal{L}$; that is, every trace appears in exactly one such $\mathcal{L}_I$. For each I such that $\mathcal{L}_I$ is not empty, choose a representative $t_I \in \mathcal{L}_I$ and let $p_I := \frac{|\mathcal{L}_I|}{|\mathcal{L}|}$ be the fraction of traces that belong to $\mathcal{L}_I$. We build a stochastic language ρ by setting $\rho(t_I) = p_I$ for each I such that $\mathcal{L}_I \neq \emptyset$ and $\rho(\tau) = 0$ for all other traces. We need to show that ρ satisfies $\mathcal{C}$. Consider a constraint $\langle \varphi, p \rangle \in \mathcal{C}$; we need to show that $\sum_{\tau \models \varphi} \rho(\tau) = p$. Note that by construction, $\sum_{\tau \models \varphi} \rho(\tau) = \sum_{t_I \models \varphi} p_I$ and since $\mathcal{L}_I$ form a partition, the latter is, in fact, the fraction of traces that satisfy φ. On the other hand, p is also the support of φ; that is, the proportion of traces satisfying φ. Hence, both values are equal, and ρ satisfies the ProbDeclare model. ⊣

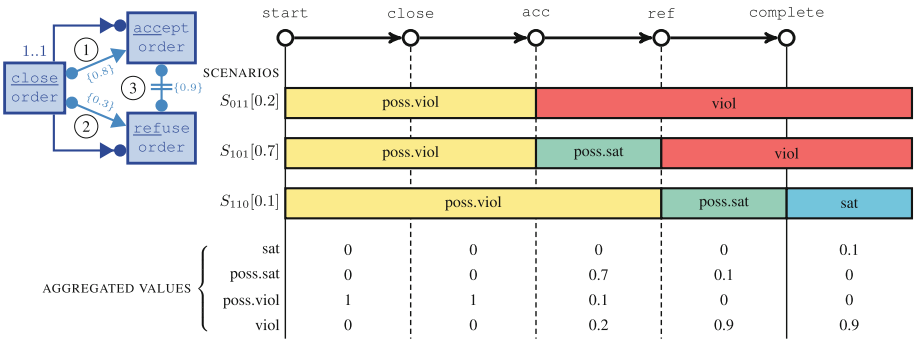


Fig. 3. Result computed by monitoring the ProbDeclare model on the top left against the trace $\langle$`close`, `acc`, `ref`$\rangle$, which conforms to the outlier constraint scenario where the two response constraints are satisfied, while the not − coexistence one is violated.

trace $\langle$`close`, `sign`$\rangle$ trace $\langle$`sign`, `close`$\rangle$

Fig. 4. Output of the implemented tool on the example in Fig.2.

By this theorem, probabilistic constraints can be discovered in a *purely local* way, having the guarantee that they will never conflict with each other. Obviously, non-local filters can still prove useful to prune implied constraints and select the most relevant ones. Also, note that the probabilities of the discovered constraints can be easily adjusted when new traces are added to the log, by incrementally

recomputing the support values after checking how many new traces satisfy the various constraints.

There are many open questions that deserve a dedicated investigation, such as: when do we stop the discovery procedure, now that every constraint can be retained, irrespectively of its support? What is the impact of retaining constraints with various degrees of support in terms of over/under-fitting? How to learn constraints with probability operators different from just equality? And how does this impact generalization?

5 Monitoring Probabilistic Constraints

In Sect. 3.2, we have shown how we can take a ProbDeclare model and generate its constraint scenarios, together with their corresponding probability intervals. We now describe how this technique can be directly turned into an operational probabilistic monitoring and conformance checking framework.

Let $M = \langle \Sigma, \mathcal{C} \rangle$ be a ProbDeclare model with n probabilistic constraints. For simplicity, we do not distinguish between crisp and genuinely probabilistic constraints, nor prune away implausible scenarios: the produced monitoring results do not change, but obviously our implementation, presented at the end of this section, takes into account these aspects for optimization reasons. M generates 2^n constraint scenarios. As discussed in Sect. 3.2, each scenario S comes with a corresponding characteristic LTL_f formula, which amounts to the conjunction of positive and negated constraints in $\mathcal{C}$, where the decision of which ones are taken positive and which negative is defined by the scenario itself. We denote such a formula by $\mathit{formula}(S)$. For example, if $\mathcal{C} = \{\langle \varphi_1, p_1 \rangle, \langle \varphi_2, p_2 \rangle, \langle \varphi_3, p_3 \rangle\}$, then $\mathit{formula}(S_{101}) = \varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$. In addition, if M is satisfiable, and hence $\mathcal{L}_M$ is solvable, each scenario S comes with its own probability. More specifically, we have to consider the case where multiple (possibly infinite) solutions exist for $\mathcal{L}_M$. There are various possibilities to handle this case. We tackle it by resorting to a quite direct approach: for each scenario S, we solve $\mathcal{L}_M$ twice by respectively imposing, as an additional constraint, that the probability variable for S has to be *minimized*/*maximized*. This, in turn, yields a probability interval for S, which we denote by $\mathit{prob}(S)$. From Example 9, we have, e.g., that $\mathit{prob}(S_{10}) = [0.7, 0.8]$. More sophisticated ways to extract probabilities from $\mathcal{L}_M$ can be investigated.

5.1 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, *conforms* to a given ProbDeclare model M. This amounts to a *probabilistic version of conformance checking* that can be tackled as follows. We fix an order over the constraints in M, and precompute the probability intervals of the scenarios induced by M. At runtime, we consider the current prefix τ and, for every formula φ of each probabilistic constraint $\langle \varphi, \bowtie, p \rangle \in M$ considered in isolation, we output 1 if $\tau \models \varphi$, and 0 otherwise. One effective way to do this check is

to precompute the finite-state automaton that recognizes all and only the finite traces accepted by φ [1], then checking at runtime whether τ is recognized by that automaton. The automaton can be determinized upfront, making in turn possible to perform this check incrementally. The overall, so-produced output, interpreted as an array of bits, matches exactly one and only one scenario of M. If the scenario has probability 0, then τ is not conforming to M, whereas if the scenario has a proper probability (interval), then τ conforms to M, and the actual probability value can be used to understand whether τ represents a common or an outlier behavior - that is, coupling "conformance" with an estimation of the degree of "conformism". This approach comes with a main limitation though: it does not reason on the possible future continuations of the current prefix. This is particularly limiting in a probabilistic setting: monitoring a prefix makes it impossible to understand if and how its matching scenario will change as new events are acquired.

5.2 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 anymore the constraints in isolation, but we have to reason at the level of scenarios. Notice that most of the computational burden is at design time, whereas, at runtime, we incur simply in the cost of incrementally recognizing a growing prefix on a fixed set of deterministic finite-state automata, which is computationally lightweight.

To handle full monitoring, first notice that the characteristic formula of a scenario is in standard LTL_f, and so we can construct a *scenario monitor* by recasting well-known automata-theoretic techniques [1,16]. Specifically, given an LTL_f formula φ over a set Σ of activities, and a partial trace τ representing an ongoing process execution, a monitor outputs one of the four following truth values:

- 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 $\tau \cdot \tau' \not\models \varphi$.

This is used as follows. For each plausible scenario S over M, we construct the monitor for S.[2] We then track the evolution of a running trace by delivering its events to *all* such monitors in parallel, returning the truth values they produce. As pointed out in Sect. 5.1, at runtime we do not always know to which scenario the trace will belong to once completed. However, we can again combine logical and probabilistic reasoning to obtain a meaningful feedback.

A first key observation is that, for every partial trace, at most one scenario can turn out to be permanently or temporarily satisfied. Call this scenario S. In the first case, this verdict is irrevocable, and also implies that all other scenarios are permanently violated. This witnesses that no matter how the execution continues, the resulting trace will for sure belong to S. We then return immediately that the trace is conforming, and also return $prob(S)$ to give an indication about the degree of conformism of the trace (see above). In the second case, 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.

A second key observation is that multiple scenarios may be at the same time temporarily or permanently violated. For this reason, we need to aggregate in some way the probabilities of the scenarios that produce the same truth value to have an indication of the overall probability associated with that value. Having this aggregated probability is useful to have sophisticated feedback about the monitored trace. For example, the aggregated probability for permanently violated scenarios is useful as it can never decrease over time: it is possible that new scenarios become permanently violated, but those that already are will never switch to a different truth value. So a high value 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. At the same time, the aggregated value of permanent violation can be used as a conditional probability, when one is interested in understanding what is the probability that a trace will end up in a given scenario. The extreme values of the *aggregated probability interval* for temporary/permanent violations are computed using the system of inequalities $\mathcal{L}_M$. In particular, this is done by adding a constraint that minimizes/maximizes the sum of the probability variables associated to the scenarios that produce that truth value.

Example 11. Consider the ProbDeclare model in Fig. 1 with its three plausible scenarios (recall that four scenarios are logically plausible there, but one of those has probability 0, so only three remains to be monitored). Figure 3 shows the result produced when monitoring a trace that at some point appears to belong to the most plausible scenario, but in the end turns out to conform to the least plausible one. From the image, we can also clearly see that the trace consisting only of a close order activity would be judged as non-conforming, as it would violate all scenarios. ◁

[2] Implausible scenarios are irrelevant: they produce an output that is associated to probability 0.

This probabilistic monitoring technique has been fully implemented.[3] For solving systems of inequalities, we use the LP solver[4]. The implementation comes with various optimizations. First, scenarios are computed by directly imposing that crisp constraints with probability 1 must hold in their positive form in all scenarios. Second, only plausible scenarios are retained for monitoring. Third, the results obtained by minimizing and maximizing for aggregate probability variables are cached, to avoid solving multiple times the same problem. Figure 4 shows the output of the implemented monitoring tool on the example in Fig. 2 and for two different traces.[5] Here, the aggregated probability intervals are shown with a dark gray or light gray background depending on whether their midpoint is closer to 1 or to 0, respectively. The first trace (on the left) is classified as belonging to scenario S_{01} and is an outlier because this scenario has low probability (corresponding to a probability interval of $prob(S_{01}) = [0.0, 0.1]$). The second trace (on the right) is classified as belonging to the highly plausible scenario S_{10} (corresponding to a probability interval of $prob(S_{10}) = [0.7, 0.8]$).

6 Conclusion

In this paper, we have introduced the notion of probabilistic business constraint and demonstrated how this notion affects the outcomes of standard process monitoring (and mining) approaches based on Declare, when standard Declare is replaced by its probabilistic counterpart. We have introduced a framework for monitoring a trace with respect to a set of probabilistic constraints. The framework classifies completed traces as violating a given probabilistic model or as belonging to a certain constraint scenario (i.e., satisfying a certain combination of probabilistic constraints). Technically, our approach seamlessly handles more sophisticated logics for specifying constraints, only requiring that they have a corresponding automata-theoretic characterization. Thus, for example, regular expressions or LDL_f [1] can be used in place of LTL_f, as well as FO-LTL_f [3].

For future work, we plan to better investigate the influence of probabilistic constraints on the state-of-the-art techniques for declarative process mining. In addition, as it has been shown in the paper, very sophisticated monitoring feedbacks can be extracted, but their interpretation is not at all straightforward. A dedicated study focused on end user-tailored feedbacks is needed. Last but not least, we plan to relate, and possibly integrate, the declarative approach presented in this paper with recent advancements in stochastic conformance checking on imperative process models [10]. Note that, if we extend our approach with probabilities within constraints (ending up in the full logic studied in [15]), we have to manipulate more sophisticated automata that are reminiscent of the stochastic automata used in [10]. At the same time, the entropy-based approach brought forward in [10] could be used in our setting to measure the "distance"

[3] https://bitbucket.org/fmmaggi/probabilisticmonitor/src/master/.
[4] http://lpsolve.sourceforge.net/5.5/.
[5] In the screenshots, 1 and 2 represent the probabilistic constraints labeled with 1 and 2 in Fig. 2, whereas 3 represents the crisp constraint in the same example.

between a set of probabilistic constraints and an event log whose trace frequencies are not fully aligned to what prescribed by the probabilistic constraints.

Acknowledgments. This work has been supported by the Estonian Research Council (project PRG887).

References

1. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints Based on LTL and LDL for finite traces. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 1–17. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_1
2. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 3–9 August 2013, pp. 854–860 (2013). http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997
3. De Masellis, R., Maggi, F.M., Montali, M.: Monitoring data-aware business constraints with finite state automata. In: International Conference on Software and Systems Process 2014, ICSSP 2014, Nanjing, China, 26–28 May 2014, pp. 134–143 (2014). https://doi.org/10.1145/2600821.2600835
4. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Ensuring model consistency in declarative process discovery. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 144–159. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_9
5. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017). https://doi.org/10.1016/j.is.2016.09.005
6. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Inf. Syst. **78**, 144–161 (2018). https://doi.org/10.1016/j.is.2018.01.011
7. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 24:1–24:37 (2015). https://doi.org/10.1145/2629447
8. Kovtunova, A., Peñaloza, R.: Cutting diamonds: a temporal logic with probabilistic distributions. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October–2 November 2018, pp. 561–570 (2018). https://aaai.org/ocs/index.php/KR/KR18/paper/view/18037
9. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_25
10. Leemans, S.J.J., Polyvyanyy, A.: Stochastic-aware conformance checking: an entropy-based approach. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 217–233. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_14
11. Leemans, S.J.J., Syring, A.F., van der Aalst, W.M.P.: Earth movers' stochastic conformance checking. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNBIP, vol. 360, pp. 127–143. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26643-1_8

12. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015). https://doi.org/10.1016/j.is.2015.02.007
13. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_18
14. Maggi, F.M., Montali, M., van der Aalst, W.M.P.: An operational decision support framework for monitoring business constraints. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 146–162. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28872-2_11
15. Maggi, F.M., Montali, M., Peñaloza, R.: Temporal logics over finite traces with uncertainty. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, 7–12 February 2020, pp. 10218–10225 (2020). https://aaai.org/ojs/index.php/AAAI/article/view/6583
16. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_13
17. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 131–146. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_11
18. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. LNBIP, vol. 56. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14538-4
19. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. ACM Trans. Web **4**(1), 3:1–3:62 (2010). https://doi.org/10.1145/1658373.1658376
20. Ognjanovic, Z.: Discrete linear-time probabilistic logics: completeness, decidability and complexity. J. Log. Comput. **16**(2), 257–285 (2006). https://doi.org/10.1093/logcom/exi077
21. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15–19 October 2007, Annapolis, Maryland, USA, pp. 287–300. IEEE Computer Society (2007). https://doi.org/10.1109/EDOC.2007.14
22. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 290–305. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_18