Probabilistic Conformance Checking Based on Declarative Process Models

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

[1] Free University of Bozen-Bolzano, Bolzano, Italy
{maggi,montali}@inf.unibz.it
[2] University of Milano-Bicocca, Milan, Italy
rafael.penaloza@unimib.it

Abstract. Conformance checking is a fundamental task to detect deviations between the actual and the expected courses of execution of a business process. In this context, temporal business constraints have been extensively adopted to declaratively capture the expected behavior of the process. However, traditionally, these 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, we equip business constraints with a natural, probabilistic notion of uncertainty. We discuss the semantic implications of the resulting framework and show how probabilistic conformance checking and constraint entailment can be tackled therein.

Keywords: Declarative process models · Probabilistic temporal logics · Conformance checking

1 Introduction

Temporal business constraints have been extensively adopted to declaratively capture the acceptable courses of execution of a business process for conformance checking. In particular, the *Declare* constraint-based process modeling language [8] has been introduced as a front-end language to specify business constraints based on Linear Temporal Logic over finite traces (LTL_f) [2].

In general, business constraints are interpreted logically in a crisp way. This means that an execution trace conforms with a constraint model if all the constraints therein are satisfied. This is too restrictive when one wants to capture patterns that recur in many application domains, such as:

https://doi.org/10.1007/978-3-030-58135-0_8

- best practices, captured as constraints that should hold in the majority, but not necessary all cases (example: an order is shipped via truck in 90% of the cases);
- outlier behaviors, i.e., constraints that only apply to very few cases that should still considered to be conforming (example: an order is shipped via car in less than 1% of the cases);
- constraints involving activities that are not all necessarily controlled by the organization that orchestrates the process, and for which only some guarantees can be given about their proper executability (example: whenever an order is accepted, payment is performed by the customer in 8 cases out of 10).

Surprisingly enough, to the best of our knowledge no attempt has been done, so far, to make constraint-based process modeling approaches able to capture this form of uncertainty. In this paper, we tackle this timely and important challenge, relying on recent results on probabilistic temporal logics over finite traces [5]. Specifically, *we equip business constraints with a natural, probabilistic notion of uncertainty based on the ratio of traces in a log that must satisfy the constraint*, and use the resulting probabilistic constraints to lift Declare to its probabilistic variant that we call ProbDeclare. We then discuss the semantic implications of this approach, showing how it has to combine logical and probabilistic reasoning to tackle the semantics of probabilistic constraints and their interplay. We finally show how this combined reasoning can be applied to verify the consistency of a ProbDeclare model, do conformance checking, and carry out probabilistic constraint entailment, i.e., estimate with which probability a ProbDeclare model implies a given LTL$_f$ formula.

2 LTL over Finite Traces and the Declare Framework

As a formal basis for specifying crisp (temporal) business constraints, we adopt the customary choice of Linear Temporal Logic over finite traces (LTL$_f$ [1,2]). This logic is at the basis of the well-known Declare [8] constraint-based process modeling language. We provide here a gentle introduction to this logic and to the Declare framework.

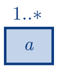
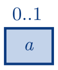
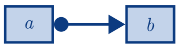
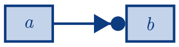
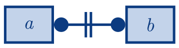
2.1 Linear Temporal Logic over Finite Traces

LTL$_f$ has exactly the same syntax as standard LTL, but, differently from LTL, it interprets formulae over an unbounded, yet finite linear sequence of states. Given an alphabet Σ of atomic propositions (in our setting, representing activities), an LTL$_f$ formula φ is built by extending propositional logic with temporal operators:

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

The semantics of LTL$_f$ is given in terms of *finite traces* denoting finite, *possibly empty* sequences $\tau = \langle \tau_0, \ldots, \tau_n \rangle$ of elements of 2^{Σ}, containing all possible

Table 1. Some Declare templates, their textual and graphical representation, the corresponding LTL$_f$ formalization and the LTL$_f$ formula capturing their complement (i.e., their logical negation).

TEXT	NOTATION	LTL$_f$ FORMULA (φ)	COMPLEMENT ($\neg\varphi$)
`existence(a)`		$\Diamond a$	$\Box\neg a$
`absence2(a)`		$\neg\Diamond(a \wedge \bigcirc\Diamond a)$	$\Diamond(a \wedge \bigcirc\Diamond a)$
`response(a, b)`		$\Box(a \rightarrow \Diamond b)$	$\Diamond(a \wedge \Box\neg b)$
`precedence(a, b)`		$\neg b \mathcal{W} a$	$\neg a \,\mathcal{U}\, b$
`not-coexistence(a, a)`		$\neg(\Diamond a \wedge \Diamond b)$	$\Diamond a \wedge \Diamond b$

propositional interpretations of the propositional symbols in Σ. In the context of this paper, consistently with the literature on business process execution traces, we make the simplifying assumption that in each point of the sequence, one and only one element from Σ holds. Under this assumption, τ becomes a total sequence of activity occurrences from Σ, matching the standard notion of (process) execution trace. We indicate with Σ^* the set of all traces over Σ. The evaluation of a formula is done in a given state (i.e., position) of the trace, and we use the notation $\tau, i \models \varphi$ to express that φ holds in the position i of τ. We also use $\tau \models \varphi$ as a shortcut notation for $\tau, 0 \models \varphi$. This denotes that φ holds over the entire trace τ starting from the very beginning and, consequently, logically captures the notion of *conformance* of τ against φ. We also say that φ is *satisfiable* if it admits at least one conforming trace.

In the syntax above, operator $\bigcirc$ denotes the *next state* operator, and $\bigcirc\varphi$ is true if there exists a next state (i.e., the current state is not at the end of the trace), and in the next state φ holds. Operator $\mathcal{U}$ instead is the *until* operator, and $\varphi_1 \,\mathcal{U}\, \varphi_2$ is true if φ_1 holds now and continues to hold until eventually, in a future state, 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:

- (eventually) $\Diamond\varphi = \mathit{true}\,\mathcal{U}\,\varphi$ 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 by requiring that φ_1 holds now and in all future states.

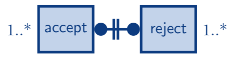
Example 1. The LTL$_f$ formula $\Box(\mathsf{accept} \rightarrow \Diamond\mathsf{pay})$ models that, whenever an order is accepted, then it is eventually paid. The structure of the formula follows what is called *response template* in Declare. ◁

2.2 Declare

Declare [8] is a declarative process modeling language based on LTL$_f$. More specifically, a Declare model fixes a set of activities, and a set of constraints over such activities, formalized using LTL$_f$ formulae. The overall model is then formalized as the conjunction of the LTL$_f$ formulae of its constraints.

Among all possible LTL$_f$ formulae, Declare selects some pre-defined patterns. Each pattern is represented as a Declare template, i.e., a formula with placeholders to be substituted by concrete activities to obtain a constraint. Constraints and templates have a graphical representation; Table 1 lists the Declare templates used in this paper. A Declare model is then graphically represented by showing its activities, and the application of templates to such activities (which indicates how the template placeholders have to be substituted to obtain the corresponding constraint).

Example 2. Consider the following Declare model, constituting a (failed) attempt of capturing a fragment of an order-to-shipment process:



The model indicates that there are two activities to accept or reject an order, that these two activities are mutually exclusive, and that both of them have to be executed. These constraints are obviously contradictory and, in fact, the model is inconsistent, since its LTL$_f$ formula $\Diamond\mathsf{accept} \wedge \Diamond\mathsf{reject} \wedge \neg(\Diamond\mathsf{accept} \wedge \Diamond\mathsf{reject})$ is unsatisfiable. ◁

3 Probabilistic Business Constraints

As recalled in Sect. 2, business constraints captured with LTL$_f$ are interpreted in a crisp way, i.e., they are expected to hold in *every* execution of the process. We now extend constraints with a natural notion of uncertainty introducing probabilistic constraints. Then, we show how this notion can be used to make Declare probabilistic and discuss informally the interplay of multiple probabilistic constraints.

3.1 Probabilistic Constraints: Definition and Semantics

For simplicity, we only consider the case of *exact* probability, but all the considerations we do directly carry over the more general case where the probability of a constraint is related to a given quantity with comparison operators ($\leq$, $<$, $=$, and their duals).

Definition 1. *A* probabilistic constraint *over a set Σ of activities is a pair $\langle\varphi, p\rangle$, where φ is an LTL$_f$ formula over Σ representing the* constraint formula*, and p is a rational value in $[0,1]$ representing the* constraint probability. ◁

Since a probabilistic constraint quantifies *how many* traces should satisfy it, it has to be interpreted over multiple traces that, as a whole, constitute an event log for the process of interest. In particular, the *constraint holds in a log if the ratio of traces in the log that satisfy the constraint formula is equal to the constraint probability.* This naturally leads to interpret the constraint probability statistically as the ratio of conforming vs non-conforming traces contained in a given log.

For simplicity, we stick here with the standard definition of event log, but we could alternatively adopt the stochastic interpretation of an event log, following [3].

Definition 2. *An* (event) log *over a set Σ of activities is a multiset of traces over Σ, i.e., a multiset over Σ^*.* ◁

Given a log $\mathcal{L}$, we write $\tau^n \in$ log to indicate that trace τ appears n times in $\mathcal{L}$. Trace τ belongs to $\mathcal{L}$ if $\tau^n \in$ log with $n > 0$. With these notions at hand, we say that a probabilistic constraint $\langle \varphi, p \rangle$ holds in a log $\mathcal{L}$ or, equivalently, that $\mathcal{L}$ satisfies $\langle \varphi, p \rangle$, if $\sum_{\tau^n \in \mathcal{L}, \tau \models \varphi} n = p$.

Note that the probabilistic constraint $\langle \varphi, p \rangle$ is equivalent to the probabilistic constraint $\langle \neg\varphi, 1 - p \rangle$. In fact, given a log $\mathcal{L}$, if the ratio of traces in $\mathcal{L}$ that satisfies φ is p, then the remaining $1 - p$ traces in $\mathcal{L}$ do not satisfy φ, i.e., they satisfy the constraint complement $\neg\varphi$.

Example 3. Consider the probabilistic constraint $\langle \texttt{existence}(\mathsf{accept}), 0.8 \rangle$. It indicates that 80% of the traces in a log contain at least one occurrence of accept or, equivalently, that 20% of the traces do not contain any execution of accept. This constraint holds in the log: $\left[\langle \mathsf{accept} \rangle^2, \langle \mathsf{accept}, \mathsf{reject} \rangle, \langle \mathsf{reject} \rangle^4, \langle \mathsf{accept}, \mathsf{cancel} \rangle^3, \langle \mathsf{accept}, \mathsf{pay}, \mathsf{ship} \rangle^{10} \right]$, since 16 out of the 20 traces contained therein include (at least) one occurrence of accept, i.e., they satisfy $\texttt{existence} = \Diamond\mathsf{accept}$. ◁

3.2 ProbDeclare and the Issue of Multiple Interacting Constraints

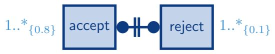
We now use the notion of probabilistic constraint as the basic building block to lift Declare to its probabilistic version, which we call ProbDeclare.

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.* ◁

A standard Declare model corresponds to a ProbDeclare model where all probabilistic constraints have probability 1. In the remainder of the paper, when drawing ProbDeclare diagrams, we then adopt the following notation: *(i)* whenever a constraint has probability 1, we draw it as a standard Declare constraint; *(ii)* Whenever a constraint has probability $p < 1$, we show it in light blue, and we annotate it with the probability value p.

The main issue that arises when multiple, genuinely probabilistic constraints are present in the same ProbDeclare model is that they interact with each other depending on their constraint formulae and probabilities. In particular, to satisfy the probabilistic constraints contained in a ProbDeclare model, a log must contain suitable fractions of traces so as to satisfy *all* probabilistic constraints and their probabilities, with the effect that some of these traces may contribute to the computation of the ratios for different constraints. The following examples intuitively illustrate this interplay. The first example shows that inconsistent Declare models may become consistent if the conflicting constraints are associated with suitable probabilities.

Example 4. Consider the following probabilistic variant of the (inconsistent) Declare diagram shown in Example 2.



This model contains two mutually exclusive activities, accept and reject, and indicates that often (in 80% of the cases) accept is selected, whereas rarely (in 10% of the cases) reject is selected. This captures a form of probabilistic choice, which also implicitly contemplates that none of the two activities occurs. In fact, from this very simple model, we can infer the following conditions on satisfying logs:

1. The `not-coexistence` constraint linking accept and reject is crisp, and consequently no trace in the log can contain both accept and reject.
2. Point 1, combined with the probabilistic `existence` constraint on accept, means that a trace in the log has 0.8 probability of containing accept (which means that reject will not occur), and 0.2 probability of not containing accept (which means that reject may occur or not).
3. A similar line of reasoning can be applied to the existence of reject, which must appear in 10% of the traces in the log.

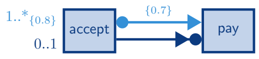
All in all, combining all the constraints, we get that the 10% of traces containing reject must be disjoint from the 80% containing accept. This implicitly means that in the remaining 10% of the traces, none of the two activities occur. $\triangleleft$

The second example shows that a consistent ProbDeclare model may become inconsistent by changing the values of probabilities.

Example 5. Consider again the ProbDeclare diagram in Example 4. Clearly, if we change to 1 the constraint probabilities attached to the two `existence` constraints, the model becomes identical to that in Example 2, consequently becoming inconsistent. More in general, the model becomes inconsistent whenever the sum of the two probabilities exceeds 1. This witnesses that there must exist some traces in which both constraints are satisfied, which contradicts the fact that accept and reject should not coexist. More precisely, if we denote by p_a and p_r the probabilities attached to the two `existence` constraints, then there is a probability $p_a + p_r - 1$ of having a trace that contains both accept and reject. For example, if we set $p_a = 0.8$ and $p_r = 0.3$, we have that 10% of the traces in the log should contain both accept and reject, which is impossible given the fact that every trace in the log should satisfy `not-coexistence`(accept, reject). $\triangleleft$

The last example shows that, as customary in models with uncertainty, it is misleading to just consider the probabilities attached to single constraints when one wants to assess the probability of satisfying all of them at once.

Example 6. Consider the following ProbDeclare model:



The model indicates that an order can be accepted at most once, and that often (in 80% of the cases) it is actually accepted. In addition, it captures that with probability 0.7 it is true that, whenever the order is accepted, then it is also consequently paid (multiple payment instalments are possible, by simply repeating the execution of pay). Finally, payments are enabled only if the order has been previously accepted.

By looking at the diagram, one could wrongly interpret that in 70% of the cases it is true that the order is accepted and then paid. This is wrong because the `response`(accept, pay) constraint can also be (vacuously) satisfied by a trace that does not contain at all occurrences of accept. A natural question is then: what is the actual probability of observing traces that at some point contain accept and, later on, pay (possibly with other activity occurrences in between and afterward)? The answer is that this happens in half of the cases. To justify this non-trivial answer, one has to apply combined reasoning by considering the interplay of `response`(accept, pay) and `existence`(accept), with their corresponding probabilities. More specifically, `response`(accept, pay) can be satisfied in this model in two different ways:

1. by not executing at all accept;
2. by executing accept (which can be done only once, due to the presence of the crisp `absence2`(accept) constraint) and, later on, at least once pay.

These two situations, which we will call later on *constraint scenarios*, should altogether cover exactly 70% of the traces, as dictated by the constraint probability attached to `response`(accept, pay). The first scenario must have probability 0.2, because in 80% of the traces accept must appear, as dictated by the `existence`(accept) constraint and its associated probability. But then, the second scenario, which is the one we are interested in, has probability $0.7 - 0.2 = 0.5$ (half of the traces in the log). ◁

In the next section, we make the reasoning carried out in the discussed examples more systematic, showing how logical and probabilistic reasoning have to be combined towards a single, combined declarative framework.

4 Reasoning on Time and Probabilities

As we have seen in the previous section, to reason on conjunctions of probabilistic constraints, i.e., on ProbDeclare models, we need to simultaneously take into account the temporal semantics of constraints and their associated probabilities.

Formally, this is done by relying on the probabilistic temporal logic over finite traces PLTL$_f$, recently introduced in [5]. More specifically, probabilistic constraints as defined here have a direct encoding into the fragment PLTL$_f^0$ of PLTL$_f$, also investigated in [5]. We do not delve into the encoding, nor highlight the formal details on how to carry out this combined reasoning. We instead show algorithmically how to accomplish this, noticing that all the algorithmic techniques discussed next are correct thanks to [5]. Again thanks to [5], we also get that, overall, the cost of reasoning on probabilistic constraints has the same complexity of reasoning with standard LTL$_f$ constraints, i.e., PSpace in the length of the constraints (this complexity bound is tight).

In the remainder of this section, we fix a ProbDeclare model $\mathcal{M} = \langle \Sigma, \mathcal{C} \rangle$, where $\mathcal{C}$ is partitioned into *crisp constraints* $\mathcal{C}_{crisp} = \{\langle \varphi, p \rangle \in \mathcal{C} \mid p = 1\}$ and (genuinely) *probabilistic constraints* $\mathcal{C}_{prob} = \{\langle \varphi, p \rangle \in \mathcal{C} \mid p < 1\}$. With a slight abuse of terminology, when we use the term "crisp constraint", we mean a constraint in $\mathcal{C}_{crisp}$, and, when we use the term "probabilistic constraint", we mean a constraint in $\mathcal{C}_{prob}$. We also assume that $\mathcal{C}_{crisp}$ is a consistent Declare model, i.e., crisp constraints are satisfiable altogether. If not, then $\mathcal{M}$ has to be discarded, as it does not admit any conforming trace.

4.1 Constraint Scenarios and Consistency of ProbDeclare Models

While crisp constraints must hold in every possible trace, probabilistic constraints may or may not hold (with a ratio specified by their probability). In addition, recall that when a constraint formula does not hold, then its negation must hold. Consequently, in the most general case, $\mathcal{M}$ is a compact description for the $2^{|\mathcal{C}_{prob}|}$ standard Declare models, each one obtained by considering all constraint formulae in $\mathcal{C}_{crisp}$, and by selecting, for each constraint $\langle \varphi, p \rangle \in \mathcal{C}_{prob}$, whether the constraint formula φ or its complement $\neg\varphi$ is assumed to hold.

We call the so-obtained Declare models *(constraint) scenarios*. To pinpoint a specific scenario, we fix an ordering over $\mathcal{C}_{prob}$, and we denote the scenario with a binary string of length $|\mathcal{C}_{prob}|$, where position number $i \in \{1, \ldots, |\mathcal{C}_{prob}|\}$ has value 1 if the i-th probabilistic constraint in $\mathcal{C}_{prob}$ must hold, 0 otherwise.

Example 7. Consider the ProbDeclare model in Example 6. By fixing the ordering over its probabilistic constraints where $\langle \texttt{existence}(\mathsf{accept}), 0.8 \rangle$ is first and $\langle \texttt{response}(\mathsf{accept}, \mathsf{pay}), 0.7 \rangle$ is second, we have the following 4 scenarios:

1. Scenario 00, where none of the two constraint formulae holds, and is consequently characterized by formula $\Box\neg\mathsf{accept} \wedge \Diamond(\mathsf{accept} \wedge \Box\neg\mathsf{pay})$.
2. Scenario 01, where the `response` constraint formula holds while the `existence` one does not, and so has formula $\Box\neg\mathsf{accept} \wedge \Box(\mathsf{accept} \rightarrow \Diamond\mathsf{pay})$.
3. Scenario 10, where the `existence` constraint formula holds while the `response` one does not, and so has formula $\Diamond\mathsf{accept} \wedge \Diamond(\mathsf{accept} \wedge \Box\neg\mathsf{pay})$.
4. Scenario 11, where both formulae holds ($\Diamond\mathsf{accept} \wedge \Box(\mathsf{accept} \rightarrow \Diamond\mathsf{pay})$). $\triangleleft$

Among the possible scenarios, only those that are logically consistent, i.e., are associated with a satisfiable formula, have to be retained. In fact, inconsistent scenarios do not admit any conforming trace. Obviously, when checking whether the scenario is consistent, its constraint formulae have to be conjoined with those in $\mathcal{C}_{crisp}$.

Example 8. Consider the 4 scenarios of Example 7. Scenario 00 has to be discarded because it is logically inconsistent: its formula $\Box\neg\mathsf{accept} \wedge \Diamond(\mathsf{accept} \wedge \Diamond\mathsf{pay})$ is unsatisfiable (it is asking for the presence and absence of accept). The other three scenarios are instead logically consistent. $\lhd$

Example 9. Consider the ProbDeclare model in Example 4. Also for this model there are 4 scenarios, obtained by considering the two `existence` constraints and their complements. The scenario where both constraints are not satisfied captures those traces where no decision is taken for the order, i.e., the order is not accepted nor rejected. The scenarios where one constraint is satisfied and the other is not account for those traces where a univocal decision is taken for the order. The scenario where both constraints are satisfied, thus requiring acceptance and rejection for the order, is inconsistent, due to the interplay of such constraints and the crisp `not-coexistence` one. This corresponds to the standard Declare model of Example 2. $\lhd$

We have explicitly used the term *logically* (in)consistent scenarios since there is no guarantee that these scenarios are actually plausible. This depends on their corresponding probabilities, which, in turn, are obtained by suitably combining the probabilities of their constitutive constraints in their positive or complemented form. This is done by enforcing the semantics of constraint probability, which requires to ensure the following: for every probabilistic constraint $\langle \varphi, p \rangle$, the sum of the probabilities assigned to those scenarios where φ must hold must be equal to p.

To do so, we construct a system of linear inequalities whose variables represent the probabilities of possible scenarios [5]. We denote such variables as x_s, where s is the boolean string representing the scenario the variable is associated with. By considering a ProbDeclare model $\mathcal{M}$, fixing $n = |\mathcal{C}_{prob}|$ and writing $i \in \{0, \ldots, n-1\}$ in binary, the system of inequalities $\mathcal{L}_{\mathcal{M}}$ is:

$$\begin{array}{rll}
x_i \geq 0 & 0 \leq i < 2^n & (x_i \text{ are probabilities})\\
\sum_{i=0}^{2^n-1} x_i = 1 & & (x_i \text{ are probabilities})\\
\sum_{j\text{th position is } 1} x_i = p_j & 0 \leq j < n & (\text{constraint semantics})\\
x_i = 0 & \text{if scenario } i \text{ is logically inconsistent} &
\end{array}$$

Notably, $\mathcal{L}_{\mathcal{M}}$ combines, at once, the logical and the probabilistic content of $\mathcal{M}$, on the one hand, imposing that the scenario probabilities agree with the constraint probabilities, and, on the other, forcing logically inconsistent scenario to have probability 0.

$\mathcal{L}_{\mathcal{M}}$ may admit: *(i) no solution*, witnessing that $\mathcal{M}$ is inconsistent; *(ii) one solution*, returning the exact probabilities for all the scenarios of $\mathcal{M}$, *(iii) multiple (possibly infinitely many) solutions*, witnessing that different probability distributions can be assigned to the scenarios. To obtain the ranges of probability for each scenario, one can turn the system of inequality into several optimizations problems where each probability variable is minimized and maximized.

It is worth noting that, when $\mathcal{L}_{\mathcal{M}}$ is solvable, its solutions may force some scenario probabilities to be always equal to 0. This witnesses the fact that even a logically consistent scenario may not have any conforming trace due to the interplay of constraint probabilities. We call *plausible* those scenarios that have a probability > 0.

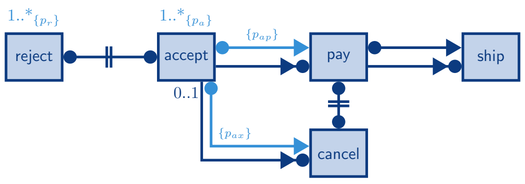
Example 10. Consider again the ProbDeclare model in Example 7 with its 4 scenarios (one of which is logically inconsistent, as discussed in Example 8). The four possible scenarios have corresponding probability variables x_{00}, x_{01}, x_{10} and x_{11}, constrained by the system of inequalities (we omit the fact that all variables are non-negative):

$$\begin{array}{rll}
x_{00} + x_{01} + x_{10} + x_{11} &= 1 & \\
x_{10} + x_{11} &= 0.8 & \text{semantics of } \langle \texttt{existence}(\mathsf{accept}), 0.8\rangle \\
x_{01} \qquad + x_{11} &= 0.7 & \text{semantics of } \langle \texttt{response}(\mathsf{accept}, \mathsf{pay}), 0.7\rangle \\
x_{00} \qquad\qquad\qquad &= 0 & \text{logical inconsistency of scenario 00}
\end{array}$$

The system admits a single solution, with $x_{00} = 0$, $x_{01} = 0.2$, $x_{10} = 0.3$ and $x_{11} = 0.5$, the last matching the informal discussion given in Example 6. ◁

We conclude the section with an informative ProbDeclare model example that combines parts of the examples seen so far to capture a non-trivial fragment of an order-to-shipment process. We use parameters for constraint probabilities, then discussing the impact of grounding such probabilities to different actual values.

Example 11. Consider the following order-to-shipment ProbDeclare model:



To construct the 16 possible scenarios for this model, the following constraints and LTL$_f$ formulae have to be considered:

- $\texttt{existence}(\mathsf{accept})$ with formula $\varphi_a = \Diamond\mathsf{accept}$, and its complement $\Box\neg\mathsf{accept}$;
- $\texttt{existence}(\mathsf{reject})$ with formula $\varphi_r = \Diamond\mathsf{reject}$, and its complement $\Box\neg\mathsf{reject}$;
- $\texttt{response}(\mathsf{accept}, \mathsf{pay})$ with formula $\varphi_{ap} = \Box(\mathsf{accept} \rightarrow \Diamond\mathsf{pay})$, and its complement $\Diamond(\mathsf{accept} \wedge \Box\neg\mathsf{pay})$;

- $\texttt{response}(\mathsf{accept}, \mathsf{cancel})$ with formula $\varphi_{ax} = \Box(\mathsf{accept} \rightarrow \Diamond\mathsf{cancel})$, and its complement $\Diamond(\mathsf{accept} \wedge \Box\neg\mathsf{cancel})$. ◁

Table 2 summarizes the different constraint scenarios, their logical consistency and, in the last column, their probabilities computed by constructing and solving the system of inequalities described above. Table 3 shows instead three different groundings for the constraint probability parameters and their impact on the probabilities of the scenarios. In particular, *Case 1* is so that all the logically consistent scenarios may actually occur, even though with different probabilities. The most likely scenario, accounting for half of the traces, captures the happy path where the order is paid and shipped. *Case 2* assigns a different probability to $\texttt{response}(\mathsf{accept}, \mathsf{cancel})$, causing scenario 1000 to be not plausible anymore, being associated with probability 0; intuitively, the interplay of constraints and their probabilities makes it impossible to just execute accept without taking further activities. Finally, *Case 3* increases the probability of $\texttt{response}(\mathsf{accept}, \mathsf{cancel})$ even more, resulting in an inconsistent model.

Table 2. Constraint scenarios of the ProbDeclare model in Example 11, indicating whether they are logically consistent and, if so, providing the (shortest) conforming trace, and the scenario probability.

Scenario				Logically consistent	Shortest conforming trace	Scenario probability
φ_a	φ_r	φ_{ap}	φ_{ax}			
0	0	0	0	N		
0	0	0	1	N		
0	0	1	0	N		
0	0	1	1	Y	Empty trace	$1 - p_a - p_r$
0	1	0	0	N		
0	1	0	1	N		
0	1	1	0	N		
0	1	1	1	Y	$\langle\mathsf{reject}\rangle$	p_r
1	0	0	0	Y	$\langle\mathsf{accept}\rangle$	$2 - p_a - p_{ap} - p_{ax}$
1	0	0	1	Y	$\langle\mathsf{accept}, \mathsf{cancel}\rangle$	$p_a + p_{ax} - 1$
1	0	1	0	Y	$\langle\mathsf{accept}, \mathsf{pay}, \mathsf{ship}\rangle$	$p_a + p_{ap} - 1$
1	0	1	1	N		
1	1	0	0	N		
1	1	0	1	N		
1	1	1	0	N		
1	1	1	1	N		

Table 3. Three different groundings for the constraint probabilities used in the ProbDeclare model in Example 11, and their impact on the scenario probabilities.

Consistent scenario				Case 1	Case 2	Case 3
φ_a	φ_r	φ_{ap}	φ_{ax}	$p_a = 0.8$ $p_r = 0.1$ $p_{ap} = 0.7$ $p_{ax} = 0.3$	$p_a = 0.8$ $p_r = 0.1$ $p_{ap} = 0.7$ $p_{ax} = 0.5$	$p_a = 0.8$ $p_r = 0.1$ $p_{ap} = 0.7$ $p_{ax} = 0.7$
0	0	1	1	0.1	0.1	Inconsistent
0	1	1	1	0.1	0.1	
1	0	0	0	0.2	0	
1	0	0	1	0.1	0.3	
1	0	1	0	0.5	0.5	

5 Reasoning with Constraint Scenarios

Constraint scenarios can be used to perform a variety of tasks. We focus here on two fundamental ones: conformance checking and probabilistic constraint entailment.

5.1 Conformance Checking

In Declare, the simplest form of conformance checking amounts to check whether a given execution trace satisfies all constraints contained in the model, thus returning a yes/no answer.

In ProbDeclare, this notion can be refined by considering the different constraint scenarios and their probabilities. Let $\mathcal{M} = \langle \Sigma, \mathcal{C} \rangle$ be a ProbDeclare model, and τ be a trace over Σ. The plausible scenarios of $\mathcal{M}$ are pairwise disjoint subsets of the overall set Σ^* of traces over Σ. Disjointness comes from the fact that every pair of plausible scenarios is so that they disagree about at least one constraint, and no trace can conform with both of them. The complement of the traces accepted by the plausible scenarios then characterizes those traces that are not conforming with $\mathcal{M}$. To assess conformance, we can then proceed as follows: (1) Check τ against every plausible scenario of $\mathcal{M}$. (2) If one plausible scenario is so that τ holds there, output *yes* together with the probability (or range of probabilities) attached to that scenario; the scenario probability gives an indication on whether the trace represents a "mainstream" execution of the process, or is instead an outlier behavior. (3) If no such scenario is found, then output *no*.

Example 12. Consider the ProbDeclare model captured by *Case 1* in Table 3. Trace $\langle \mathsf{accept}, \mathsf{cancel}, \mathsf{pay} \rangle$ does not conform with the model, since paying and canceling are mutually exclusive. Trace $\langle \mathsf{accept}, \mathsf{cancel} \rangle$ is instead conforming, as it satisfies scenario 1001. Since this scenario is associated with probability 0.1, the analyzed trace represents an outlier behavior. Finally, trace

$\langle \mathsf{accept}, \mathsf{pay}, \mathsf{ship}, \mathsf{ship}, \mathsf{pay}, \mathsf{ship} \rangle$ represents a mainstream behavior since it conforms with the most likely scenario 1010, with probability 0.5. $\lhd$

5.2 Constraint Entailment

It is well-known that Declare and other declarative process modeling languages have the issue of *hidden dependencies* [7], namely the fact that constraints may interact with each other in subtle ways. This becomes even more complex in the case of probabilistic constraints. In this light, it becomes crucial to be able to ascertain whether a constraint is implied by a given model. Checking constraint implication in Declare is very simple: this simply amounts to check whether the LTL$_f$ formula of the model implies the given constraint. In the case of ProbDeclare, we extend this approach by computing, for a given LTL$_f$ formula, what is the probability with which it is implied by the ProbDeclare model. This is done as follows: (1) Initialize the constraint probability range to $0, 0$. (2) For every plausible scenario, check whether the scenario implies the formula of interest in the classical LTL$_f$ sense; if so, update the constraint probability by summing its minimum and maximum to the minimum and maximum probability associated with the scenario. (3) Return the constraint probability range.

Example 13. Consider again the ProbDeclare model captured by *Case 1* in Table 3. We want to check to what extent the model implies that the order is eventually shipped ($\Diamond \mathsf{ship}$). Shipment only occur if a payment occurs before, and therefore this formula is implied only by scenario 1010, consequently getting a probability of 0.5.

We are also interested in checking to what extent the model implies that the order is not rejected ($\neg\Diamond \mathsf{reject}$). This formula holds in all those scenarios where $\mathtt{existence}(\mathsf{reject})$ is false. Therefore, this formula is implied with probability 0.9.

Finally, consider the LTL$_f$ constraint $\neg(\Diamond \mathsf{cancel} \wedge \Diamond \mathsf{ship})$, expressing the mutual exclusion between cancel and ship. This constraint is implied with probability 1, due to the presence of the two crisp constraints $\mathtt{not\text{-}coexistence}(\mathsf{cancel}, \mathsf{pay})$ and $\mathtt{precedence}(\mathsf{pay}, \mathsf{ship})$, which must hold in every possible scenario (including the plausible ones). $\lhd$

6 Conclusions

We have studied how to enrich constraint-based process models with uncertainty, captured as the probability that a trace will conform with a constraint or not. We have discussed how this impacts the semantics of a constraint model, and how logical and probabilistic reasoning have to be combined to provide core services such as consistency and conformance checking, as well as probabilistic constraint entailment.

Notably, all the techniques presented in this paper can be directly grounded with existing tools: automata-based techniques for LTL$_f$ to carry out logical reasoning, and off-the-shelf systems to solve systems of linear inequalities (and corresponding optimization problems) to handle probabilities.

In [6], beside a concrete implementation of the techniques presented in this paper, we investigate the application of probabilistic business constraints to process mining, not only considering standard problems like discovery, but also delving into online operational support and, in particular, process monitoring [4].

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, Beijing, China, August 3–9, IJCAI 2013, pp. 854–860, AAAI Press (2013). http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997
3. 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
4. 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
5. 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, USA, February 7–12, pp. 10218–10225 (2020). https://aaai.org/ojs/index.php/AAAI/article/view/6583
6. Maggi, F.M., Montali, M., Peñaloza, R., Alman, A.: Extending temporal business constraints with uncertainty. In: Business Process Management - 18th International Conference, BPM 2020, September 13–18. pp. 1–20, Sevilla, Spain (2020). https://doi.org/10.1007/978-3-030-58666-9
7. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. TWEB **4**(1), 1–62 (2010). https://doi.org/10.1145/1658373.1658376
8. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: Proceedings of the 11th - IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15–19 October, pp. 287–300. Annapolis, USA (2007). https://doi.org/10.1109/EDOC.2007.14