Probabilistic Compliance in Declarative Process Mining

Probabilistic Compliance in Declarative Process Mining

Michela Vespa[1,*], Elena Bellodi[1], Federico Chesani[2], Daniela Loreti[2], Paola Mello[2], Evelina Lamma[1] and Anna Ciampolini[2]

[1]Dipartimento di Ingegneria, Università di Ferrara, Via Saragat 1, Ferrara, Italy
[2]Dipartimento di Informatica - Scienza e Ingegneria, Viale Risorgimento 2, Bologna, Italy

Abstract

When addressing real-world processes, it is crucial to account for their intrinsic uncertainty to better reflect the nature of such processes. In this work, we introduce the concept of Probabilistic Declarative Process Specification (PDS), which is based on the Distribution Semantics from Probabilistic Logic Programming, in order to describe declarative process models with both crisp and probabilistic constraints. The probability associated to a constraint represents its strength or importance in a specific process domain. From this, we propose a novel notion of probabilistic compliance of a process trace w.r.t. a PDS, and how to compute it with an existing algorithm. Preliminary experimental results on a healthcare protocol are presented to evaluate the feasibility of our proposed semantics on process conformance checking.

Keywords

Process Mining, Declarative language, Compliance, Distribution Semantics, Probabilistic Logic Programs

1. Introduction

In the Business Process Mining community, many different languages have been proposed to model/describe a process, capturing the many aspects of a work process, ranging from the resource perspective, to the artifact-based and data perspective, up to the control-flow aspects. The latter viewpoint in particular has been subject to intense research activity. Two families of process modeling languages emerged: procedural approaches, such as BPMN[1], and declarative ones, such as DECLARE [1, 2] and DCR Graphs [3]. Declarative approaches have been introduced to overcome some forms of *rigidness* derived by procedural ones, with the aim of finding a balance between *flexibility* and *control*. DECLARE in particular, was defined with this aim: allowing the specification of which properties a process instance should exhibit, without specifying the exact steps to achieve them. Since the initial proposal, DECLARE has been equipped with a formal semantics based on a strict subset of LTL_f. This enables a straightforward specification of process properties through the use of established formulas.

However, the adoption of a logic-based semantics has also raised a practical issue when dealing with the problem of evaluating if a log is compliant with a process specification. On one side, a log is composed of many traces; on the other side a DECLARE process specification is composed of many constraints. Typically, in real-world applications, different subsets of traces are compliant with different subsets of constraints. The direct consequence is that a logic-crisp notion of compliance might be difficult or unsatisfactory to identify. In turn, such difficulty could undermine the capacity of a process specification to properly describe the process. In a seminal work [4], the authors tackled the problem by introducing the notion of probabilities into the DECLARE formalism: each constraint is assigned a probability (and a relational operator). A probabilistic constraint is satisfied over the log if the number

PMAI@ECAI24: International ECAI Workshop on Process Management in the AI era, October 19, 2024, Santiago De Compostela, Spain

*Corresponding author.
†These authors contributed equally.
michela.vespa@unife.it (M. Vespa); elena.bellodi@unife.it (E. Bellodi); federico.chesani@unibo.it (F. Chesani); daniela.loreti@unibo.it (D. Loreti); paola.mello@unibo.it (P. Mello); evelina.lamma@unife.it (E. Lamma); anna.ciampolini@unibo.it (A. Ciampolini)
0009-0004-4350-8151 (M. Vespa); 0000-0002-3717-3779 (E. Bellodi); 0000-0003-1664-9632 (F. Chesani); 0000-0002-6507-7565 (D. Loreti); 0000-0002-5929-8193 (P. Mello); 0000-0003-2747-4292 (E. Lamma); 0000-0002-9314-1958 (A. Ciampolini)

[1]https://www.bpmn.org/

of traces satisfying the constraint over the log cardinality achieves the mathematical relation established by the relational operator and the probability assigned. As a practical example, a constraint c_1 with assigned probability $(0.9, \geq)$ is satisfied if the number of traces complaint with c_1 is at least ($\geq$) the 90% of the total traces. In summary, probabilities express the relative frequency of traces, and are employed with such a meaning for the tasks of discovering, monitoring and conformance checking. Nonetheless, when the available log does not represent the whole process domain, the application of the approach [4] by domain expert might be challenging.

In this work, we also tackle the problem of probabilistic conformance checking with DECLARE constraints, but we propose a different semantics underlying the probabilistic constraints. We introduce the concept of Probabilistic Declarative Process Specification (PDS) starting from Probabilistic Logic Programming (PLP) and the Distribution Semantics [5]. A PLP program under that semantics defines a probability distribution over normal logic programs called worlds. In a PLP program, logic formulas are equipped with a probability, which is interpreted as the probability of them appearing or not in a normal logic program. Similarly, the probability of a DECLARE constraint is treated as the probability that such constraint will appear (or not) in a possible process specification. The presence or not of a constraint in a process specification tells us how "important" or strong that constraint is. The stronger or more important the constraint, the greater its probability. According to the our semantics, all worlds including one such constraint "inherit" its probability, while those not including the constraint take into account the complement to 1 of its probability. Differently from [4], this probability does not represent a fraction of traces satisfying the constraint, but allows one to model uncertainty in the domain by means of the probability attached to the constraint.

To better clarify our proposal, let us consider the following university course scenario, described with two constraints: (a) it is mildly advised that a student only take the final examination if she has attended all the classes; (b) it is strongly advised that a student attend a practical session only if she attended the corresponding preparatory class. DECLARE provides the way for expressing such constraints; however, there is no way to specify that rule (a) is "mildly advised", while rule (b) is "strongly advised". Our proposal consists of attaching probabilities to constraints (a) and (b): such values would indicate the probability that each rule would end up in a possible model (process specification), and the probability attached to (b) would be higher than the one attached to (a). By making (a) and (b) probabilistic, we would find four possible process models, and we can formally define the compliance of the students' careers (process traces) towards these four models (i.e., worlds in the Distribution Semantics terminology).

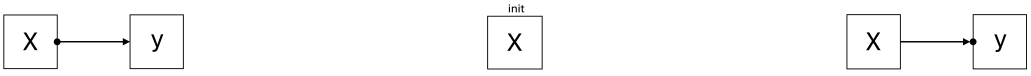
The contribution of this work is the following: we introduce a semantics for probabilistic declarative process specifications, provide a novel notion of compliance, and ground its application on a health protocol. Then, we exploit previous results, and in particular (i) the mapping provided to DECLARE in terms of the SCIFF modeling language [6] and (ii) the extension of the Distribution Semantics to Abductive Logic Programming (ALP) in the SCIFF Framework [7]. Thanks to the existing work, we can implement our proposed semantics and compute a compliance probability. In order to evaluate the feasibility of our approach, we report some preliminary experimentation over the clinical guideline.

The paper is organized as follows: Section 2 introduces background on DECLARE and the distribution semantics, Section 3 describes the proposed semantics, Section 4 describes how we perform probabilistic compliance. Section 5 presents first experimental results and Section 6 concludes the paper.

2. Background

2.1. Traces, logs, and the DECLARE modeling language

In the BPM setting [8] the starting point is the observation of a process execution in terms of the execution of the activities that compose the process. Each execution of the process is usually referred to as a *process instance* or *trace*, where the executions of the distinct activities are referred to as distinct *activity instances* or *activity executions*. Each activity is characterized by at least a name, and each activity instance is usually denoted with a temporal timestamp. The timestamp provides, within the



(a) The response template. (b) The init template. (c) The precedence template.

Figure 1: Examples of the DECLARE graphical notation: x and y are placeholders that should be substituted with proper activity names.

same trace, a relation order between the different activity executions: it is often the case that traces are represented directly as a sequence of activity names, ordered on the base of the timestamp. Depending on the specific context, the timestamp can be omitted: this is also our choice.

Formally, we consider the existence of a finite alphabet of symbols Σ corresponding to the activity names. A trace and a log then are defined as follows:

Definition 1 (Trace t and Log $\mathcal{L}$)**.** *Given a finite set Σ of symbols (i.e., activity names), a* trace t *is a finite, ordered sequence of symbols over Σ, i.e. $t \in \Sigma^*$, where the latter is the infinite set of all the possible finite sentences over Σ. A* log $\mathcal{L}$ *is a finite set of traces.*

In a process log, each trace t represents a different process instance. Different process instances may have the same trace, although referring to different executions.

Example 1. *Let us suppose that a process is made of activities a, b, c, and d. $\Sigma = \{a, b, c, d\}$. An example of a log might be: $\mathcal{L} = \{t_1 = \langle a, b, c\rangle, t_2 = \langle a, b, a, d\rangle, t_3 = \langle a, a, d\rangle, t_4 = \langle a, b, c\rangle, t_5 = \langle a, b, c\rangle\}$.*

The DECLARE modeling language was initially introduced by [2] and subsequently studied in [9]. Aimed to overcome the "rigidness" of procedural modeling languages, it focuses on modeling processes by specifying *what* are the relevant properties that a process instance should exhibit, without specifying *how* these properties should be achieved. To this end, DECLARE models a process in terms of *constraints*, sort of rules about activities that can appear in a process execution (i.e., in a trace), with qualitative temporal relations among these activities. A simple example of a DECLARE constraint is response(a,b), meaning that every occurrence of (the execution of) activity a in a trace should be followed by the occurrence of an activity b.

DECLARE provides a finite set of constraint patterns, whose arguments should be properly instantiated with the activity names peculiar to the specific process to be modeled. For example, a common pattern is response(x,y) with x and y placeholders to be substituted with proper activities. Another pattern is init(x), that specifies that every process instance should always start with the execution of activity x. Another common pattern is precedence(x,y), that states that every occurrence of the execution of y should be preceded by the execution of x.

DECLARE provides a graphical representation for the patterns (see Figure 1 for a few examples), and has been equipped with two different formal semantics, both based on logic formalisms. In the original formulation in [2] the semantics was given using the *LTL* temporal logic; subsequent works have shown the feasibility of adopting the LTL_f logic [10]: for a recent recap see [11]. A second formal semantics has been proposed in [6], where the SCIFF language and ALP [12] has been exploited.

Both the semantics exploit the idea that each DECLARE template can be mapped onto one (or more) logic formula φ, and that *logical entailment* can be used to define the notion of *compliance/violation* of a trace t w.r.t. to a constraint formula φ. With the aim of being the most general, we provide an intuitive definition of compliance/violation, where the meaning of the entailment symbol $\models$ should be referred to the chosen semantics (LTL_f or ALP).

Definition 2 (Compliance/violation of a trace versus a constraint)**.** *A trace t is* compliant *with a DECLARE constraint if, named φ the corresponding logic formula modelling that constraint, it holds: $t \models \varphi$.*

A trace t violates *a DECLARE constraint if it does not entail the corresponding formula φ, i.e. if $t \not\models \varphi$.*

Definition 3 (Declarative Process Specification, from [11])**.** *A* declarative process specification *is a tuple DS=(REP, Σ, C) where:*

- *REP is a finite non-empty set of* templates*, where each* template *is a predicate* $c(x_1, \ldots, x_m) \in$ *REP on variables* $x_1, \ldots, x_m$ *(with* $m \in \mathbb{N}$ *the arity of* c*);*
- Σ *is a finite non-empty set of* activity names*;*
- C *is a finite set of* constraints*, obtained by instantiating templates from REP to* Σ*; we will compactly denote such constraints with* $c(a_1, \ldots, a_m)$*,* $a_1, \ldots, a_m \in \Sigma$*.*

Usually the constraints $c(a_1, \ldots, a_m)$ in a DS are considered as being in *logical conjunction.* The notion of compliance can be then lifted from a trace vs. a constraint towards a trace vs. a DS as follows:

Definition 4 (Compliance of a trace versus a Declarative Process Specification)**.** *A trace is compliant with a DS if it entails the conjunction of the formulas* φ_i *corresponding to the* $c_i \in C$*:*

$$t \models \varphi_1 \wedge \ldots \wedge \varphi_m$$

where m *is the cardinality of* C*.*

Example 2. *Let us consider the log introduced in Example 1, and the following DS (REP and* Σ *are omitted for the sake of simplicity):*

$$C = \{ \quad c_1 = \textit{response(a,b)} \\ \qquad\quad c_2 = \textit{init(a)} \quad \}$$

Even without considering the corresponding formal semantics, we can notice that:

- t_1*,* t_4*, and* t_5 *are compliant with* c_1*;*
- t_2 *is not compliant with* c_1 *because the second occurrence of activity a is not followed by an occurrence of activity b;*
- t_3 *is not compliant with* c_1 *because two occurrences of a are not followed by an occurrence of b;*
- $t_1, \ldots, t_5$ *are all compliant with* c_2*.*

2.2. Distribution Semantics and Probabilistic Logic Programming

Probabilistic Logic Programming has recently received an increasing attention for its ability to incorporate probability in logic programming. Among various proposals for PLP, the one based on the distribution semantics [5] has gained popularity as being introduced for the PRISM language [5] but is the basis for many other languages such as Probabilistic Logic Programs [13], Probabilistic Horn Abduction (PHA) [14], Independent Choice Logic (ICL) [15], pD [16], Logic Programs with Annotated Disjunctions (LPADs) [17], ProbLog [18] and CP-logic [19]. Such semantics is particularly appealing for its intuitiveness and because efficient inference algorithms were proposed. By combining probability with logic programming, PLP languages are a suitable framework to handle uncertain information.

A program in one of these languages defines a probability distribution over normal logic programs called *worlds*. The distribution semantics has been defined both for programs that do not contain function symbols, and thus have a finite set of worlds, and for programs that contain them, that have an infinite set of worlds. We consider here the first case for the sake of simplicity, for the treatment of function symbols see [20]. A survey of the distribution semantics in PLP can be found in [21]. In the following, the distribution semantics will be described with reference to LPADs. Formally, an LPAD consists of a finite set of "annotated disjunctive clauses", where the head is a disjunction in which each atom is annotated with a probability. If the body holds true, only one of the atoms in the head will be true with the associated probability. An annotated disjunctive clause R_i is of the form

$$h_{i1} : p_{i1}; \ldots; h_{in_i} : p_{in_i} :- b_{i1}, \ldots, b_{im_i},$$

where $h_{i1}, \ldots, h_{in_i}$ are logical atoms and $\{p_{i1}, \ldots, p_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} p_{ik} \leq 1$; $b_{i1}, \ldots, b_{im_i}$ is indicated with $body(R_i)$. If $\sum_{k=1}^{n_i} p_{ik} < 1$, the head implicitly

contains an extra atom $null$ that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} p_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

An *atomic choice* [15] is a triple (R_i, θ_j, k) where $R_i \in T$, θ_j is a substitution that grounds R_i and $k \in \{1, \ldots, n_i\}$ identifies one of the head atoms. (R_i, θ_j, k) means that, for the ground clause $R_i\theta_j$, the head h_{ik} was chosen. A set of atomic choices κ is *consistent* if only one head is selected from the same ground clause; we assume independence between the different choices. A *composite choice* κ is a consistent set of atomic choices [15]. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the independent atomic choices, i.e. $P(\kappa) = \prod_{(R_i,\theta_j,k)\in\kappa} p_{ik}$.

A *selection* σ is a composite choice that, for each clause $R_i\theta_j$ in $ground(T)$, contains an atomic choice (R_i, θ_j, k). Let us indicate with S_T the set of all selections. A selection σ identifies a normal logic program w_σ defined as $w_\sigma = \{(h_{ik} \leftarrow body(R_i))\theta_j | (R_i, \theta_j, k) \in \sigma\}$. w_σ is called a (possible) *world* of T. Since selections are composite choices, we can assign a probability to worlds: $P(w_\sigma) = P(\sigma) = \prod_{(R_i,\theta_j,k)\in\sigma} p_{ik}$.

We denote the set of all worlds of T by W_T. $P(W_T)$ is a probability distribution over worlds, i.e., $\sum_{w\in W_T} P(w) = 1$. A composite choice κ identifies a set of worlds $w_\kappa = \{w_\sigma | \sigma \in S_T, \sigma \supseteq \kappa\}$. Similarly we can define the set of possible worlds associated to a set of composite choices K: $W_K = \bigcup_{\kappa\in K} w_\kappa$.

Example 3. *Consider the following LPAD T encoding the effect of flu and hay fever on the sneezing symptom.*

(R_1) $strong\_sneezing(X) : 0.3; moderate\_sneezing(X) : 0.5 : -flu(X).$
(R_2) $strong\_sneezing(X) : 0.2; moderate\_sneezing(X) : 0.6 : -hay\_fever(X).$
(R_3) $flu(bob).$
(R_4) $hay\_fever(bob).$

If somebody has the flu or hay fever, there is the possibility that he experiences sneezing symptoms with different intensity: if she has the flu, then she might show strong sneezing with probability 0.3, or moderate sneezing with probability 0.5; similarly for the second clause. She might not experience any symptom with probability 0.2 in both cases. We know for sure that Bob has both the flu and hay fever. T *has* $3 \cdot 3 = 9$ *worlds, as each probabilistic clause has one grounding* $\theta_1 = \{X/bob\}$*. Worlds are shown in Table 1.*

Table 1
Worlds for Example 3. The probabilities of the worlds sum up to 1.

World id	World	$P(w)$
w_1	strong_sneezing(bob):-flu(bob). strong_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.3×0.2 = 0.06
w_2	strong_sneezing(bob):-flu(bob). moderate_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.3×0.6=0.18
w_3	strong_sneezing(bob):-flu(bob). flu(bob). hay_fever(bob).	0.3×0.2=0.06
w_4	moderate_sneezing(bob):-flu(bob). strong_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.5×0.2=0.1
w_5	moderate_sneezing(bob):-flu(bob). moderate_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.5×0.6=0.3
w_6	moderate_sneezing(bob):-flu(bob). flu(bob). hay_fever(bob).	0.5×0.2=0.1
w_7	strong_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.2×0.2=0.04
w_8	moderate_sneezing(bob):-hay_fever(bob). flu(bob). hay_fever(bob).	0.2×0.6=0.12
w_9	flu(bob). hay_fever(bob).	0.2×0.2=0.04

Given a goal G, its probability $P(G)$ can be defined by marginalizing the joint probability of the goal and the worlds:

$$P(G) = \sum_{w\in W_T} P(G, w) = \sum_{w\in W_T} P(G|w)P(w) = \sum_{w\in W_T : w\models G} P(w) \quad (1)$$

The probability of a goal G given a world w is $P(G|w) = 1$ if $w \models G$ and 0 otherwise. $P(w) = P(w_\sigma) = P(\sigma)$, i.e. is the product of the annotations p_{ik} of the atoms selected in σ. Therefore, the probability of a goal G can be computed by summing the probability of the worlds where the goal is true. In practice, given a goal to solve, it is unfeasible to enumerate all the worlds where G is entailed. Inference algorithms, instead, find *explanations* for a goal: a composite choice κ is an *explanation* for G if G is entailed by every world of w_κ. In particular, algorithms find a covering set of explanations w.r.t. G, where a set of composite choices K is *covering* with respect to G if every program $w_\sigma \in W_T$ in which G is entailed is in w_K.

3. Probabilistic Declarative Process Specifications under the Distribution semantics

In this Section, we propose a semantics for Declarative Process Specifications that is highly inspired by the Distribution Semantics introduced in Section 2.2. First of all, we introduce the notion of *probabilistic constraint*. Please note that also in Definition 8 of [4] it is adopted the term *probabilistic constraint*, but that definition includes also a relational operator that we do not need in our semantics.

Definition 5 (Probabilistic Constraint)**.** *Given a finite, non-empty set* REP *of constraint templates and a set* Σ *of activity names* $a_1, \ldots, a_m$*, a* probabilistic constraint *is a constraint template* $c \in$ REP *instantiated over* Σ *that is annotated with a real number* $p \in [0..1]$ *(the* probability *of the constraint). We will indicate a generic* i*-th probabilistic constraint with the syntax:*

$$p_i :: c_i(a_1, \ldots, a_m)$$

The probability is to be interpreted as the strength or importance of c_i. The stronger or more important the constraint, the greater its probability. By applying the Distribution Semantics, a constraint $c_i(\ldots)$ will have a probability p_i of being part of a *Probabilistic Declarative Process Specification*, defined as follows:

Definition 6 (Probabilistic Declarative Process Specification)**.** *A* Probabilistic Declarative Process Specification *PDS is a Declarative Process Specification DS where each constraint* $c_i \in DS$ *is a probabilistic constraint.*

We will refer to constraints annotated with probability $p_i = 1$ as *crisp* constraints. If all the constraints have probability equal to 1, we simply end up having a Declarative Specification as in Def. 3. Probabilities will be omitted when they are equal to 1. We now propose to interpret the probability p_i of a constraint borrowing a few concepts from Section 2.2.

Definition 7 (Atomic Choice, and its probability)**.** *An* atomic choice *over a probabilistic constraint* $p_i :: c_i$ *is a couple* (c_i, e) *where* $e \in \{0, 1\}$*.* e *indicates whether* c_i *is chosen to be included in a PDS with probability* p_i *(e = 1), or not with probability* $1 - p_i$ *(e = 0).*

Note that here we do not need the substitution θ as in the definition of atomic choice in subsection 2.2, since, according to Def. 5, the constraint c_i is ground.

Definition 8 (Composite Choice, and its probability)**.** *A set of atomic choices* κ *is* consistent *if there is only one choice in* κ *for each probabilistic constraint* c_i*; we assume independence between the different atomic choices. A* composite choice κ *is a consistent set of atomic choices.*

The probability $P(\kappa)$ of a composite choice κ *is* $P(\kappa) = \prod_{(c_i,1)\in\kappa} p_i \prod_{(c_i,0)\in\kappa}(1 - p_i)$*, where* p_i *is the probability associated with* c_i*.*

Definition 9 (Selection σ over a PDS, and its probability)**.** *A* selection σ over a Probabilistic Declarative Process Specification *is a composite choice that contains an atomic choice* (c_i, e) *for every probabilistic constraint of the PDS. A selection* σ *identifies a world in this way:* $w_\sigma = \{c_i | (c_i, 1) \in \sigma\}$*. The probability of a world* w_σ *is* $P(w_\sigma) = P(\sigma) = \prod_{(c_i,1)\in\sigma} p_i \prod_{(c_i,0)\in\sigma}(1 - p_i)$*.*

Definition 10 (Probability of a DS)**.** *Given the set of all selections S_T over a Probabilistic Declarative Process Specification, every world w_{σ_i} identified by a selection $\sigma_i \in S_T$ is a different Declarative Process Specification DS_i, and its probability is $P(DS_i) = P(w_{\sigma_i}) = P(\sigma_i)$.*

A PDS defines a probability distribution over regular (non-probabilistic) DSs that correspond to worlds. Let us consider the following example to better illustrate this concept.

Example 4. *Let us consider the following PDS, obtained as an extension of the DS in Example 2:*

$$C = \{ \quad 0.9 :: \textit{response(a,b)} \quad (c_1)$$
$$\quad\quad 0.8 :: \textit{init(a)} \quad (c_2) \quad \}$$

The probability of c_1 indicates that the constraint response(a,b) is considered very strong therefore, the more traces satisfy it, the better. The probability of c_2 indicates that the fact that a trace starts with the execution of activity a is rather important, but less than the fact that a is followed by b. For each constraint there are 2 different atomic choices, and this leads to 4 selections and 4 possible worlds; in turn, 4 different regular DS are possible, each one corresponding to a world:

Selection	DS	$P(DS_i)$
$\sigma_1 = \{(k_1, 1), (k_2, 1)\}$	$DS_1 = \{\text{response(a,b), init(a)}\}$	$P(DS_1) = 0.9 \times 0.8 = 0.72$
$\sigma_2 = \{(k_1, 1), (k_2, 0)\}$	$DS_2 = \{\text{response(a,b)}\}$	$P(DS_2) = 0.9 \times 0.2 = 0.18$
$\sigma_3 = \{(k_1, 0), (k_2, 1)\}$	$DS_3 = \{\text{init(a)}\}$	$P(DS_3) = 0.1 \times 0.8 = 0.08$
$\sigma_4 = \{(k_1, 0), (k_2, 0)\}$	$DS_4 = \{\quad\}$	$P(DS_4) = 0.1 \times 0.2 = 0.02$

Example 4 allows us to highlight some characteristics of the semantics proposed in this paper. First of all, we might notice that, thanks to the notion of *world*, we started from a probabilistic declarative specification, and ended up with a set of non-probabilistic declarative specifications. Secondly, we might notice that each different DS_i has its own probability (i.e., the one obtained by the corresponding selection σ_i), and that the set of all the possible selections achieves a probability distribution over DSs, i.e. $\sum_i P(DS_i) = 1$.

4. Probabilistic Compliance

In this Section we will define the notion of *compliance* of a trace w.r.t. a Probabilistic Declarative Process Specification PDS. In doing so, we will heavily rely on the simpler notion of compliance of a trace vs. a DS as in Definition 4. Note that in turn Def. 4 builds upon Def. 2, that abstracts away from the formal DECLARE semantics (LTL_f or ALP). As a consequence, our definition of compliance towards a probabilistic specification is general, and is valid for both LTL_f and ALP. Obviously, in order to compute the compliance, one semantics should be chosen: we will discuss such a choice and the implementation in the next Section.

Definition 11 (Compliance of a trace versus a PDS)**.** *Given a a Probabilistic Declarative Process Specification PDS, let us consider all the possible selections σ_i over PDS, and all the possible Declarative Specification DS_i associated with σ_i. A trace t has a probability of compliance w.r.t. a PDS defined as follows:*

$$comp(t, PDS) = \sum_{\sigma_i \,:\, t \text{ is compliant with } DS_i} P(DS_i) \tag{2}$$

We might observe that, w.r.t. Def. 4, we move forward from a crisp boolean concept towards a *degree* of compliance: this is an expected consequence of introducing probabilities in the specification. Then, we might notice that the formula above indeed is simply an application of Equation 1 to the process specification domain. Let us apply the notion of compliance of a trace vs. a PDS by considering the following example:

Example 5 (continued from Ex. 4). *Consider the PDS of Ex. 4 and the trace $t_1 = \langle a, b, c \rangle$: t_1 is compliant with all 4 DS_i and, as expected, its probability of compliance is:*

$$comp(t_1, PDS) = P(DS_1) + P(DS_2) + P(DS_3) + P(DS_4) = 0.72 + 0.18 + 0.08 + 0.02 = 1$$

Let us consider then a trace $t_2 = \langle c, a, b \rangle$ and suppose temporarily that c_1 and c_2 are crisp constraints: t_2 is compliant with c_1 since after a there is b; however it is not compliant with c_2 since t_2 does not start with a. By considering again the constraint as probabilistic, out of the four declarative specifications DS_i, t_2 is compliant with DS_2 and DS_4, i.e. those specifications that do not contain c_2. t_2 probability of compliance is:

$$comp(t_2, PDS) = P(DS_2) + P(DS_4) = 0.18 + 0.02 = 0.2$$

By violating c_2, which was a rather important constraint as indicated by a probability value of 0.8, the trace has a low degree of compliance vs. the PDS.

Finally, let us consider the trace $t_3 = \langle c, a \rangle$. t_3 is not compliant with c_1 nor with c_2, considered individually, however it is compliant with the empty specification DS_4, so its probability of compliance is not zero but is equal to $P(DS_4) = 0.02$.

Example 5 illustrates the intuition behind the idea, proposed in this work, of interpreting the probability of a constraint as its *strength* or, in other words, how much *important* it is to satisfy such constraint. In the aforementioned example, trace t_1 is compliant with both the constraints, hence its degree of compliance is the maximum possible. Trace t_2 instead is compliant with only one of the two constraints, hence its degree is lower in relation to the strength/importance we have associated to the violated constraint. We might finally notice that trace t_3, even if it does not comply with any constraints, still achieves a score. This is a direct consequence of the interpretation of the probability of a constraint as its strength: if the strength p_i is lower than 1, then we are saying that we will admit non-compliant traces with a degree $(1 - p_i)$.

In more formal terms, if all the constraints in a PDS have the attached probabilities $p_i < 1$, then there is always a selection σ^* corresponding to including no constraint in the DS^* (the empty specification), and whose probability is $P(\sigma^*) > 0$. In Example 4, such a special selection is σ_4. Every possible trace will be always compliant with DS^*, since it's empty. Similarly, all the constraints annotated with probability $p_i = 1$ will appear in all the corresponding non-probabilistic DS, as shown in the following example:

Example 6. *Let us consider the following PDS:*

$$C = \{ \quad 0.8 :: \textit{response(a,b)} \quad (c_1)$$
$$\textit{init(a)} \quad (c_2) \quad \}$$

The corresponding selections and specifications would be:

Selection	DS	$P(DS_i)$
$\sigma_1 = \{(c_1, 1)\}$	$DS_1 = \{\text{response(a,b)}, \text{init(a)}\}$	$P(DS_1) = 0.8$
$\sigma_2 = \{(c_1, 0)\}$	$DS_2 = \{\text{init(a)}\}$	$P(DS_2) = 0.2$

Example 6 clearly illustrates how our proposed semantics for PDS accommodates for both probabilistic and non-probabilistic constraints: the latter are treated as mandatory constraints as usual in Process Mining, and the derived specifications DS_i will always contain them.

5. Implementation and Evaluation

For computing the probabilistic compliance of a trace vs a PDS we leveraged on the implementation presented in [22]. This implementation supports reasoning on ALP programs featuring "integrity

constraints" similar to those offered by IFF [23], extended with the possibility of annotating them with a probability value. The semantics of these programs, called IFF^{Prob}programs, defines a probability distribution over IFF programs inspired by the distribution semantics, so the IFF^{Prob}implementation could be directly used for our goals. IFF^{Prob}is based on the implementation of the SCIFF proof-procedure. SCIFF [12] is an extension of the IFF proof procedure that also features constraints (*à la* CLP) and universally quantified abducibles. In [22], the IFF sub-language is extended to the probabilistic case with a new CHR constraint that represents the current explanation $(expl, P)$ meaning that, in the current derivation branch, the explanation is $expl$ and has probability P. $Expl$ is a collection of couples (c_i, e), holding the constraint c_i and the Boolean value e indicating whether c_1 belongs to $expl$ or not.

In [9] DECLARE constraints were mapped into a rule-based language known as CLIMB (Computational Logic for the verIfication and Modeling of Business constraints), a specialized subset of the SCIFF language that is based on abductive semantics. This mapping allows declarative models to be converted into sets of executable, logic-based rules. These rules are referred to as integrity constraints since they constrain the courses of interaction to ensure their integrity and compliance with interaction protocols. Each atomic DECLARE activity can be expressed in CLIMB as a *term* and is linked to a single time point, allowing its execution to be represented as either *happened* (H), *expected* (E), or *forbidden*. For example, the fact that an atomic activity a has happened at time T can be denoted by $H(\text{exec}(a), T)$, while $E(\text{exec}(a), T)$ states that a is expected to occur at time T. We do not consider here forbidden activities. Integrity constraints are of the form $Body \rightarrow Head$, where $Body$ contains (a conjunction of) happened events, together with constraints on their variables, as well as Prolog predicates; $Head$ contains (a disjunction of conjunctions of) positive and negative expectations, together with constraints and Prolog predicates, applied on their variables and/or on variables contained in the $Body$.

We created a DECLARE process model for elective colorectal surgery based on the ERAS® (Enhanced Recovery After Surgery) Society guidelines for perioperative care [24], as shown in Figure 2. The model is a PDS written in the CLIMB language, where the protocol activities are subject to both crisp and probabilistic constraints: the former represent *strongly recommended* practices according to the guidelines, reflecting critical, evidence-supported activities that must be adhered to rigorously, while the latter represent *weakly recommended* practices, whose associated probability states the importance to apply them in the healthcare flow. Weakly recommended practices may have some degree of flexibility in their implementation due to individual patient needs, local variations in resources and demographics, advancements in medical research, institutional policies, and cultural considerations. The model includes 21 constraints capturing essential perioperative events, from patient admission to post-surgery recovery. Of these, 14 constraints were identified as crisp ($p_i = 1$) and 7 as probabilistic. An excerpt of the PDS is:

$$\text{true} \rightarrow E(\text{exec}(\text{program\_admission}), 0). \tag{c_1}$$

$$H(\text{exec}(\text{program\_admission}), T_1) \rightarrow E(\text{exec}(\text{counseling}), T_2) \wedge T_2 > T_1. \tag{c_2}$$

$$H(\text{exec}(\text{start\_surgery}), T_2) \rightarrow E(\text{exec}(\text{counseling}), T_1) \wedge T_1 < T_2. \tag{c_3}$$

$$0.3 :: H(\text{exec}(\text{start\_surgery}), T_2) \rightarrow E(\text{exec}(\text{preanesthesia}), T_1) \wedge T_1 < T_2. \tag{c_4}$$

$$0.4 :: H(\text{exec}(\text{start\_surgery}), T_2) \rightarrow E(\text{exec}(\text{fasting}), T_1) \wedge T_1 < T_2. \tag{c_5}$$

Here, the integrity constraint c_1 represents the init constraint. It specifies that the process must always start with the event *program admission* of the patient, occurring at time 0. Obviously, it is a mandatory constraint. Constraint c_2 represents the response constraint, indicating that whenever the event *program admission* happens, the pre-operative patient *counseling* and education must eventually follow at a time T2 later than T1. It's strongly recommended to begin patient counseling after admission into the program and prior to surgical procedures, thus constraints c_2 and c_3 are treated as crisp. Instead, c_4 and c_5 are modeled as probabilistic precedence constraints since, in the guidelines, *preanesthesia* and *fasting* before surgery could sometimes be skipped. The relatively low probability assigned to them derives from the consensus around those specific perioperative practices, suggesting they are generally not enforced for optimal postoperative outcomes. Each event trace was then adapted to the format required by the algorithm, transforming each trace into an interpretation. Every fact has two

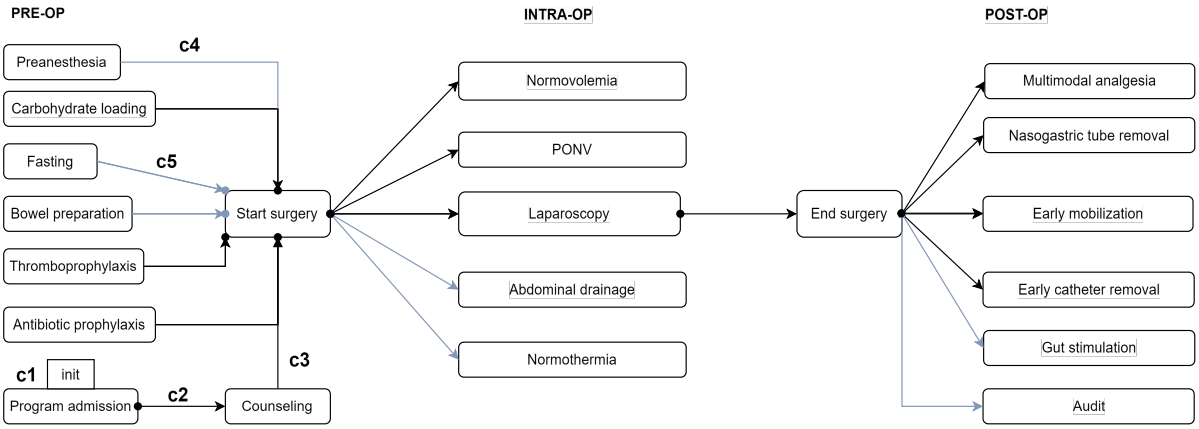


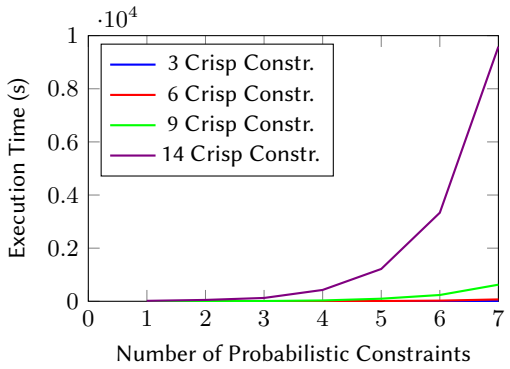
Figure 2: DECLARE model of the ERAS® protocol for colorectal surgery across the pre-operative (PRE-OP), intra-operative (INTRA-OP), and post-operative (POST-OP) phases. Strong recommendations are shown in black, while weak ones in gray.

arguments: the first one is a ground term that records the event name, and the second one is an integer that indicates the timestamp. An example of an interpretation is the following:

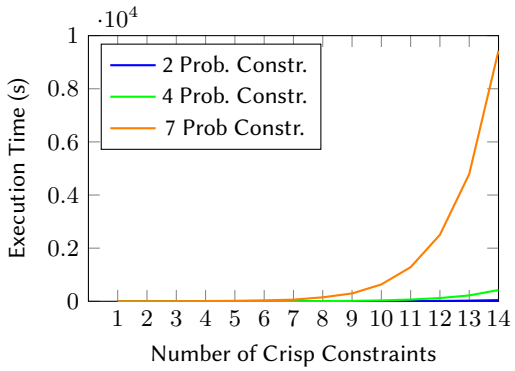
$$hap(exec(program\_admission), 0).$$
$$hap(exec(counseling), 5).$$
$$hap(exec(fasting), 80).$$

To perform our experiments we wrote one synthetic trace composed of 21 events, one for each activity of the protocol. The process trace satisfied all the model constraints.

We provide two preliminary scalability tests in order to show how both crisp and probabilistic constraints influence the execution time in computing the probability of compliance of that trace. Experiments were carried out on a Linux machine with Intel® Xeon® E5-2630v3 running at 2.40 GHz with 20 GB of RAM and a time limit of 8 hours. In the first test, the number of crisp constraints was kept fixed respectively to 3, 6, 9, and 14 and the number of probabilistic constraints was increased from 1 to 7 at steps of 1. In the second test, the number of probabilistic constraints was kept fixed at 2, 4, and 7 and the number of crisp constraints was increased from 1 to 14 at steps of 1.



(a) Execution time as the number of probabilistic constraints varies, keeping the number of crisp constraints fixed.



(b) Execution time as the number of crisp constraints varies, keeping the number of probabilistic constraints fixed.

Figure 3: Scalability analysis of the computation of the probability of compliance of a trace vs. a PDS with varying numbers of crisp and probabilistic constraints.

Figure 3 illustrates the results, showing an exponential trend in execution times as the number of either crisp or probabilistic constraints increases. The impact on the time is higher when using

probabilistic constraints: we obtain an exponential trend with 7 probabilistic constraints compared to 14 crisp ones.

6. Conclusions and Future Work

In this paper, we presented a new semantics for DECLARE process model specifications which allows one to specify probabilistic constraints representing their strength/importance in a specific process domain. In this way, we can model domains where some constraints are stronger (or weaker) than others. Such models are called Probabilistic Declarative Process Specifications. The underlying semantics is inspired by the distribution semantics from Probabilistic Logic Programming, and abstracts away from the formal Declare semantics (LTL_f or ALP), aiming at computing the probability of compliance of a trace versus the model. The computation of the compliance relies on an existing algorithm. Preliminary tests show that the time taken for computing the probability has an exponential trend by increasing the number of both crisp and probabilistic constraints. Future work includes: extending experimental evaluation, defining the compliance of a set of traces (log) vs. a PDS; extending the new semantics to manage uncertainty at the level of events, traces or the log itself; studying the profiles of energy consumption when computing the probability of compliance.

7. Acknowledgments

Research funded by the Italian Ministerial grant PRIN 2022 “Probabilistic Declarative Process Mining (PRODE)”, n. 20224C9HXA - CUP F53D23004240006, funded by European Union – Next Generation EU. Research partly funded by the Italian Ministry of University and Research through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI" - CUP J33C22002830006, funded by the European Union under the NextGeneration EU programme". This work was realized by Daniela Loreti with a research contract co-financed by the European Union (PON Ricerca e Innovazione 2014-2020 art. 24, comma 3, lett. a), della Legge 30/12/2010, n. 240 e s.m.i. e del D.M. 10/08/2021 n. 1062).

References

[1] M. Pesic, H. Schonenberg, W. M. van der Aalst, Declare: Full support for loosely-structured processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 2007, pp. 287–287. doi:10.1109/EDOC.2007.14.

[2] M. Pesic, Constraint-based workflow management systems : shifting control to users, Phd thesis 1 (research tu/e / graduation tu/e), Industrial Engineering and Innovation Sciences, 2008. doi:10.6100/IR638413, proefschrift.

[3] T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, in: K. Honda, A. Mycroft (Eds.), Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010, volume 69 of *EPTCS*, 2010, pp. 59–73. doi:10.4204/EPTCS.69.5.

[4] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Inf. Syst. 109 (2022) 102033. doi:10.1016/J.IS.2022.102033.

[5] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995, MIT Press, 1995, pp. 715–729.

[6] M. Montali, Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, volume 56 of *LNBIP*, Springer, 2010. doi:10.1007/978-3-642-14538-4.

[7] D. Azzolini, E. Bellodi, S. Ferilli, F. Riguzzi, R. Zese, Abduction with probabilistic logic programming under the distribution semantics, Int. J. Approx. Reason. 142 (2022) 41–63. doi:10.1016/J.IJAR.2021.11.003.

[8] van der Aalst, et al., Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I, volume 99 of *LNBIP*, Springer, 2011, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.

[9] M. Montali, Specification and Verification of Declarative Open Interaction Models - A Logic-based framework, Ph.D. thesis, alma, 2009. URL: http://amsdottorato.unibo.it/1829/.

[10] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 854–860. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

[11] C. D. Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: W. M. P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, volume 448 of *LNBIP*, Springer, 2022, pp. 108–152. doi:10.1007/978-3-031-08848-3_4.

[12] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, Verifiable agent interaction in abductive logic programming: The SCIFF framework, ACM Trans. Comput. Log. 9 (2008) 29:1–29:43. doi:10.1145/1380572.1380578.

[13] E. Dantsin, Probabilistic logic programs and their semantics, in: Russian Conference on Logic Programming, volume 592 of *LNCS*, Springer, 1991, pp. 152–164.

[14] D. Poole, Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities, New Generation Computing 11 (1993) 377–400.

[15] D. Poole, The Independent Choice Logic for modelling multiple agents under uncertainty, Artificial Intelligence 94 (1997) 7–56.

[16] N. Fuhr, Probabilistic datalog: Implementing logical information retrieval for advanced applications, Journal of the American Society for Information Science 51 (2000) 95–110.

[17] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: B. Demoen, V. Lifschitz (Eds.), 20th International Conference on Logic Programming (ICLP 2004), volume 3131 of *LNCS*, Springer, 2004, pp. 431–445. doi:10.1007/978-3-540-27775-0_30.

[18] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: A probabilistic Prolog and its application in link discovery, in: M. M. Veloso (Ed.), 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), volume 7, AAAI Press, 2007, pp. 2462–2467.

[19] J. Vennekens, M. Denecker, M. Bruynooghe, CP-logic: A language of causal probabilistic events and its relation to logic programming, Theory and Practice of Logic Programming 9 (2009) 245–308. doi:10.1017/S1471068409003767.

[20] F. Riguzzi, T. Swift, Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics, Theory and Practice of Logic Programming 13 (2013) 279–302. doi:10.1017/S1471068411000664.

[21] E. Bellodi, The distribution semantics in probabilistic logic programming and probabilistic description logics: a survey, Intelligenza Artificiale 17 (2023) 143 – 156. doi:10.3233/IA-221072.

[22] E. Bellodi, M. Gavanelli, R. Zese, E. Lamma, F. Riguzzi, Nonground abductive logic programming with probabilistic integrity constraints, Theory and Practice of Logic Programming 21 (2021) 557–574. doi:10.1017/S1471068421000417.

[23] T. H. Fung, R. A. Kowalski, The IFF proof procedure for abductive logic programming, Journal of Logic Programming 33 (1997) 151–165. doi:10.1016/S0743-1066(97)00026-5.

[24] U. O. Gustafsson, et al., Guidelines for perioperative care in elective colorectal surgery: Enhanced recovery after surgery (eras®) society recommendations: 2018, World Journal of Surgery 43 (2019) 659–695. doi:10.1007/s00268-018-4844-y.