ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Data-aware conformance checking with SMT

This is the submitted version (pre peer-review, preprint) of the following publication:

Published Version:
Felli P., Gianola A., Montali M., Rivkin A., Winkler S. (2023). Data-aware conformance checking with SMT. INFORMATION SYSTEMS, 117, 1-19 [10.1016/j.is.2023.102230].

Availability:
This version is available at: https://hdl.handle.net/11585/943276 since: 2024-07-02

Published:
DOI: http://doi.org/10.1016/j.is.2023.102230

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

07 September 2026

Multi-Perspective Conformance Checking of Uncertain Process Traces: an SMT-Based Approach

Paolo Felli[a], Alessandro Gianola[b], Marco Montali[b], Andrey Rivkin[b], Sarah Winkler[b]

[a] *University of Bologna, Bologna, 40126, BO, Italy*
[b] *Free University of Bozen-Bolzano, Piazza Domenicani 3, Bolzano, 39100, BZ, Italy*

Abstract

Conformance checking, one of the central tasks in process mining, compares the expected behaviour described by a reference process model to the actual behaviour recorded in an event log, with the goal of detecting deviations. Traditionally, it is assumed that the log provides a faithful and complete digital footprint of reality. However, assuming perfect logs is often unrealistic: real-life logs typically suffer from data quality issues, exposing uncertainty in their events, timestamps, and data attributes. We attack this problem by introducing a comprehensive framework for multi-perspective conformance checking dealing with uncertainty along three perspectives: control-flow, time, and data. From the modelling point of view, we consider process models formalised as Petri nets operating over data variables, and event logs presenting uncertainty at the event- and attribute-level. We cast conformance checking as an alignment problem, extending the traditional notions of alignment and cost function to deal with uncertainty along the three aforementioned perspectives. From the operational point of view, we show how (optimal) alignments can be computed through well-established automated reasoning techniques from Satisfiability Modulo Theories (SMT). Specifically, we show how previous results on data-aware SMT-based conformance checking can be lifted to this more sophisticated setting, obtaining a flexible framework that can seamlessly handle different variants of the problem. We formally prove correctness of our approach and implement it in the conformance checker `cocomot`. Finally, we perform a thorough experimental evaluation on synthetic and real-life logs, demonstrating the overall promising performance of our framework.

Keywords:

stochastic process mining, conformance checking, data quality, uncertainty, SMT

1. Introduction

Process mining is a family of approaches that combine data science and process management to support companies in evidence-based continuous improvement of their processes. Process mining techniques provide detailed insights into operational and administrative processes, as well as a variety of data-driven recommendations on how to optimise performance and reduce costs.

The core data used in process mining are event data that record the execution of several instances (also called cases) of the process of interest. Cases are, for example, patients in a healthcare process and orders in an order-to-delivery scenario. It is usually assumed that each event comes with the identifier of the case it belongs to, a timestamp, the indication of the activity in the process it refers to, plus additional attributes [1]. This allows one to reconstruct the full trace of events for each enacted process instance, forming an *event log* of the process.

One of the central tasks in process mining, which we consider in this article, is *conformance checking* [2]. Conformance compares the expected behaviour described by a reference process model and the actual behaviour recorded in an event log, with the goal of detecting deviations. There are multiple techniques of detecting deviations [2], but in this work we are interested in the one that describes such deviations in terms of *alignments* [3, 4, 5] – special structures relating log events to execution steps of the model, and providing a fine-grained feedback on the nature and extent of deviations.

Process mining tasks typically assume that the *event log* containing such data provides an accurate and complete representation of reality. However, this rather unrealistic assumption often falls short, due to two main reasons. First and foremost, the execution of process instances may inherently be traced with imprecision. Events may be missing or totally/partially wrongly recorded, due to various factors such as human errors, faulty loggers, and errors experienced during the event acquisition (e.g., through sensors). This regularly happens when the process contains parts that are under the responsibility of human actors, and that are executed in the real world without the direct support of an information system. Think, for example, at treatments in a hospital and activities executed at home by the patient in the context

of a healthcare process, or activities happening in the warehouse or during transportation in an order-to-delivery scenario. Such activities may not at all be logged, or logged a posteriori with uncertainty.

A second crucial factor is the fact that event logs are only rarely directly produced at runtime in a way that can be directly digested by process mining techniques. Often, instead, event logs are extracted from multiple, possibly heterogeneous information systems, through complex data integration and transformation pipelines, which may on their own introduce additional errors.

In the process mining literature, there are two lines of research used to mitigate this issue. The first line predominantly focuses on the data quality aspect and offers methodologies and techniques that can be used to handle various types of uncertainty during data preparation (see, e.g., [6]). The second line, instead, embraces the idea that event data are uncertain, and handles uncertainty directly inside the process mining task. This has led to two major families of "uncertainty-aware" process mining techniques. In the first family, usually known under the name of *stochastic process mining*, uncertainty is incorporated at the model level, while assuming that the log is accurate and complete [7, 8, 9, 10]. In the second family, instead, the event log itself is considered uncertain, and is consequently augmented with additional metadata indicating where uncertainty is located, and with which degree [11, 12].

In this work, we concentrate on this latter family of techniques, studying *conformance checking of uncertain process traces* where uncertainty can manifest itself at the event- and attribute-levels. Dealing with uncertainty in data attributes calls for considering data-aware process models. Our formalism of choice for reference process models is consequently that of data Petri net [13, 14] (DPN for short), which provides an interesting trade-off between expressiveness and tractability, and that naturally matches event logs with data attributes.

We attack this problem by introducing a comprehensive framework for *multi-perspective conformance checking*, which lifts data-aware conformance checking [15] to the relevant setting where event logs present uncertainty in the control-flow, time, and data. Our approach then belongs to the *single, deterministically known process* and *stochastically known log* configuration in the classification framework recently introduced in [16].

To deal with stochastically known logs, we build on the taxonomy of uncertainty types of event data proposed in [17, 11], and handle in particular the following four types.

- *Uncertain events* (called indeterminate events in [17]) are associated in a log trace with a *confidence value* that captures the degree of (un)certainty about whether the recorded event actually occurred during the process execution. This confidence value provides a measure for the *degree of trust* associated to the logging of the event, and should hence not be confused with a probability estimating the likelihood/frequency of the corresponding behaviour.
- *Uncertain timestamps* associate a recorded event with a time range when the event happened, instead of a single timestamp. This reflects that, due to coarse-grained and ambiguous logging, events in a trace are in general not totally ordered, and/or may be associated to multiple possible orderings. Consequently, a log trace must be considered as *a set* of occurred events, without pre-imposing a specific sequencing over those.
- *Uncertain activities* pertain events whose associated activity is unclear. This is realised by assigning to a recorded event a candidate set of possible activities (each with its own confidence value).
- *Uncertain data values*: event data attributes in data-aware processes often come with both ambiguity and coarseness, for example because sensors lack precision. Concretely, only a set of possible values or an interval may be associated with a data attribute of an event, so that the entire value range must be considered.

Using the notation in [11], this corresponds to logs of type $[\textsc{ead}]_W[\textsc{t}]_S$ or $[\textsc{eac}]_W[\textsc{t}]_S$, as data attributes can be either discrete or continuous.

In this work, we consider event logs where events can carry annotations concerning these four types of uncertainty, as well as a quantification of the respective lack of confidence. These annotations can be extracted from operational characteristics of the information system recording the event data, e.g. from information about the precision and reliability of the registration; or they can be directly attached to the generated events. For example, a log can be augmented with data about the precision or coarseness of logging devices in a sensor network. In other settings, uncertainty-related annotations may stem from domain knowledge on the precision and frequency of a specific human activity, or may be explicitly recorded by the logging infrastructure of the information system.

On top of these uncertain event data, we provide a twofold contribution:

1. From the modelling point of view, we formalise multi-perspective alignments under uncertainty, extending the traditional notions of alignment and cost function to deal with uncertainty along control-flow, time, and

data.

2. From the operational point of view, we show how (optimal) alignments can be computed through well-established automated reasoning techniques from Satisfiability (and Optimisation) Modulo Theories (SMT) [18].

More specifically, in the case of traces with uncertain data, alignments must take into consideration the possible values that uncertain events and attributes might take. To this end, we borrow and adapt to our setting the idea of *trace realization* proposed in [11], which resembles, in spirit, the well-known possible worlds semantics used in probabilistic databases [19]. Each realization of a log trace with uncertainty is an *ordered sequence* of events in which all the four types of uncertainty described above are resolved; however, due to uncertainty over timestamps and data attributes, infinitely many possible realizations may exist for a given uncertain trace.

Our task is then concretely substantiated as follows. Given a reference process model represented as data Petri net [13, 14] (DPN) and a log trace $\mathbf{ue}$ annotated with uncertainty, exhibit (*i*) a realization $\mathbf{e}$ of $\mathbf{ue}$, (*ii*) a model run $\mathbf{f}$ of the DPN, and (*iii*) an alignment between $\mathbf{e}$ and $\mathbf{f}$ such that this alignment is *optimal* with respect to all possible realizations and model runs, that is, the alignment has minimal cost among all choices for $\mathbf{e}$ and $\mathbf{f}$. In contrast to [11], the confidence values of the trace with uncertainty are employed as an essential component for measuring the cost when selecting realizations. In fact, all four uncertainty types described above are supported when constructing trace realizations, and the computation of best alignments is driven by a generic cost function whose components can be flexibly instantiated by domain experts so as to homogeneously account for a variety of domain-specific measures. To substantiate this technique, we formally prove that our approach is indeed capable of producing trace realizations and, given a concrete cost function instantiation, compute corresponding optimal alignments.

Instead of operationalizing our technique using ad-hoc algorithms, we build on previous work [20] that cast the problem of computing data-aware alignments for DPNs as an automated reasoning problem that can be solved by well-established SMT technologies dealing with Satisfiability and Optimisation Modulo Theories, such as Yices [21] and Z3 [22]. Specifically, we show how this previous SMT encoding can be lifted to the more sophisticated setting of this paper, compactly handling trace realizations in a symbolic way, and ultimately obtaining a flexible framework that can seamlessly han-

dle different variants of the problem. The encoding is implemented as an extension of CoCoMoT – the SMT-based conformance checking framework first presented in [20]. To the best of our knowledge, this is the first implementation of the conformance checking problem over logs with uncertain data using techniques coming from the domain of automated reasoning. To witness the practical feasibility of our approach, we subject this proof-of-concept implementation to a thorough experimental evaluation on publicly available and synthetically generated event logs and DPN models. As part of this evaluation, we compare our tool against the one proposed in [11], using the benchmarks reported therein and focusing on how both tools scale when different shares of event data are affected by uncertainty. In addition, we include experiments demonstrating (1) how various SMT solvers perform on solving the conformance checking problem over uncertain data, and (2) the advantage of our direct encoding of uncertain trace realizations, by comparing the runtime of our tool against that of original `cocomot` executed over pre-computed realizations.

This contribution is a revised and extended version of the conference paper [23], where we proposed a modeling framework and SMT encoding for data-aware conformance checking under uncertainty.

We complement the work in [23] by:

- providing a formal proof of correctness of our approach in dealing with uncertain traces and computing (optimal) alignments of their trace realizations. This includes showing the existence of an upper bound on the length of optimal alignments;
- illustrating in greater detail the implementation of the approach (based on the one in [23]), which extends `cocomot` to handle uncertainty;
- presenting a thorough experimental evaluation.

The paper is structured in the following way. The next section discusses approaches related to the one studied in this paper. In Sec. 3 we summarize preliminaries on DPNs and SMT. Sec. 4 describes the representation of log traces with uncertainty, as well as the notion of alignments. In Sec. 5 we examine components of the cost model that arise in the setting with uncertain event data. Specifically, we fix a concrete data-aware cost function that is used in the sequel (Sec. 5.2). Sec. 6 details the ingredients of our SMT-based encoding and provides theoretical results on the correctness of our approach. In Sec. 7 we outline the tool implementation, and in Sec. 8 we report on extensive experimental evaluation results. Finally, in Sec. 9 we conclude

with directions for future work.

2. Related Work

In the last decades, many advanced techniques for collecting and managing data have been proposed [24, 25]. In practice, it is often the case that the collected data are incomplete, contain errors, or simply vaguely specified. Notably, whereas for incomplete or faulty data it is hard to understand the level of imprecision, in the last case one usually talks about *uncertain data* for which the level of uncertainty can be quantified in some way [24].

To characterize problems with data quality in event logs, Bose *et al* [26] identify four categories of insufficiency: *missing*, *incorrect*, *imprecise*, and *irrelevant* data. In this work, we focus on imprecise data where a loss of precision occurs due to too coarse-grained logging, so that later process analysis tasks may lead to wrong results. For this reason, as discussed in the previous section, we assume that such imprecision is either quantified or that the set of alternatives is known, hence departing from existing techniques for handling missing data. For instance, in [27] the authors address the task of computing realizations from partially ordered event logs when timestamp information is absent, whereas here we support uncertain timestamps.

To address problems with *imprecise* data in logs, two lines of research recently emerged.

The first one is concerned with methodologies and techniques to improve the data quality in the data preparation phase, thus handling uncertainty in a preprocessing step [6, 28]. Both [6] and [28] emphasized the importance of data quality for responsible process mining, and focus on enhancing the data quality of event logs in the data preparation phase. Specifically, in [28] the authors proposed measures for data quality that retain the precision, accuracy, and reliability of activity labels and timestamps. These data quality characteristics are supposed to be assessed in the preprocessing phase. The authors then propose a conformance checking approach with a dedicated cost function that takes data quality into account. However, activity labels and timestamps of trace events are already considered fixed at this point, so that different realizations of the trace are not taken into account. While our SMT-based technique could be used for this conformance checking task (by implementing the designated cost function), it is in fact more general because we do not consider a single realization, but select the one that allows for a minimal-cost alignment.

The second line of research aims to incorporate the handling of uncertainty into the process mining tasks themselves, leading to novel process mining techniques where process models [7, 8, 9, 10] and/or event logs [29, 12, 11] explicitly reflect different kinds of uncertainty. In this work we are focusing on event logs with uncertainties. Perhaps, one of the most well known issues concerns the quality of timestamps: while traditional conformance checking approaches rely on traces being totally ordered, in reality time recordings are too coarse, or may deviate from the real times at which activities took place. To mitigate this problem, partially ordered traces are used [30]. In [29], the authors used this concept and design a conformance checking approach that produces partially ordered alignments. It works by first transforming the partially ordered trace to an event net, then computing a product net to achieve conformance checking. The approach is also applied to data annotated logs. We note that a partially ordered trace can be represented by an uncertain trace in our sense, where timestamp intervals are chosen appropriately. In [31], the authors propose a pre-processing technique that adds partial order to event logs so as to account for possible uncertainties that may arise due to coarse timestamps.

Our work is similar to the conformance checking approach over uncertain logs studied in [11]. There, the authors defined two types of uncertainty. *Strong* uncertainty considers attribute values in the log with unknown probability distribution values (for example, it is unknown whether an event was produced by executing activity a or b). Instead, *weak* uncertainty assumes that probabilities are known for such attribute values. However, the proposed conformance checking technique relies on transforming a weakly uncertain log into one containing only strong uncertainties. In our work, this is mitigated by providing an SMT encoding that accounts for both types of uncertainties.

In [32], the authors proposed an alignment-based conformance checking technique for logs with weakly uncertain data (that is, data for which probability values are known). Specifically, the authors considered only weakly uncertain activities and, instead of transforming logs into strongly uncertain ones as it is done in [11], proposed a notion of a stochastic trace model (represented as a stochastic Petri net), in which all probabilistic activity alternatives are taken into account at once. To compute alignments, the authors then defined the stochastic synchronous product between the reference process and trace models. Using such product, alignments can be computed using a shortest path algorithm on the product's reachability graph [2]. Although the reference process model considered in our approach is not stochas-

tic, weakly uncertain activities are managed by fully accounting for their respective probabilities which are then also reflected in the general cost model used for computing optimal alignments.

A similar distinction holds with respect to the work in [33]. There, a technique for computing probability estimations of individual realizations is presented, and although the main task is not conformance checking, it is discussed how conformance scores of uncertain traces can be computed by aggregating the scores of individual realizations, weighted by their individual probability estimate. Instead, our SMT-based approach does not need to compute realizations and their probabilities explicitly, as the encoding accounts for of them at once. Also, realizations are infinite in general (when data variables are present).

Recent works showed also the feasibility of SAT-based techniques to solve various process mining problems. For instance, the authors in [34] proposed an SMT encoding of the discovery task for classical Petri nets from event logs. In [35, 36] the authors propose to encode various conformance checking artifacts in SAT so that each artifact is computed using one single SAT instance. To obtain optimality of the result, the authors employ MaxSAT, using as objective function an encoding of the edit distance between a process run and a given log trace. Similar to this approach, we encoded the multi-perspective conformance checking problem in SMT in our previous work [20]. The approach presented in this work builds on top of the encoding from [20] and adds to it additional clauses allowing for the management of uncertain event data in a given trace.

3. Preliminaries

This section provides the required preliminaries on data Petri nets (DPNs) [37] and their execution semantics, as well as the main notions of Satisfiability Modulo Theories (SMT) [18].

3.1. Data Petri Nets

We rely on Data Petri nets (DPNs) to model multi-perspective processes, borrowing concepts and notations from [20, 15]. We consider the set of *(process variable) types* $\Sigma = \{\texttt{bool}, \texttt{int}, \texttt{rat}, \texttt{string}\}$ to capture the data types of variables that a process operates on. These types have the following associated domains: $\mathcal{D}(\texttt{bool}) = \mathbb{B}$, the booleans; $\mathcal{D}(\texttt{int}) = \mathbb{Z}$, the integers; $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$, the rational numbers; and $\mathcal{D}(\texttt{string}) = \mathbb{S}$, the strings over

some fixed alphabet. A set of *process variables* is denoted by V, and we assume that it is *typed* in the sense that there is a function $type\colon V \to \Sigma$ assigning a type to each variable in V. In order to refer to the variables that are read and written by activities in a process, we use two sets of annotated variables $V^r = \{v^r \mid v \in V\}$ (for read variables) and $V^w = \{v^w \mid v \in V\}$ (for written variables). We suppose that these sets are disjoint, and that $type(v^r) = type(v^w) = type(v)$ for every $v \in V$. Given V and a type $\sigma \in \Sigma$, V_σ denotes the subset of annotated variables in $V^r \cup V^w$ of type σ. To manipulate typed variables, we consider the following expressions:

Definition 3.1. *For a set of variables V,* constraints *c and expressions s, n, and r of types* `string`*,* `int`*, and* `rat` *are defined by the following grammar:*

$$c ::= v_b \mid b \mid n \geq n \mid r \geq r \mid r > r \mid s = s \mid c \wedge c \mid \neg c \qquad s ::= v_s \mid t$$
$$n ::= v_z \mid z \mid n + n \mid -n \qquad r ::= v_r \mid q \mid r + r \mid -r$$

where $v_b \in V_{\texttt{bool}}$, $b \in \mathbb{B}$, $v_s \in V_{\texttt{string}}$, $t \in \mathbb{S}$, $v_z \in V_{\texttt{int}}$, $z \in \mathbb{Z}$, $v_r \in V_{\texttt{rat}}$, and $q \in \mathbb{Q}$.

The set of constraints that can be constructed from a set of variables V is denoted by $\mathcal{C}(V)$, and the set of variables that appear in a constraint c by $\mathcal{V}ar(c)$, hence $\mathcal{V}ar(c) \subseteq V^w \cup V^r$. Standard equivalences apply, so disjunction (i.e., $\vee$) of constraints can be used, as well as comparisons $\neq$, $<$, $\leq$ on integer and rational expressions, though expressions of types `bool` and `string` only support (in)equality comparisons. Constraints as defined in Def. 3.1 serve to express on the one hand requirements on the values of variables that are read (V^r), and on the other hand conditions on values that are assigned (i.e., written) to variables (V^w) when executing an activity. Thus, given a constraint c as above, we refer to the annotated variables in V^r and V^w that appear in c as the *read* and *written variables*, respectively. Constraints attached to activities are called *guards*. For instance, a guard $(x^r > y^r)$ of an activity a demands that the value of variable x is greater than the value of y whenever a is executed. Similarly, a guard $(x^w > y^r + 1) \wedge (x^w < z^r)$ of an activity a' states that the new value assigned to x when executing a' is strictly greater than the current value of y plus 1, and strictly smaller than z.

Definition 3.2 (DPN)**.** *A* Petri net with data *(DPN) is given by a tuple $\mathcal{N} = (P, T, F, \ell, A, V, guard)$ such that:*

- (P, T, F, ℓ) *is a Petri net where* P *is a non-empty set of places,* T *is a non-empty disjoint set of transitions,* $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$ *is a flow relation, and* $\ell\colon T \to A \cup \{\tau\}$ *is a labeling function, where* A *is a finite set of activity labels and* τ *is a special symbol for silent transitions;*
- V *is a typed set of process variables; and*
- $guard\colon T \to \mathcal{C}(V)$ *assigns executability constraints.*

Following conventional notation, for $x \in P \cup T$, the *preset* of x is denoted by ${}^\bullet x := \{y \mid F(y, x) > 0\}$, the *postset* of x is denoted by and $x^\bullet := \{y \mid F(x, y) > 0\}$. The sets of variables read and written by a transition t are denoted by $read(t) = \{v \mid v^r \in \mathcal{V}ar(guard(t))\}$ and $write(t) = \{v \mid v^w \in \mathcal{V}ar(guard(t))\}$, respectively. We write $G_{\mathcal{N}}$ to refer to the set of transition guards occurring in $\mathcal{N}$.

Hereinafter, we adopt the important assumption that *silent transitions never modify process variables*. Formally, for every $t \in T$ s.t. $\ell(t) = \tau$, it holds that $write(t) = \emptyset$. This assumption is very natural as it prohibits process models to perform any adversarial, unobservable changes on the process data that can also not be aligned with events in logs.

In the sequel, V is assumed to be a fixed set of process variables. As customary, we use assignments to associate variables with values. First, we define *state variable assignments* to specify the current values of process variables: a state variable assignment is a function α such that $\text{DOM}(\alpha) = V$ and $\alpha(v) \in \mathcal{D}(type(v))$ for all $v \in V$; where DOM denotes the domain of a function. Correspondingly, we use *transition variable assignments* to assign values to annotated variables, and thus specify how variables change as the result of activity executions (cf. Def. 3.3): a transition variable assignment is a partial function β such that $\text{DOM}(\beta) \subseteq V^r \cup V^w$ and $\beta(x) \in \mathcal{D}(type(x))$ for all $x \in \text{DOM}(\beta)$. For a transition variable assignment β and a constraint c such that $\mathcal{V}ar(c) \subseteq \text{DOM}(\beta)$, β *satisfies* c, denoted $\beta \models c$, if the expression obtained from c by substituting variables with values according to β is valid in the respective domain theory.

A *state* of a DPN $\mathcal{N}$ is a tuple (M, α) where $M\colon P \to \mathbb{N}$ is a marking of the underlying Petri net (P, T, F, ℓ), and α is a state variable assignment. Thus, a state of a DPN describes at the same time the control flow progress and the current values of the process variables V. To simplify notation, we denote by $[p_1^{i_1}, \ldots, p_n^{i_n}]$ a multiset representing a marking in which each place p_k contains i_k tokens.

We next define when a state allows to fire a transition of the DPN.

Definition 3.3 (Transition firing)**.** *A transition $t \in T$ is* enabled *in a DPN state (M, α) with respect to a transition variable assignment β if:*

- $\mathcal{V}ar(guard(t)) \subseteq \text{DOM}(\beta)$*, i.e., β is defined for all variables in the guard;*
- $\beta(v^r) = \alpha(v)$ *for all $v \in read(t)$, i.e., β corresponds to α for read variables;*
- $\beta \models guard(t)$*, i.e., β satisfies the guard; and*
- $M(p) \geq F(p,t)$ *for all $p \in {}^\bullet t$.*

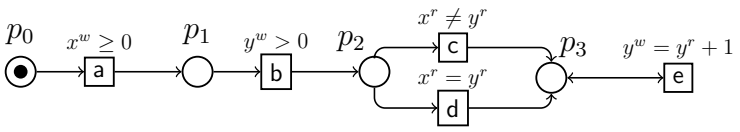
An enabled transition can fire *and create a new state (M', α'), that satisfies $M'(p) = M(p) - F(p,t) + F(t,p)$ for all $p \in P$, $\alpha'(v) = \beta(v^w)$ for all variables $v \in write(t)$, and $\alpha'(v) = \alpha(v)$ for every $v \notin write(t)$. If t is enabled with respect to β, the pair (t, β) is called a (valid)* transition firing*, and its execution is denoted by $(M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$.*

We assume that every DPN $\mathcal{N}$ has a fixed *initial* state (M_I, α_0), where M_I is the initial marking of the underlying Petri net and α_0 is a designated initial assignment, used to determine the initial values of the process variables V. The final marking of $\mathcal{N}$ is denoted M_F, and any state of the form (M_F, α_F) is called *final*, for α_F an arbitrary assignment.

A state (M, α) is called *reachable* in a given DPN iff there exists a sequence of valid transition firings $\mathbf{f} = \langle f_1, \ldots, f_n \rangle = \langle (t_1, \beta_1), \ldots, (t_n, \beta_n) \rangle$ such that $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} (M_1, \alpha_1) \xrightarrow{(t_2,\beta_2)} \ldots \xrightarrow{(t_n,\beta_n)} (M_n, \alpha_n) = (M, \alpha)$. For short, this is denoted by $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M, \alpha)$. Furthermore, $\mathbf{f}$ is termed a (valid) *process run* of $\mathcal{N}$ if $(M_I, \alpha_0) \xrightarrow{\mathbf{f}} (M_F, \alpha_F)$ for some α_F, so that the sequence of transition firings $\mathbf{f}$ leads from the initial to some final state. As conventional in conformance checking [13], we restrict ourselves to DPNs that are *relaxed data sound*, so that some final state is reachable.

In the sequel, given a DPN $\mathcal{N}$, the set of valid transition firings is denoted by $\mathcal{F}(\mathcal{N})$, and the set of process runs by $Runs(\mathcal{N})$.

Example 3.1. *Let $\mathcal{N}$ be the following DPN, with initial marking $[p_0^1]$, final marking $[p_3^1]$, and where the initial assignment is fixed as $\alpha_0 = \{x, y \mapsto 0\}$:*



Due to infinitely many possible assignments, the set $Runs(\mathcal{N})$ is infinite, but it contains for instance the following runs:

$$\langle(\mathsf{a}, \{x^w \mapsto 2\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\mathsf{c}, \{x^r \mapsto 2, y^r \mapsto 1\})\rangle$$
$$\langle(\mathsf{a}, \{x^w \mapsto 1\}), (\mathsf{b}, \{y^w \mapsto 1\}), (\mathsf{d}, \{y^r \mapsto 1, x^r \mapsto 1\})\rangle$$

3.2. Satisfiability and Optimization Modulo Theories (SMT)

In this section we summarize the main technical machinery exploited by the approach presented in this paper, namely SMT solving. For the sake of a formal description, we assume common syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, truth, satisfiability, and validity) concepts of first-order logic.

Following a convention in the SMT literature [38, 18], a theory $\mathcal{T}$ is a pair (Σ, Z), of a signature Σ and a class of Σ-structures Z, where the structures in Z represent the models of the theory $\mathcal{T}$. Given $\mathcal{T} = (\Sigma, Z)$, a Σ-formula ϕ is *$\mathcal{T}$-satisfiable* if Z contains a Σ-structure $\mathcal{M}$ such that ϕ is true in $\mathcal{M}$ under some assignment ν that assigns values from the carrier of $\mathcal{M}$ to the free variables of ϕ. This is denoted as $(\mathcal{M}, \nu) \models \phi$. The problem of (quantifier-free) *satisfiability modulo the theory* $\mathcal{T}$ ($SMT(\mathcal{T})$) asks to decide the $\mathcal{T}$-satisfiability of quantifier-free Σ-formulae.

Intuitively, the Satisfiability Modulo Theories (SMT) problem [18] is a generalization of the problem of propositional satisfiability (SAT). The SAT problem requires to decide whether a given propositional formula φ is satisfiable, and if this is the case, to find a satisfying assignment ν under which φ evaluates to true. For example, the formula $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg q)$ is satisfied by the assignment that sets $\nu(q) = \bot$ and $\nu(p) = \nu(r) = \top$. The SMT problem generalizes SAT by extending the language of propositional formulas with constants and operators from one or more first-order theories $\mathcal{T}$, and asking to decide satisfiability of a respective formula φ. In the last two decades, a plethora of powerful *SMT solvers* [22, 21, 39] have been developed that can solve the SMT problem for many common theories: they combine SAT solvers with designated decision procedures for the theories involved. SMT solvers have been proven useful for computer-aided verification, to prove correctness of software programs against some property of interest, and for program synthesis. Examples of well-established, decidable SMT theories are the theory of uninterpreted functions $\mathcal{EUF}$, the theory of bitvectors $\mathcal{BV}$ and the theory of arrays $\mathcal{AX}$, as well as the theories of linear arithmetic over the integers ($\mathcal{LIA}$) and the rationals ($\mathcal{LQA}$). For

this paper, only the latter two are relevant. For instance, the SMT formula $\varphi := a > 1 \wedge b \geq a \wedge (a + b = 10 \vee a - b = 20) \wedge p$, where a, b are integer and p is a propositional variable, is satisfiable by the assignment ν such that $\nu(a) = \nu(b) = 5$ and $\nu(p) = \top$.

An important generalization of the SMT problem that is relevant to this paper is the problem of Optimization Modulo Theories (OMT) [40]. The OMT problem requires to determine a satisfying assignment for a given formula φ that moreover minimizes or maximizes a given objective function. For instance, the satisfying assignment ν for the formula φ above is also a solution to the OMT problem that minimizes the objective b. Finally, we mention SMT-LIB [38] as an international initiative that defines common language standards and interfaces for SMT solvers, and provides an extensive on-line library of benchmarks.

4. Event Logs with Uncertainty and Alignments

We start by introducing the notion of an *event with uncertainty* that is used to represent an event recorded in the log, to which a certain kind and degree of uncertainty is explicitly associated. This uncertainty can be an annotation expressing the confidence that an event actually occurred, as well as uncertainty about an event's timestamp, its reference activity, or its data values. We assume that the present uncertainty is explicit in the sense that a fixed set of possible alternatives is known, possibly with their associated confidence values. Note that, in the case of data variables, the set of considered possibilities may be infinite.

We assume a finite set of event identifiers ID, a finite set of activity labels A, and a totally ordered set of possible timestamps TS. In the sequel, we assume that TS is $\mathbb{N}$ for the sake of simplicity.

Definition 4.1. *An event with uncertainty is given by a quintuple* $ue = \langle \text{ID}, conf, \text{LA}, \text{TS}, \alpha \rangle$ *such that the following are satisfied:*

- $\text{ID} \in ID$ *specifies an event identifier;*
- $conf$ *accounts for the confidence that the event actually happened and satisfies* $0 < conf \leq 1$. *The event is called* uncertain *if* $conf < 1$*;*
- $\text{LA} = \{b_1 : p_1, \ldots, b_n : p_n\}$ *is a set of activity labels* $b_i \in A$ *for all* $1 \leq i \leq n$*, each with a respective confidence value* $0 < p_i \leq 1$ *s.t.* $\sum_{i=1}^{n} p_i = 1$*;*
- TS *is an interval over* TS *or a finite subset of* TS*;*

- α *is a (possibly partial) function mapping variables* $v \in V$ *to a finite set of values in* $\mathcal{D}(type(v))$*, or an interval over this domain if* $type(v)$ *is* `int` or `rat`.

Though this use of α constitutes an abuse of notation, we adopt it for the sake of simplicity. The components of an event $ue = \langle \text{ID}, conf, \text{LA}, \text{TS}, \alpha \rangle$ are denoted by $\text{ID}(ue)$, $conf(ue)$, $\text{LA}(ue)$, $\text{TS}(ue)$ and $\alpha(ue)$, respectively. In line with [11], we do not associate confidence values with timestamps.

Definition 4.2. *A* log trace with uncertainty $\boldsymbol{ue}$ *is given by a finite set of events with uncertainty with unique event identifiers.*

Note that by defining a trace as a set, the order among its events need not be fixed (though it is constrained by the possible range of timestamps of events). A multiset of log traces with uncertainty L is called an *event log*.

For the sake of simplicity, we assume that event logs do not contain *lifecycle information*, i.e., the events recorded in the log do not keep track of the information about their start and end states, or other intermediate states of the executed activities. However, it is not difficult to extend our framework so as to incorporate also lifecycle information, although the setting becomes more complex to represent: we will comment on this point when we will present our SMT-based encoding of the event logs.

Example 4.1. *The following three sets are examples for traces with uncertainty, that we will later on relate to the DPN* $\mathcal{N}$ *from Ex. 3.1.*

$$\begin{aligned}
\boldsymbol{ue}_1 =& \{\langle \#_1, .25, \{\mathsf{a}:1\}, [0\text{-}5], \{x \mapsto \{2,3\}\}\rangle,\ \langle \#_2, .9, \{\mathsf{b}:.8, \mathsf{c}:.2\}, \{2\}, \{y \mapsto \{1\}\}\rangle \\
\boldsymbol{ue}_2 =& \{\langle \#_3, 1, \{\mathsf{a}:1\}, \{0\}, \{x \mapsto [1, 6.5]\}\rangle,\ \langle \#_4, 1, \{\mathsf{b}:1\}, \{2\}, \{y \mapsto \{1\}\}\rangle, \\
& \langle \#_5, 1, \{\mathsf{c}:1\}, \{3\}, \emptyset\rangle\} \\
\boldsymbol{ue}_3 =& \{\langle \#_6, 1, \{\mathsf{a}:1\}, \{2\}, \{x \mapsto \{6\}\}\rangle,\ \langle \#_7, 1, \{\mathsf{b}:1\}, \{2\}, \{y \mapsto \{1\}\rangle\}\}
\end{aligned}$$

For example, $\boldsymbol{ue}_1$ *consists of two events with uncertainty, having ids* $\#_1$ *and* $\#_2$*. The first one is uncertain as it happened with confidence* 0.25*; its associated activity* `a` *is fixed (as it has confidence* 1*); it occurred in the time interval* $[0, 5]$*; and the associated variable assignment assigns* x *to* 2 *or* 3*. Also the second event* $\#_2$ *is uncertain as it has confidence 0.9; its activity label may be* `b` *or* `c` *(with associated confidences* 0.8 *and* 0.2*, respectively); its timestamp is* 2*; and the associated variable assignment sets* y *to 1. The possibility of a variable being assigned a value from an interval is illustrated by* $\#_3$ *in* $\boldsymbol{ue}_2$*, where* x *can take any value in* $[1, 6.5]$*.*

An activity label $b \in A$ is said to be *admissible* for an event with uncertainty ue if there is some $p > 0$ such that $(b, p) \in \text{LA}(ue)$. In a similar way, timestamp and variable values are called admissible if they are possible according to the respective components in ue.

Next, we define *realizations* of a given log trace with uncertainty **ue**. Intuitively, a realization of **ue** is a sequence $\mathbf{e} = \langle e_1, \ldots, e_n \rangle$ of events without uncertainty that represents a possible sequentialization of *a subset of* **ue**, such that every event in **e** is an admissible concretization of an event with uncertainty in **ue**. Note that in this way, there may be events with uncertainty in **ue** that have no correspondent in **e**, and thus get discarded. Accounting for the likelihood of realizations, that is, the cost associated with preferring one realization over another when judging which realization fits better a trace with uncertainty, is discussed and formalized in Section 5.

An event without uncertainty, also called *event* for short, is a triple $(\text{ID}, b, \hat{\alpha})$ such that $\text{ID} \in ID$ is an event identifier, $b \in A$ is an activity label, and the variable assignment $\hat{\alpha}$ assigns to each variable $v \in V$ a fixed value of appropriate type. The components of an event $e = (\text{ID}, b, \hat{\alpha})$ are denoted by $\text{ID}(e)$, $lab(e)$ and $\hat{\alpha}(e)$, respectively. Events without uncertainty correspond to the usual notion of events in the conformance checking literature, augmented with variable assignments as in [20] to reflect the data dimension as well as identifiers. The latter will be used to relate them uniquely to an event with uncertainty, as explained below. We will write $\mathcal{E}$ to refer to the set of all events without uncertainty.

Definition 4.3 (Realization)**.** *A sequence* $\mathbf{e} = \langle e_1, \ldots, e_n \rangle$ *of events without uncertainty is a* realization *of a log trace with uncertainty* $\boldsymbol{ue}$ *if there exists a subset* $\{ue_1, \ldots, ue_n\} \subseteq \boldsymbol{ue}$ *and a timestamp sequence* $t_1 \leq t_2 \leq \cdots \leq t_n$ *such that for each* $1 \leq i \leq n$*:*

(i) t_i *is admissible for* ue_i*;*
(ii) $\text{ID}(e_i) = \text{ID}(ue_i)$*, so that every event in* **e** *corresponds to an event with uncertainty in* $\boldsymbol{ue}$ *via their identifiers;*
(iii) $lab(e_i) = b$ *for some activity label* b *admissible for* ue_i*;*
(iv) $\text{DOM}(\hat{\alpha}(e_i)) = \text{DOM}(\alpha(ue_i))$ *and* $\hat{\alpha}(e_i)(v) \in \alpha(ue_i)(v)$ *for all* v *such that* $\alpha(ue_i)(v)$ *is defined, i.e., for all* $v \in \text{DOM}(\alpha(ue_i))$*.*

In addition, for every $ue \in \boldsymbol{ue}$ *with* $conf(ue) = 1$ *there must be an event* $e \in \mathbf{e}$ *such that* $\text{ID}(e_i) = \text{ID}(ue_i)$*, i.e., a realization can only discard events in the log that are uncertain.*

Thus, a realization of a trace with uncertainty **ue** is a possible selection of

events in **ue** where a single activity label and a single value for each variable are selected from the corresponding event with uncertainty $ue \in \mathbf{ue}$, and this subset of events is ordered in an admissible way. The notation $\mathbf{e} \in \mathcal{R}(\mathbf{ue})$ expresses that $\mathbf{e}$ is a realization of **ue**. Confidence values are dropped from the events in a realization $\mathbf{e}$ since these values can be obtained from **ue**, using event identifiers to trace back matching events.

Note that the set of realizations $\mathcal{R}(\mathbf{ue})$ is never empty: one can always select $\{t_1, \ldots, t_n\}$ as in Def. 4.3 since even if two events have the same unique timestamp, both orderings are allowed in realizations. However, $\mathcal{R}(\mathbf{ue})$ may be infinite if the set of possible values of a variable specified in **ue** is an interval over a dense domain.

Example 4.2. *Let* $\boldsymbol{ue}_1$ *be as in Ex. 4.1. This trace with uncertainty has 13 realizations:* $\#_1$ *allows two variable assignments,* $\#_2$ *two possible admissible labels, the events may be ordered in two ways and since both events are uncertain, each one can be discarded. The following two are example realizations of* $\boldsymbol{ue}_1$ *with distinct event order, activity labels, and variable assignments:*

$$\mathbf{e}' = \langle\langle \#_1, \mathsf{a}, \{x \mapsto 2\}\rangle, \langle \#_2, \mathsf{b}, \{y \mapsto 1\}\rangle\rangle$$
$$\mathbf{e}'' = \langle\langle \#_2, \mathsf{c}, \{y \mapsto 1\}\rangle, \langle \#_1, \mathsf{a}, \{x \mapsto 3\}\rangle\rangle$$

We aim to develop a conformance checking procedure that constructs an *alignment* between a log trace $\mathbf{e}$ that is a realization of a log trace with uncertainty **ue**, and a process run of the DPN $\mathcal{N}$, by pairing event labels in the trace with activity labels of the model $\mathcal{N}$. Typically, when constructing an alignment, it is not possible to establish a one-to-one correspondence between events in the log trace and DPN transitions. Hence, we employ a dedicated "skip" symbol $\gg$ and consider the extended set of events $\mathcal{E}^{\gg} = \mathcal{E} \cup \{\gg\}$, as well as the extended set of transition firings $\mathcal{F}^{\gg} = \mathcal{F}(\mathcal{N}) \cup \{\gg\}$ for a DPN $\mathcal{N}$.

Given a DPN $\mathcal{N}$ and a set $\mathcal{E}$ of events (without uncertainty) as above, a *move* is a pair $(e, f) \in \mathcal{E}^{\gg} \times \mathcal{F}^{\gg} \setminus \{(\gg, \gg)\}$ which we call: (i) *log move* if $e \in \mathcal{E}$ and $f = \gg$; (ii) *model move* if $e = \gg$ and $f \in \mathcal{F}(\mathcal{N})$; (iii) *synchronous move* if $(e, f) \in \mathcal{E} \times \mathcal{F}(\mathcal{N})$. We denote by $Moves_{\mathcal{N}}$ the set of all moves.

A sequence of moves $\gamma = \langle (e_1, f_1), \ldots, (e_n, f_n)\rangle$, induces on the one hand a *log projection* $\gamma|_L$, which is the maximal subsequence $\langle e'_1, \ldots, e'_i\rangle$ of $\langle e_1, \ldots, e_n\rangle$ that is in $\mathcal{E}^*$, i.e., the subsequence of γ obtained by discarding all $\gg$ elements; and on the other hand a *model projection* $\gamma|_M$ which is the maximal subsequence $\langle f'_1, \ldots, f'_j\rangle$ of $\langle f_1, \ldots, f_n\rangle$ for which $\langle f'_1, \ldots, f'_j\rangle \in \mathcal{F}(\mathcal{N})^*$.

Definition 4.4 (Alignment)**.** *A sequence of moves γ is a* complete alignment *of a realization* $\mathbf{e}$ *with respect to a DPN* $\mathcal{N}$ *if* $\gamma|_L = \mathbf{e}$ *and* $\gamma|_M \in Runs(\mathcal{N})$*.*

Example 4.3. *Let the realization* $\mathbf{e}' = \langle\langle \#_1, \mathsf{a}, \{x \mapsto 2\}\rangle, \langle \#_2, \mathsf{b}, \{y \mapsto 1\}\rangle\rangle$ *be as in Ex. 4.2. We show three possible complete alignments of* $\mathbf{e}'$ *with respect to the DPN from Ex. 3.1:*

$\gamma^1_{\mathbf{e}'}$

$\#_1$		$\#_2$		$\gg$
a	$x^w \mapsto 2$	b	$y^w \mapsto 1$	c

$\gamma^2_{\mathbf{e}'}$

$\#_1$		$\#_2$		$\gg$
a	$x^w \mapsto 5$	b	$y^w \mapsto 1$	c

$\gamma^3_{\mathbf{e}'}$

$\#_1$		$\gg$		$\#_2$
a	$x^w \mapsto 2$	b	$y^w \mapsto 2$	d

In the remainder of this paper we only consider alignments that are complete, and we use the notation $Align(\mathcal{N}, \mathbf{e})$ to denote the set of all complete alignments for a log trace $\mathbf{e}$ with respect to $\mathcal{N}$.

As Ex. 4.3 demonstrates, alignments differ in how closely the process run matches the log trace: synchronous moves can have mismatching variable assignments (cf. the first move of $\gamma^2_{\mathbf{e}'}$) and different activity labels (cf. the third move of $\gamma^3_{\mathbf{e}'}$). This comparison of different alignments is captured by a cost function as discussed next.

5. Cost Functions and Optimal Alignments

This section addresses the problem of comparing different possible alignments of possible realizations of the same trace with uncertainty, by extending to event logs with uncertainty the established approach in the literature based on the notion of cost of alignments. We do so by introducing the cost of an alignment $\gamma_{\mathbf{e}}$ of a realization $\mathbf{e}$ of a trace with uncertainty $\mathbf{ue}$, which we denote by $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$.

5.1. General structure of the cost model

In this section, we present our cost model. We do not limit ourselves to one specific cost function; instead, we fix only the general *structure* for $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$, as shown in Fig. 1, leaving some of its elements arbitrary. Nevertheless, we illustrate and justify the cost components of said structure and focus on one of its possible instantiations, which we use in the encoding in Sec. 6 and in the implementation illustrated in Sec. 7. We call this instantiation *likelihood cost* (as it relies first and foremost on confidence values recorded in the log) and we denote it by $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$.

We first give the intuition on the structure of $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$. Importantly, we do not simply aim to find an optimal alignment for an arbitrary realization

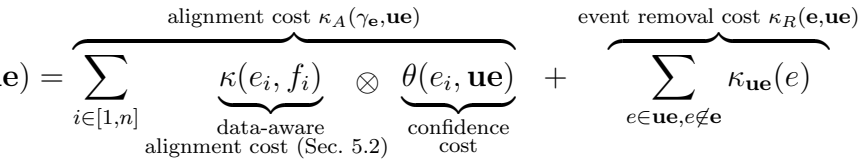


Figure 1: Cost structure for an alignment $\gamma_{\mathbf{e}} = \langle (e_1, f_1), \ldots, (e_n, f_n) \rangle$ selecting a realization $\mathbf{e}$ of a trace with uncertainty $\mathbf{ue}$. Intuitively, the cost associated to the selection of $\mathbf{e}$ is given by $\kappa_R(\mathbf{e}, \mathbf{ue})$ in addition to, at each step, the confidence cost $\theta(e_i, \mathbf{ue})$.

of a log trace with uncertainty $\mathbf{ue}$ as it was done in [11]. Instead, we want to take the confidence values associated with different realizations into account, by imposing a cost on the selection of a realization *in addition* to the cost of aligning it, as illustrated by the cost structure in Fig. 1. Consequently, the cost $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$ of an alignment $\gamma_{\mathbf{e}}$ with respect to an uncertain trace $\mathbf{ue}$ is obtained as the sum of two costs:

1) The alignment cost $\kappa_A(\gamma_{\mathbf{e}}, \mathbf{ue})$ rates the quality of the alignment $\gamma_{\mathbf{e}}$ for a chosen realization $\mathbf{e}$ of $\mathbf{ue}$. We follow a common approach in conformance checking, and assume that the alignment cost is based on a function κ: $Moves_{\mathcal{N}} \to \mathbb{R}^+$ that specifies a cost for each move $(e_i, f_i) \in \gamma_{\mathbf{e}}$. In Sec. 5.2 we will discuss in more detail how this function κ can be defined.

In addition, for a synchronous move or log move with event e_i, this cost is combined with a confidence penalty that depends on $conf(e_i)$ and on the confidence value p associated to the chosen activity label $b = lab(e_i)$, as recorded in the event with uncertainty ue so that $\text{ID}(e_i) = \text{ID}(ue)$, i.e., $(b, p) \in \text{LA}(ue)$. Intuitively, this imposes a penalty depending on the confidence of e_i and for selecting b as the activity chosen for e_i in the realization $\mathbf{e}$ of $\mathbf{ue}$.

We do not fix a specific calculation of this penalty but rather keep it parametric and denote it as $\theta(e_i, \mathbf{ue})$. The alignment cost of $\gamma_{\mathbf{e}}$ can then be defined as follows:

$$\kappa_A(\gamma_{\mathbf{e}}, \mathbf{ue}) = \textstyle\sum_{i=1}^{n} \kappa(e_i, f_i) \otimes \theta(e_i, \mathbf{ue}) \tag{1}$$

where $\otimes$ denotes an arbitrary operator to combine the two costs.

For our proposed cost function $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ shown in Fig. 2 and used in Sec. 6 we assume, given a realization $\mathbf{e}$ of a trace with uncertainty $\mathbf{ue}$ and

an alignment $\gamma_{\mathbf{e}} = \langle (e_1, f_1), \ldots, (e_n, f_n) \rangle$:

$$\kappa(e_i, f_i) \otimes \theta(e_i, \mathbf{ue}) = \begin{cases} \kappa(e_i, f_i) & \text{if } e_i = \gg, \text{ otherwise:} \\ \theta(e_i, \mathbf{ue}) & \text{if } \kappa(e_i, f_i) = 0 \\ \kappa(e_i, f_i) \cdot (1 + \theta(e_i, \mathbf{ue})) & \text{if } \kappa(e_i, f_i) > 0 \end{cases} \tag{2}$$

in which we fix $\theta(e_i, \mathbf{ue}) = (1 - \mathit{conf}(e_i)) + (1 - p)$, where b is the label of e_i, i.e., $b = \mathit{lab}(e_i)$, and p is the confidence value associated to b, i.e., $(b, p) \in \textsc{la}(ue)$ for $\textsc{id}(e_i) = \textsc{id}(ue)$.

Intuitively, in this instantiation of $\kappa_A(\gamma_{\mathbf{e}}, \mathbf{ue})$ used in $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ (see Fig. 2), the cost of model moves is simply (a data-aware extension of) the usual alignment cost, which we illustrate in Sec. 5.2. For other moves, the cost must include a penalty for having included e_i in the realization $\mathbf{e}$ of $\mathbf{ue}$ and for having selected $b = \mathit{lab}(e_i)$ as its label. Such penalty must decrease the more we are confident about event e_i and about b among its admissible activity labels, namely it must be inversely proportional to $\mathit{conf}(e_i)$ and p. In our choice of $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$, when the data-aware alignment cost is zero, we set the penalty to $\theta(e_i, \mathbf{ue}) = (1 - \mathit{conf}(e_i)) + (1 - p)$. When instead the data-aware alignment cost is greater than zero, then the penalty is computed by multiplying that cost by $(1 + \theta(e_i, \mathbf{ue}))$. Other definitions of θ and $\otimes$ are however possible.

2) The event removal cost $\kappa_R(\mathbf{e}, \mathbf{ue})$ measures the cost of selecting the subsets of the events in $\mathbf{ue}$ that appear in $\mathbf{e}$, discarding the remaining (uncertain) events. Although we do not fix a specific function κ_R, a reasonable option is to assume it to be based on a mapping $\kappa_{\mathbf{ue}} \colon \mathcal{E} \to \mathbb{R}_{\geq 0}$ assigning a removal cost to each event, proportionally to the confidence value $\mathit{conf}(ue)$ for $ue \in \mathbf{ue}$ so that $\textsc{id}(e) = \textsc{id}(ue)$. The cost for event removal is thus calculated as follows:

$$\kappa_R(\mathbf{e}, \mathbf{ue}) = \textstyle\sum_{e \in \mathbf{ue}, e \notin \mathbf{e}} \ \kappa_{\mathbf{ue}}(e)$$

For instance, for our proposed cost function $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ used in Sec. 6 and in our implementation we take $\kappa_{\mathbf{ue}}(e)$ to be precisely $\mathit{conf}(ue)$, for ue as above, when such a confidence value is smaller than 1, and equal to infinity otherwise (to prevent events that are not uncertain to be discarded from realizations). Namely, in $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ we define:

$$\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue}) = \sum_{i\in[1,n]} \overbrace{\kappa(e_i, f_i) \otimes \theta(e_i, \mathbf{ue})}^{\text{as in eq. (2)}} + \sum_{e\in\mathbf{ue}, e\notin\mathbf{e}} \overbrace{\kappa_{\mathbf{ue}}(e)}^{\text{as in eq. (3)}}$$

Figure 2: The instantion of the cost structure in Fig. 1, used in the encoding in Sec. 6.

$$\kappa_{\mathbf{ue}}(e) = \begin{cases} conf(ue) & \text{if } conf(ue) < 1, \text{ with } \textsc{id}(e) = \textsc{id}(ue) \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

However, different definitions of κ_R are conceivable as well.

In conclusion, according to these expressions, the cost of selecting $\mathbf{e}$ as a realization of $\mathbf{ue}$ results from $\kappa_R(\mathbf{e}, \mathbf{ue})$ for removed events combined, at each step, with a penalty $\theta(e_i, \mathbf{ue})$ for not having discarded e_i but having selected one admissible label among those associated to the uncertain event in $\mathbf{ue}$ with the same ID.

Accordingly, the cost function $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ that we propose as instantiation of the general cost function $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$ for alignments, and which we use for the encoding and implementation, is summarized in Fig. 2.

Example 5.1. *Consider again the trace with uncertainty* $\boldsymbol{u}\mathbf{e}_1$ *from Ex. 4.2:* $\boldsymbol{u}\mathbf{e}_1 = \{\langle \#_1, .25, \{\mathsf{a}:1\}, [0\text{-}5], \{x \mapsto \{2,3\}\}\rangle, \langle \#_2, .9, \{\mathsf{b}:.8, \mathsf{c}:.2\}, \{2\}, \{y \mapsto \{1\}\}\rangle\}$ *and three of its possible realizations:*

$$\mathbf{e}_1 = \langle\langle \#_1, \mathsf{a}, \{x \mapsto 3\}\rangle\rangle \quad \mathbf{e}_2 = \langle\langle \#_2, \mathsf{b}, \{y \mapsto 1\}\rangle\rangle \quad \mathbf{e}_3 = \langle\langle \#_2, \mathsf{c}, \{y \mapsto 1\}\rangle\rangle$$

where in all cases one of the two events was removed. By adopting $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \boldsymbol{u}\mathbf{e})$ *in Fig. 2 as instantiation of the general cost function, we have that* $\kappa_R(\mathbf{e}_2, \boldsymbol{u}\mathbf{e}_1) > \kappa_R(\mathbf{e}_1, \boldsymbol{u}\mathbf{e}_1)$ *since* $conf(\#_2) > conf(\#_1)$*. Similarly,* $\mathbf{e}_2$ *and* $\mathbf{e}_3$ *differ only in the activity chosen for* $\#_2$*, therefore the cost of selecting* $\mathbf{e}_2$ *is smaller than the cost of* $\mathbf{e}_3$ *since the confidence associated to activity* b *is greater than the one associated to* c*; hence* $\theta(\langle \#_2, \mathsf{b}, \{y \mapsto 1\}\rangle, \boldsymbol{u}\mathbf{e}_1) < \theta(\langle \#_2, \mathsf{c}, \{y \mapsto 1\}\rangle, \boldsymbol{u}\mathbf{e}_1)$.

The following definition formalizes the notion of cost of alignments illustrated so far in this section.

Definition 5.1 (Cost of alignments)**.** *Fixed the two arbitrary cost functions* κ_A *and* κ_R *introduced above, given* $\mathcal{N}$*, a trace with uncertainty* $\boldsymbol{u}\mathbf{e}$*, a realization* $\mathbf{e} = \langle e_1, \ldots, e_n\rangle$ *of* $\boldsymbol{u}\mathbf{e}$ *and an alignment* $\gamma_{\mathbf{e}} = \langle (e_1, f_1), \ldots, (e_n, f_n)\rangle \in$

Align$(\mathcal{N}, \mathbf{e})$*, the* cost *of alignment* $\gamma_{\mathbf{e}}$ *w.r.t.* $\boldsymbol{u}\mathbf{e}$*, denoted* $\mathfrak{K}(\gamma_{\mathbf{e}}, \boldsymbol{u}\mathbf{e})$*, is computed as shown in Fig. 1:*

$$\mathfrak{K}(\gamma_{\mathbf{e}}, \boldsymbol{u}\mathbf{e}) = \kappa_A(\gamma_{\mathbf{e}}, \boldsymbol{u}\mathbf{e}) + \kappa_R(\mathbf{e}, \boldsymbol{u}\mathbf{e}).$$

We say that an alignment $\gamma_{\mathbf{e}}$ is *optimal for* a given trace (specifically, a realization) $\mathbf{e}$ if $\kappa_A(\gamma_{\mathbf{e}}, \mathbf{ue})$ is minimal among all complete alignments for $\mathbf{e}$, namely there is no alignment $\gamma'_{\mathbf{e}} \in \mathit{Align}(\mathcal{N}, \mathbf{e})$ with $\kappa_A(\gamma'_{\mathbf{e}}, \mathbf{ue}) < \kappa_A(\gamma_{\mathbf{e}}, \mathbf{ue})$. Similarly, given $\mathcal{N}$ and a trace with uncertainty $\mathbf{ue}$, we say that $\gamma_{\mathbf{e}}$ is *optimal for* $\mathbf{ue}$ if $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$ is minimal among all possible realizations of $\mathbf{ue}$, i.e., there is no other realization $\mathbf{e}' \in \mathcal{R}(\mathbf{ue})$ and alignment $\gamma_{\mathbf{e}'} \in \mathit{Align}(\mathcal{N}, \mathbf{e}')$ that has a smaller cost, i.e., such that $\mathfrak{K}(\gamma_{\mathbf{e}'}, \mathbf{ue}) < \mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$.

Definition 5.2 (Conformance checking)**.** *Given a DPN* $\mathcal{N}$*, the* conformance checking task *for a trace with uncertainty* $\boldsymbol{u}\mathbf{e}$ *is to find a realization* $\mathbf{e} \in \mathcal{R}(\boldsymbol{u}\mathbf{e})$ *and an alignment* $\gamma_{\mathbf{e}} \in \mathit{Align}(\mathcal{N}, \mathbf{e})$ *that is optimal for* $\boldsymbol{u}\mathbf{e}$*.*

It is possible that, for a given trace with uncertainty $\mathbf{ue}$, multiple realizations $\mathbf{e}$ and optimal alignments $\gamma_{\mathbf{e}}$ exist, though the minimal cost is unique once we fix our cost function. The *conformance checking task for a log with uncertainty* consists of the conformance checking task for all its traces.

As an example of further instantiation of the general cost structure in Fig. 1, we show how to capture the alignment problem in [11] (once adapted to our representation of uncertain event logs).

Example 5.2. *It is possible to capture the task of finding the lower-bound on the cost of possible alignments among the set of all realizations of a given trace with uncertainty* $\boldsymbol{u}\mathbf{e}$ *(as considered in [11]), by simply choosing* $\kappa_R(\mathbf{e}, \boldsymbol{u}\mathbf{e}) = 0$*,* $\theta(e, \boldsymbol{u}\mathbf{e}) = 1$ *and by taking* $\otimes$ *as product. Intuitively, this corresponds to paying no cost for the selection of arbitrary realizations (hence no cost for discarding events), thus simply returning one that has minimal alignment cost* κ*. Namely, given an alignment* $\gamma_{\mathbf{e}} = \langle (e_1, f_1), \ldots, (e_n, f_n) \rangle$ *for a realization* $\mathbf{e}$ *of* $\boldsymbol{u}\mathbf{e}$*, this corresponds to considering* $\mathfrak{K}^{br}(\gamma_{\mathbf{e}}, \boldsymbol{u}\mathbf{e}) = \sum_{i=1}^{n} \kappa(e_i, f_i)$*. In the rest of the paper, we call this instantiation of the cost structure the* best-realization cost*; it is also supported by our implementation.*

In the remainder of this section, we discuss separately the definition of the alignment cost κ, namely our data-aware extension of the usual distance-based cost function. This is used both in the generic cost function $\mathfrak{K}(\gamma_{\mathbf{e}}, \mathbf{ue})$ (see Fig. 1) and in our instantiation $\mathfrak{K}^{lh}(\gamma_{\mathbf{e}}, \mathbf{ue})$ (see Fig. 2).

5.2. Data-aware Alignment Cost Function

We use a generalized form of a cost function to measure the conformance between a trace (and specifically a realization) and a process run in $Runs(\mathcal{N})$, i.e., to define $\kappa\colon Moves_{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ as used in Def. 5.1. As in [20], we parameterize this by three penalty functions:

$$P_L\colon \mathcal{E} \to \mathbb{N} \qquad P_M\colon \mathcal{F}(\mathcal{N}) \to \mathbb{N} \qquad P_=\colon \mathcal{E} \times \mathcal{F}(\mathcal{N}) \to \mathbb{N}$$

called *log move penalty*, *model move penalty* and *synchronous move penalty*, respectively. Intuitively, $P_L(e)$ gives the cost that has to be paid for a log move e; $P_M(f)$ penalizes a model move f; and $P_=(e, f)$ expresses the cost to be paid for a synchronous move of e and f. Then, the data-aware cost function $\kappa\colon Moves_{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ we adopt in Def. 5.1 is simply defined as $\kappa(e, f) = P_L(e)$ if $f = \gg$, $\kappa(e, f) = P_M(f)$ if $e = \gg$, and $\kappa(e, f) = P_=(e, f)$ otherwise. By suitably instantiating $P_=$, P_L, and P_M, one can obtain conventional cost functions [20]: the Levenshtein distance [35, 36], standard cost function for multi-perspective conformance checking [13, 15].

Data-aware Cost Component of $P_=$. Crucially, as we are considering DPNs in place of standard Petri nets, we typically want to consider a data-aware extension of the usual distance-based cost function for synchronous moves. Therefore, when comparing an event $e = (\text{ID}, b, \hat{\alpha})$ belonging to a realization and a transition firing $f = (t, \beta)$, we want $P_=(e, f)$ to compare also the values that are assigned to variables by $\hat{\alpha}$ and β, respectively. In Ex. 4.3, as an example, in the alignment $\gamma^2_{\mathbf{e}_1}$ the first (synchronous) move presents a mismatch between the value assigned to variable x by the event $\#_1$ (i.e., $\hat{\alpha}(\#_1)(x) = 2$) and the value assigned by transition firing (i.e., $(\mathsf{a}, \{x^w \mapsto 5\})$). Various data-aware realizations of $P_=$ have been already addressed in the literature [13, 20].

Example 5.3. *Consider again the trace with uncertainty $\boldsymbol{u}\mathbf{e}_1$ from Ex. 5.1,*

$$\{\langle \#_1, .25, \{\mathsf{a}\colon 1\}, [0\text{-}5], \{x \mapsto \{2, 3\}\}\rangle, \langle \#_2, .9, \{\mathsf{b}\colon .8, \mathsf{c}\colon .2\}, \{2\}, \{y \mapsto \{1\}\}\rangle\}.$$

Assume to fix P_M, P_L to be as usual in the standard cost function, as illustrated in [20], namely $P_L(b, \alpha) = 1$; $P_M(t, \beta) = 0$ if t is silent (i.e., $\ell(t) = \tau$) and $P_M(t, \beta)$ equal to 1 plus the number of variables written by guard(t) otherwise. For $P_=$, assume a data-aware extension (of the $P_=$ used to match the standard cost function [20]) defined as: $P_=(\langle \text{ID}, b, \hat{\alpha}\rangle, (t, \beta)) = |\{v \mid \hat{\alpha}(v) \neq$

$\beta(v^w)\}| \,/\, |V|$ *if* b *is the label of* t*, i.e.* $b = \ell(t)$*, and* $P_{=}(\langle \textsc{id}, b, \hat{\alpha}\rangle, (t, \beta)) = \infty$ *otherwise. Then, if we instantiate cost functions as in Ex. 5.1 (also used in our encoding in Sec. 6), the optimal alignment of* $\boldsymbol{u}\mathbf{e}_1$ *w.r.t. the DPN* $\mathcal{N}$ *depicted in Ex. 3.1 is* $\gamma^1_{\mathbf{e}'}$ *as shown in Ex. 4.3 (of cost 2.05).*

Further, if we consider the task of finding the lower-bound on the cost of optimal alignments for any realization of $\boldsymbol{u}\mathbf{e}_1$ *(as discussed below Def. 5.2), then this is 1 and it is given as well by the realization* $\mathbf{e}'$ *and* $\gamma^1_{\mathbf{e}'}$*.*

6. Encoding

This section describes our encoding for conformance checking of traces with uncertainty. We give all details for the two cost functions proposed in Sec. 5, but it is not hard to adapt the encoding to a similar cost scheme. The conformance checking process via SMT can be structured in four phases:

(1) represent the process run, the trace realization, and the alignment symbolically by a set of SMT variables,
(2) set up constraints Φ that express optimality of the alignment,
(3) solve Φ to obtain a satisfying assignment ν, and
(4) decode from ν the process run, trace realization, and optimal alignment.

A similar procedure was followed in [20], though with crucial differences: In the uncertainty setting, step (1) requires to represent not only the process run but also the trace realization. To this end, all uncertainty features of the given uncertain trace must be suitably reflected. A particular complication is given by the fact that the order of the events need not be fixed. In addition, different cost functions are used, as explained in Sec. 5, which requires modifications in phase (2). All changes also affect the decoding in (4).

6.1. Upper Bounding the Alignment Length

We start by showing that there is a computable upper bound on the length of the process run associated with an optimal alignment. This is a crucial observation, since, as was already done in earlier SAT-based approaches [36, 20, 41], we need to construct a symbolic representation of both a process run and an alignment, that are subsequently concretized using an SMT solver. This symbolic representation must use a finite number of variables, so that we need to fix upfront an upper bound on the size of the process run. In general, the upper bound, and even its existence, depends on the chosen cost function. The next lemma establishes a (coarse) upper bound for the likelihood cost function from Fig. 2.

Lemma 6.1 (Upper bound for $\mathfrak{K}^{lh}$)**.** *Let $\mathcal{N}$ be a DPN and $\boldsymbol{ue}$ a trace with uncertainty that has m_1 certain and m_2 uncertain events. Let $\langle f_1, \ldots, f_n \rangle$ be a run of $\mathcal{N}$ such that $c = \sum_{j=1}^{n} P_M(f_j)$ is minimal, and k the length of the longest acyclic sequence of silent transitions in $\mathcal{N}$. Then there is an optimal alignment γ for $\boldsymbol{ue}$ such that the length of $\gamma|_M$ is at most $(4m_1 + 2m_2 + c) \cdot k$.*

Proof. Let $\mathbf{e} = \langle e_1, \ldots, e_m \rangle \in \mathcal{R}(\mathbf{ue})$ be the realization associated with γ, so $m \leq m_1 + m_2$. Then $\gamma_0 = \langle (e_1, \gg), \ldots, (e_m, \gg), (\gg, f_1), \ldots (\gg, f_n) \rangle$ is a valid alignment for $\mathbf{e}$, with the following cost: First, the log steps $\gamma_0' = \langle (e_1, \gg), \ldots, (e_m, \gg) \rangle$ have cost $\kappa_A(\gamma_0', \mathbf{ue}) + \kappa_R(\gamma_0', \mathbf{ue})$. Since for each uncertain event e in $\mathbf{ue}$ the event removal cost is $\kappa_{\mathbf{ue}}(e) < 1$, we have $\kappa_R(\gamma_0', \mathbf{ue}) \leq m_2$. Second, $\kappa_A(\gamma_0', \mathbf{ue}) = \sum_{i=1}^{m_1} \big(\kappa(e_i, \gg) \cdot (1 + \theta(e_i, \mathbf{ue}))\big)$, where $\kappa(e_i, \gg) = 1$ and $\theta(e_i, \mathbf{ue}) \leq 2$. Hence, $\kappa_A(\gamma_0', \mathbf{ue}) \leq 3m_1$. Overall, $\mathfrak{K}^{lh}(\gamma_0, \mathbf{ue}) \leq 3m_1 + m_2 + c$, where c is the cost of $\langle (\gg, f_1), \ldots (\gg, f_n) \rangle$ by assumption.

An optimal alignment γ must satisfy $\mathfrak{K}^{lh}(\gamma, \mathbf{ue}) \leq \mathfrak{K}^{lh}(\gamma_0, \mathbf{ue})$. By assumption, γ can have at most m synchronous moves. For a conservative estimate, we assume their cost is 0. In addition, $\gamma|_M$ may feature non-silent moves, each costing at least 1, and thus have at most $3m_1 + m_2 + c$ non-silent moves (otherwise, we would have $\mathfrak{K}^{lh}(\gamma, \mathbf{ue}) > \mathfrak{K}^{lh}(\gamma_0, \mathbf{ue})$). Thus γ has at most $4m_1 + 2m_2 + c$ moves that are either synchronous or non-silent model moves. However, in between every one of these, as well as before and afterwards, there may be silent transitions of cost 0. There could also be loops which consist of silent transitions only, and executing such a loop an arbitrary number of times does not incur any additional cost. However, as silent transitions do not write variables, an alignment whose process run involves such a loop cannot have strictly smaller cost than the alignment obtained by omitting the loop. So it is safe to assume that in the optimal alignment γ, in between two non-silent transitions there are at most k silent ones. Thus, the length of $\gamma|_M$ is at most $(4m_1 + 2m_2 + c) \cdot k$. □

Note that if the DPN has loops that consist of silent transitions only, there can be infinitely many optimal alignments that are not bounded in length (as such loops can be repeated arbitrarily often without increasing the cost). Indeed, the above lemma shows *existence* of an optimal alignment within that bound, but the bound need not apply to *all* optimal alignments.

For the best-realization cost function $\mathfrak{K}^{br}$ from [11] (cf. Ex. 5.2), it is also possible to upper-bound the length of the process run in an optimal

alignment: one can simply pick an arbitrary realization of the given trace with uncertainty, and use the results from [20].

6.2. Encoding the Process Run

We assume that $\mathcal{N}$ is the given DPN as in Def. 3.2, and n is an upper bound on the length of the process run $\gamma|_M$ in the optimal alignment γ (cf. Sec. 6.1). To ensure that an optimal alignment of length *exactly* n exists, we first modify $\mathcal{N}$ such that it has a silent transition from the final marking to itself. Then, the following SMT variables are used to represent this run:

(a) transition variables $\mathtt{S}_i$ for $1 \leq i \leq n$ of type integer. The intended meaning is that, if the set of transitions in $\mathcal{N}$ is $T = \{t_1, \ldots, t_{|T|}\}$, $\mathtt{S}_i$ is assigned j iff the i-th transition in the process run is t_j;

(b) marking variables $\mathtt{M}_{i,p}$ of type integer for all places $p \in P$ and all i with $0 \leq i \leq n$. The intended meaning is that $\mathtt{M}_{i,p}$ is assigned k iff there are k tokens in place p at instant i;

(c) data variables $\mathtt{X}_{i,v}$ for all $v \in V$ and all i, $0 \leq i \leq n$, where the type of these variables is that of v. The intended meaning is that $\mathtt{X}_{i,v}$ is assigned r iff the value of v at instant i is r. We also write $\mathtt{X}_i$ for $(\mathtt{X}_{i,v_1}, \ldots, \mathtt{X}_{i,v_k})$.

To encode that variables (a)–(c) are assigned values which represent a valid process run, we use the constraints

$$\varphi_{run} = \varphi_{init} \wedge \varphi_{fin} \wedge \varphi_{trans} \wedge \varphi_{enabled} \wedge \varphi_{mark} \wedge \varphi_{data}$$

where the subformulas reflect the requirements to a process run as follows:

$$\textstyle\bigwedge_{p \in P} \mathtt{M}_{0,p} = M_I(p) \wedge \bigwedge_{v \in V} \mathtt{X}_{0,v} = \alpha_0(v) \qquad (\varphi_{init})$$

$$\textstyle\bigwedge_{p \in P} \mathtt{M}_{n,p} = M_F(p) \qquad (\varphi_{fin})$$

$$\textstyle\bigwedge_{1 \leq i \leq n} 1 \leq \mathtt{S}_i \leq |T| \qquad (\varphi_{trans})$$

$$\textstyle\bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq |T|} (\mathtt{S}_i = j) \rightarrow \bigwedge_{p \in {}^\bullet t_j} \mathtt{M}_{i-1,p} \geq |{}^\bullet t_j|_p \qquad (\varphi_{enabled})$$

$$\bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq |T|} (\mathtt{S}_i = j) \rightarrow \bigwedge_{p \in P} \mathtt{M}_{i,p} - \mathtt{M}_{i-1,p} = |t_j{}^\bullet|_p - |{}^\bullet t_j|_p \qquad (\varphi_{mark})$$

$$\bigwedge_{1 \leq i < n} \bigwedge_{1 \leq j \leq |T|} (\mathtt{S}_i = j) \rightarrow \rho(guard(t_j)) \wedge \bigwedge_{v \notin write(t_j)} \mathtt{X}_{i-1,v} = \mathtt{X}_{i,v} \qquad (\varphi_{data})$$

Here (φ_{init}) enforces the initial marking M_I, (φ_{fin}) enforces the last marking to be final, (φ_{trans}) ensures that transitions variables are assigned numbers

that correspond to transitions in $\mathcal{N}$, ($\varphi_{enabled}$) makes sure that fired transitions were enabled, where $|{}^\bullet t_j|_p$ denotes the multiplicity of p in the multiset ${}^\bullet t_j$, (φ_{mark}) ascertains that the moving of tokens follows the flow relation of the Petri net, where $|t_j{}^\bullet|_p$ denotes the multiplicity of p in $t_j{}^\bullet$, and finally (φ_{data}) transition guards must be satisfied. Here ρ is the substitution with domain $V_r \cup V_w$ defined as $\rho(v^r) = \mathtt{X}_{i-1,v}$ and $\rho(v^w) = \mathtt{X}_{i,v}$.

Example 6.1. *We illustrate the encoding on the DPN from Ex. 3.1, and* $\boldsymbol{ue}_1$ *be as in Ex. 4.1. For simplicity, we assume that the maximal length of the process run in the optimal alignment is* $n = 3$*. We add an additional silent transition from place 3 to itself, called* f*, and encode transitions* $\mathsf{a}, \mathsf{b}, \ldots, \mathsf{f}$ *by integers 1 to 6. This results in the following constraints:*

$$\mathtt{M}_{0,0}=1 \wedge \mathtt{M}_{0,1}=\mathtt{M}_{0,2}=\mathtt{M}_{0,3}=0 \wedge \mathtt{X}_{0,x}{=}\mathtt{X}_{0,y}{=}0 \qquad (\varphi_{init})$$

$$\mathtt{M}_{3,0}=\mathtt{M}_{3,1}=\mathtt{M}_{3,2}=0 \wedge \mathtt{M}_{3,3}=1 \qquad (\varphi_{fin})$$

$$\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{a} \vee \mathtt{S}_i=\mathsf{b} \vee \mathtt{S}_i=\mathsf{c} \vee \mathtt{S}_i=\mathsf{d} \vee \mathtt{S}_i=\mathsf{e} \vee \mathtt{S}_i=\mathsf{f}) \qquad (\varphi_{trans})$$

$$\begin{aligned}
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{a} \to \mathtt{M}_{i-1,0}\ge 1) \wedge (\mathtt{S}_i=\mathsf{b} \to \mathtt{M}_{i-1,1}\ge 1) \wedge (\mathtt{S}_i=\mathsf{c} \to \mathtt{M}_{i-1,2}\ge 1) \wedge \\
&\qquad (\mathtt{S}_i=\mathsf{d} \to \mathtt{M}_{i-1,2}\ge 1) \wedge (\mathtt{S}_i=\mathsf{e} \to \mathtt{M}_{i-1,3}\ge 1) \wedge (\mathtt{S}_i=\mathsf{f} \to \mathtt{M}_{i-1,3}\ge 1) \qquad (\varphi_{enabled})\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{a} \to (\mathtt{M}_{i-1,0}{-}\mathtt{M}_{i,0}=1 \wedge \mathtt{M}_{i,1}{-}\mathtt{M}_{i-1,1}=1 \wedge \mathtt{M}_{i,2}=\mathtt{M}_{i-1,2} \wedge \mathtt{M}_{i,3}=\mathtt{M}_{i-1,3}))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{b} \to (\mathtt{M}_{i,0}=\mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i-1,1}{-}\mathtt{M}_{i,1}=1 \wedge \mathtt{M}_{i,2}{-}\mathtt{M}_{i-1,2}=1 \wedge \mathtt{M}_{1,3}=\mathtt{M}_{0,3}))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{c} \to (\mathtt{M}_{i,0}=\mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1}=\mathtt{M}_{i-1,1} \wedge \mathtt{M}_{i-1,2}{-}\mathtt{M}_{i,2}=1 \wedge \mathtt{M}_{i,3}{-}\mathtt{M}_{i-1,3}=1))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{d} \to (\mathtt{M}_{i,0}=\mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1}=\mathtt{M}_{i-1,1} \wedge \mathtt{M}_{i-1,2}{-}\mathtt{M}_{i,2}=1 \wedge \mathtt{M}_{i,3}{-}\mathtt{M}_{i-1,3}=1))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{e} \to (\mathtt{M}_{i,0}=\mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1}=\mathtt{M}_{i-1,1} \wedge \mathtt{M}_{i,2}=\mathtt{M}_{i-1,2} \wedge \mathtt{M}_{i,3}=\mathtt{M}_{i-1,3}))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{f} \to (\mathtt{M}_{i,0}=\mathtt{M}_{i-1,0} \wedge \mathtt{M}_{i,1}=\mathtt{M}_{i-1,1} \wedge \mathtt{M}_{i,2}=\mathtt{M}_{i-1,2} \wedge \mathtt{M}_{i,3}=\mathtt{M}_{i-1,3})) \qquad (\varphi_{mark})\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{a} \to (\mathtt{X}_{i,x}\ge 0 \wedge \mathtt{X}_{i,y}=\mathtt{X}_{i-1,y})\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{b} \to (\mathtt{X}_{i,y}>0 \wedge \mathtt{X}_{i,x}=\mathtt{X}_{i-1,x})\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{c} \to (\mathtt{X}_{i-1,x}\ne\mathtt{X}_{i-1,y} \wedge \mathtt{X}_{i,x}=\mathtt{X}_{i-1,x} \wedge \mathtt{X}_{i,y}=\mathtt{X}_{i-1,y}{+}1))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{d} \to (\mathtt{X}_{i-1,x}=\mathtt{X}_{i-1,y} \wedge \mathtt{X}_{i,x}=\mathtt{X}_{i-1,x} \wedge \mathtt{X}_{i,y}=\mathtt{X}_{i-1,y}{+}1))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{e} \to (\mathtt{X}_{i,y}=\mathtt{X}_{i-1,y}{+}1 \wedge \mathtt{X}_{i,x}=\mathtt{X}_{i-1,x}))\\
&\textstyle\bigwedge_{1\le i\le 3} (\mathtt{S}_i=\mathsf{f} \to (\mathtt{X}_{i,x}=\mathtt{X}_{i-1,x} \wedge \mathtt{X}_{i,y}=\mathtt{X}_{i-1,y})) \qquad (\varphi_{data})
\end{aligned}$$

6.3. Trace Realization Constraints

If a trace contains some kind of uncertainty, the realization taken for an optimal alignment must be encoded as well, which requires additional variables. We assume that $\mathbf{ue} = \{ue_1, \ldots, ue_m\}$ where $ue_j = \langle \text{ID}, conf, \text{LA}, \text{TS}, \alpha \rangle$

for each $1 \leq j \leq m$, with $\text{LA} = \{b_1 : p_1, \ldots, b_{N_j} : p_{N_j}\}$. Then, three kinds of variables are used to represent the realization, for all $1 \leq j \leq m$:

(d) a boolean *drop variable* $\texttt{drop}_{ue_j}$ that expresses whether the event is discarded and satisfies $\texttt{drop}_{ue_j} \rightarrow (ue_j.conf < 1)$, i.e., it can only be assigned true for uncertain events;

(e) an integer *activity variable* $\texttt{A}_{ue_j}$ that encodes which of the labels $b_1, \ldots, b_{N_j}$ is chosen in the realization, with the constraint that $\bigvee_{s=1}^{N_j} \texttt{A}_{ue_j} = b_s$, and

(f) *trace data variables* $\texttt{D}_{v,ue_j}$ of suitable type for all $v \in V$ that express which value a variable v takes in the realization of ue_j; these variables must satisfy $\bigvee_{c \in ue.\alpha} \texttt{D}_{v,ue_j} = c$ if $\alpha(ue)(v)$ is a set, or $l \leq \texttt{D}_{v,ue_j} \leq u$ if $\alpha(ue)(v)$ is an interval $[l, u]$.

We call **ue** *sequential* if every element of **ue** has a single, distinct timestamp. In this case, **ue** can be assumed to be ordered by the timestamps as $\langle ue_1, \ldots, ue_n \rangle$. Otherwise, **ue** has at least one uncertain timestamp, and we use the following additional variables for all j, $1 \leq j \leq m$:

(g) a *time stamp variable* $\texttt{T}_{ue_j}$ to express when event ue_j happened, with the constraint $\bigvee_{t \in \text{TS}(ue_j)} \texttt{T}_{ue_j} = t$ if $\text{TS}(ue_j)$ is a set, or $l \leq \texttt{T}_{ue_j} \leq u$ if $\text{TS}(ue_j)$ is an interval $[l, u]$,

(h) an integer *position variable* $\texttt{P}_{ue_j}$ for the position of ue_j in the realization,

(i) an integer *item variable* $\texttt{L}_k$ that indicates the k-th element in the realization, i.e., $\texttt{L}_k$ has value $\text{ID}(ue_j)$ if and only if the k-th event in the trace with uncertainty is ue_j; we thus issue the constraint $\bigvee_{j=1}^{m} \texttt{L}_k = \text{ID}(ue_j)$ to fix the range of $\texttt{L}_k$, for all $1 \leq k \leq m$.

The formula φ_{trace} consists of the range constraints in (d)-(i), in addition to:

$$\bigwedge_{j=1}^{m} \bigwedge_{k=1}^{m} (\texttt{P}_{ue_j} < \texttt{P}_{ue_k} \longrightarrow \texttt{T}_{ue_j} \leq \texttt{T}_{ue_k}) \wedge (\texttt{T}_{ue_j} < \texttt{T}_{ue_k} \longrightarrow \texttt{P}_{ue_j} < \texttt{P}_{ue_k})$$
$$\bigwedge_{j=1}^{m} \bigwedge_{k=1}^{m} \texttt{L}_j = \text{ID}(ue_k) \longleftrightarrow \texttt{P}_{ue_k} = j$$

In this way we require that, first, the positions assigned to uncertain events by $\texttt{P}_{ue_j}$ are compatible with the time stamps assigned by $\texttt{T}_{ue_j}$ and, second, that the $\texttt{P}_{ue_j}$ variables work as an "inverse function" of the $\texttt{L}_k$ variables.

Example 6.2. *We continue Ex. 6.1. Since event ue_1 in $\boldsymbol{ue}_1$ has an uncertain timestamp, constraints encoding the sequence of events must be included as well. Since both events have a confidence below 1, the constraints in (d) amount to $\texttt{drop}_j \rightarrow \top$, which amount to $\top$. To simplify notation, we write*

$\mathtt{A}_1$ *instead of* $\mathtt{A}_{ue_1}$ *etc, and obtain the following constraints for* φ_{trace}*:*

$$\mathtt{A}_1 = \mathsf{a} \wedge (\mathtt{A}_2 = \mathsf{b} \vee \mathtt{A}_2 = \mathsf{c}) \wedge (\mathtt{D}_{x,1} = 2 \vee \mathtt{D}_{x,1} = 3) \wedge \mathtt{D}_{y,2} = 1 \qquad \text{(e), (f)}$$

$$(0 \leq \mathtt{T}_1 \leq 5) \wedge \mathtt{T}_1 = 2 \wedge (\mathtt{L}_1 = 1 \vee \mathtt{L}_1 = 2) \wedge (\mathtt{L}_2 = 1 \vee \mathtt{L}_2 = 2) \qquad \text{(g), (i)}$$

$$(\mathtt{P}_1 < \mathtt{P}_2 \longrightarrow \mathtt{T}_1 \leq \mathtt{T}_2) \wedge (\mathtt{T}_1 < \mathtt{T}_2 \longrightarrow \mathtt{P}_1 < \mathtt{P}_2) \wedge$$

$$(\mathtt{P}_2 < \mathtt{P}_1 \longrightarrow \mathtt{T}_2 \leq \mathtt{T}_1) \wedge (\mathtt{T}_2 < \mathtt{T}_1 \longrightarrow \mathtt{P}_2 < \mathtt{P}_1) \wedge (\mathtt{L}_1 = 1 \longleftrightarrow \mathtt{P}_1 = 1) \wedge$$

$$(\mathtt{L}_1 = 2 \longleftrightarrow \mathtt{P}_2 = 1) \wedge (\mathtt{L}_2 = 1 \longleftrightarrow \mathtt{P}_1 = 2) \wedge (\mathtt{L}_2 = 2 \longleftrightarrow \mathtt{P}_2 = 2)$$

Notice that incorporating in the traces some information about the lifecycle of events would require to add suitable string variables, one for each event/activity, such that they store the current execution state of the referred event (e.g., if it is in one of its starting states, in some intermediate state, or at one of its end states). Moreover, one needs to conjoin other formulae, involving these new variables, that constrain the lifecycle behavior of activities: for example, it is required to impose that, when some event occurs, the initial states happen first and before the possible intermediate states, whereas the end states only after all the other states, and during the execution of the event no other activity can interfere. The encoding is conceptually similar, but certainly increases the size of the formula that represents realizations of traces.

6.4. Encoding the Cost Function

All encoding ingredients outlined so far are independent from the chosen cost function. To encode the alignment and its cost we use additional variables:

(j) distance variables $\mathtt{d}_{i,j}$ of type integer for $0 \leq i \leq m$ and $0 \leq j \leq n$; the intended meaning is that $\mathtt{d}_{i,j}$ represents the alignment cost of the prefix $\mathbf{e}|_i$ of the log trace realization $\mathbf{e}$ and prefix $\mathbf{f}|_j$ of the process run $\mathbf{f}$ (both of which are yet to be determined).

The search for an optimal alignment is based on a notion of *edit distance*, similar as in [20, 36]. More precisely, we assume that the data-aware alignment cost $\kappa(e_i, f_i)$ in Fig. 1 can be encoded using a *distance-based* cost function with *penalty functions* P_L, P_M, and $P_=$ as discussed in Sec. 5.2. The function $P_=$ is assumed to be data-aware, i.e., to depend only on (mis)matching variable assignments and activity labels between the events in realizations and transition firings in process runs. We assume that there are SMT encodings of these penalty functions, denoted as $[P_=]_{i,j}$, $[P_M]_j$, and $[P_L]_i$. Moreover, we assume that there are encodings of the event removal cost function $[\kappa_{\mathbf{ue}}]_i$

and the confidence cost function $[\theta_{\mathbf{ue}}]_i$, defined for the i-th element of the log trace realization, as well as an encoding $[\otimes]$ of the operator $\otimes$. We then consider the following constraints for $i, j > 0$: we set $\mathtt{d}_{0,0} = 0$, and

$$\mathtt{d}_{i,0} = \min([P_L]_i \; [\otimes] \; [\theta_{\mathbf{ue}}]_i, [\kappa_{\mathbf{ue}}]_i) + \mathtt{d}_{i-1,0} \qquad \mathtt{d}_{0,j} = [P_M]_j + \mathtt{d}_{0,j-1}$$

$$\mathtt{d}_{i,j} = \min \begin{cases} [P_=]_{i,j} \; [\otimes] \; [\theta_{\mathbf{ue}}]_i + \mathtt{d}_{i-1,j-1} \\ [P_L]_i \; [\otimes] \; [\theta_{\mathbf{ue}}]_i + \mathtt{d}_{i-1,j} \\ [\kappa_{\mathbf{ue}}]_i + \mathtt{d}_{i-1,j} \\ [P_M]_j \; [\otimes] \; [\theta_{\mathbf{ue}}]_i + \mathtt{d}_{i,j-1} \end{cases} \tag{φ_δ}$$

This encoding works for all cost function that follow the structure of Fig. 2. It constitutes an *operational way* to evaluate such cost functions, where the components $\kappa_{\mathbf{ue}}$ and θ are distributed to single moves, which allows us to use the encoding schema based on the edit distance. The inductive case $\mathtt{d}_{i,j}$ is computed so as to locally choose the move with minimal cost. Finally, $\mathtt{d}_{m,n}$ encodes the cost of the complete alignment, which will thus be used as the minimization objective.

For our likelihood cost function, according to the Eqs. 2 and 3, we use $u \; [\otimes] \; w = u$ for model moves, i.e., in the last line of (φ_δ), and $u \; [\otimes] \; w = \mathit{ite}(u = 0, w, u \cdot (1 + w))$ in all other places. Next, $[\kappa_{\mathbf{ue}}]_i$ can be defined as a (nested) case distinction on the element from **ue** that is chosen for the i-th position (represented with variable $\mathtt{L}_i$, cf. Sec. 6.3):

$$\begin{aligned} [\kappa_{\mathbf{ue}}]_i = & \mathit{ite}(L_i = \textsc{id}(ue_1) \wedge \mathtt{drop}_{ue_1}, \mathit{conf}(ue_1), \dots \\ & \mathit{ite}(L_i = \textsc{id}(ue_m) \wedge \mathtt{drop}_{ue_m}, \mathit{conf}(ue_m), \infty) \dots) \end{aligned} \tag{4}$$

The expression $[\theta_{\mathbf{ue}}]_i$ can be encoded via case distinctions in a similar way.

For the best-realization cost function from Ex. 5.2, we simply set $u \; [\otimes] \; w = u \cdot w$, $[\kappa_{\mathbf{ue}}]_i = 0$, and $[\theta_{\mathbf{ue}}]_i = 1$. Then the expressions can be simplified such that no actual multiplication is needed, and the encoding remains in the (decidable) realms of linear arithmetic. We illustrate this cost function on the running example:

Example 6.3. *We continue Ex. 6.2 and instantiate (φ_δ) for the best-realization cost function, using for $\kappa(e, f)$ the Levenshtein distance as in [11]. So we set $[P_L]_i = [P_M]_j = 1$ and $[\theta_{\mathbf{ue}}]_i = 1$ for all i, j, and*

$$\begin{aligned} [P_=]_{i,j} = \mathit{ite}(\mathtt{S}_j = \mathtt{a} \wedge \mathtt{L}_i = 1, 0, \mathit{ite}(\mathtt{S}_j = \mathtt{b} \wedge \mathtt{L}_i = 2 \wedge \mathtt{A}_2 = \mathtt{b}, 0, \\ \mathit{ite}(\mathtt{S}_j = \mathtt{c} \wedge \mathtt{L}_i = 2 \wedge \mathtt{A}_2 = \mathtt{c}, 0, \infty))) \end{aligned}$$

which gives cost 0 if the jth transition in the model run and the ith label in the realization match, and ∞ otherwise. Note that $[\kappa_{\mathbf{ue}}]_i = 0$ for all i since all events are uncertain. Thus, we get from (φ_δ) the equations $\mathtt{d}_{i,0} = 0$ for all $0 \leq i \leq 2$; and moreover $\mathtt{d}_{0,j} = 1 + \mathtt{d}_{0,j-1}$ for all $0 < j \leq 3$, and

$$\mathtt{d}_{1,1} = min([P_=]_{1,1} + \mathtt{d}_{0,0}, \mathtt{d}_{0,1}, 1{+}\mathtt{d}_{1,0}) \quad \mathtt{d}_{2,1} = min([P_=]_{2,1} + \mathtt{d}_{1,0}, \mathtt{d}_{1,1}, 1{+}\mathtt{d}_{2,0})$$
$$\mathtt{d}_{1,2} = min([P_=]_{1,2} + \mathtt{d}_{0,1}, \mathtt{d}_{0,2}, 1{+}\mathtt{d}_{1,1}) \quad \mathtt{d}_{2,2} = min([P_=]_{2,2} + \mathtt{d}_{1,1}, \mathtt{d}_{1,2}, 1{+}\mathtt{d}_{2,1})$$
$$\mathtt{d}_{1,3} = min([P_=]_{1,3} + \mathtt{d}_{0,2}, \mathtt{d}_{0,3}, 1{+}\mathtt{d}_{1,2}) \quad \mathtt{d}_{2,3} = min([P_=]_{2,3} + \mathtt{d}_{1,2}, \mathtt{d}_{1,3}, 1{+}\mathtt{d}_{2,2})$$

6.5. Solving and Decoding

Given the constraints above, we solve the following OMT problem:

$$\varphi_{run} \wedge \varphi_{trace} \wedge \varphi_\delta \quad \text{minimizing} \quad \mathtt{d}_{m,n} \tag{Φ}$$

Given a satisfying assignment ν for (Φ), we construct a process run $\mathbf{f}_\nu = \langle f_1, \ldots, f_n \rangle$ as follows: Assuming that the set of transitions T consists of $t_1, \ldots, t_{|T|}$ in the ordering already used for the encoding, we set $f_i = (t_{\nu(\mathtt{s}_i)}, \beta_i)$. Let α_j, $0 \leq j \leq n$, be the state variable assignments given by $\alpha_j(v) = \nu(\mathtt{X}_{j,v})$ for all $v \in V$, and $\beta_i(v^r) = \alpha_{i-1}(v)$ and $\beta_i(v^w) = \alpha_i(v)$ for all $v \in V$. Next, a realization $\mathbf{e}_\nu = \langle e_1, \ldots, e_k \rangle$ is obtained by ordering the events with uncertainty in $\mathbf{ue}$ according to $\nu(\mathtt{T}_{ue_i})$, dropping all ue_i such that $\mathtt{drop}_{ue_i}$ is true, and fixing the label and data values to $\nu(\mathtt{A}_{ue_i})$ and $\nu(\mathtt{D}_{ue_i})$, respectively. In order to obtain an alignment between $\mathbf{f}_\nu$ and $\mathbf{e}_\nu$, a family of partial alignments is defined, for all $i, j > 0$:

$$\gamma_{0,0} = \epsilon \qquad \gamma_{0,j} = \gamma_{0,j-1} \cdot (\gg, f_j)$$

$$\gamma_{i,0} = \begin{cases} \gamma_{i-1,0} \cdot (e_i, \gg) & \text{if } \nu(\delta_{i,0}) = \nu([P_L]_i \,[\otimes]\, [\theta_{\mathbf{ue}}]_i + \delta_{i-1,0}) \\ \gamma_{i-1,0} & \text{if } \nu(\delta_{i,0}) = \nu([\kappa_{\mathbf{ue}}]_i + \delta_{i-1,0}) \end{cases}$$

$$\gamma_{i,j} = \begin{cases} \gamma_{i-1,j} \cdot (e_i, \gg) & \text{if } \nu(\delta_{i,j}) = \nu([P_L]_i \,[\otimes]\, [\theta_{\mathbf{ue}}]_i + \delta_{i-1,j}) \\ \gamma_{i-1,j} & \text{if } \nu(\delta_{i,j}) = \nu([\kappa_{\mathbf{ue}}]_i + \delta_{i-1,j}) \\ \gamma_{i,j-1} \cdot (\gg, f_j) & \text{if } \nu(\delta_{i,j}) = \nu([P_M]_j \,[\otimes]\, [\theta_{\mathbf{ue}}]_i + \delta_{i,j-1}) \\ \gamma_{i-1,j-1} \cdot (e_i, f_j) & \text{if } \nu(\delta_{i,j}) = \nu([P_=]_{i,j} \,[\otimes]\, [\theta_{\mathbf{ue}}]_i + \delta_{i-1,j-1}). \end{cases}$$

In fact, the decoding need not be unique, as multiple alternatives in the above case distinction can apply. In this case, any applicable option will yield an optimal (though not unique) alignment.

Example 6.4. *For the constraints collected in Ex. 6.1–6.4, and minimization objective* $\mathtt{d}_{2,3}$*, we obtain an assignment* ν *with* $\nu(\mathtt{d}_{2,3}) = 1$*, so the optimal alignment has cost 1. For the variables representing the realization, we get* $\nu(\mathtt{A}_1) = \mathsf{a}$*,* $\nu(\mathtt{A}_2) = \mathsf{c}$*,* $\mathtt{D}_{x,1} = 2$*, and* $\mathtt{D}_{y,2} = 1$ *to fix the activity labels and data,* $\nu(\mathtt{T}_1) = 1$*,* $\nu(\mathtt{T}_2) = 2$ *for the timestamps, and* $\nu(\mathtt{P}_1) = \nu(\mathtt{L}_1) = 1$*,* $\nu(\mathtt{P}_2) = \nu(\mathtt{L}_2) = 2$ *to fix the event positions. This choice represents the realization* $\mathbf{e} = \langle\langle \#_1, \mathsf{a}, \{x \mapsto 2\}\rangle, \langle \#_2, \mathsf{c}, \{y \mapsto 1\}\rangle\rangle$*. For the model run,* $\nu(\mathtt{S}_1) = \mathsf{a}$*,* $\nu(\mathtt{S}_2) = \mathsf{b}$*, and* $\nu(\mathtt{S}_3) = \mathsf{c}$ *fix the transitions;* $\nu(\mathtt{M}_{0,0}) = 1$*,* $\nu(\mathtt{M}_{1,1}) = 1$*,* $\nu(\mathtt{M}_{2,2}) = 1$*, and* $\nu(\mathtt{M}_{3,3}) = 1$ *and all other* $\mathtt{M}_{i,j}$ *are assigned 0, which amounts to the markings where one token moving from* p_0 *to* p_3*; and* $\nu(\mathtt{X}_{i,x}) = 0$ *for all* i*,* $\nu(\mathtt{X}_{0,y}) = 0$*, and* $\nu(\mathtt{X}_{i,y}) = 1$ *for all* $i > 0$*, i.e.,* x *is set to 0 in the* a *transition and* y *is set to 1 by* b*. The distance variables are assigned as follows:*

$\nu(\mathtt{d}_{0,0}) = 0$	$\nu(\mathtt{d}_{0,1}) = 1$	$\nu(\mathtt{d}_{0,2}) = 2$	$\nu(\mathtt{d}_{0,3}) = 3$
$\nu(\mathtt{d}_{1,0}) = 1$	$\nu(\mathtt{d}_{1,1}) = 0$	$\nu(\mathtt{d}_{1,2}) = 1$	$\nu(\mathtt{d}_{1,3}) = 2$
$\nu(\mathtt{d}_{2,0}) = 2$	$\nu(\mathtt{d}_{2,1}) = 1$	$\nu(\mathtt{d}_{2,2}) = 2$	$\nu(\mathtt{d}_{2,3}) = 1$

$\#_1$		$\gg$	$\#_2$	
a	$x^w \mapsto 0$	b	c	$y^w \mapsto 1$

The arrows indicate how $\mathtt{d}_{2,3}$ *was computed, from which the alignment can be deduced: diagonal arrows represent synchronous moves, the horizontal arrow a model move. This amounts to the alignment on the right, which has indeed cost 1 according to the cost function from Ex. 5.2.*

6.6. Correctness

In this section we prove that the constructed sequence of moves is indeed an alignment that solves our conformance checking task, cf. Def. 5.2.

Lemma 6.2. *For any satisfying assignment* ν *to* (Φ)*,* (i) $\mathbf{f}_\nu$ *is a process run, and* (ii) $\mathbf{e}_\nu$ *is a realization of* $\boldsymbol{u}\mathbf{e}$*.*

Proof. (i) Let M_i be the marking such that $M_i(p) = \nu(\mathtt{M}_{i,p})$, for all $p \in P$, and α_i the state variable assignment such that $\alpha_i(v) = \nu(\mathtt{X}_{i,v})$, for all $v \in V$ and $0 \leq i \leq n$. For $\mathbf{f}_\nu = \langle f_1, \ldots, f_n\rangle$, we show by induction on i that the transition sequence $\mathbf{f}_{\nu,i} = \langle f_1, \ldots, f_i\rangle$ satisfies $(M_I, \alpha_0) \xrightarrow{\mathbf{f}_{\nu,i}} (M_i, \alpha_i)$ for all $0 \leq i \leq n$. In the base case $i = 0$, so $\mathbf{f}_{\nu,i}$ is empty. As ν satisfies $\varphi_{init,fin}$, it must be that $M_0 = M_I$ and α_0 is the initial assignment, so the claim trivially holds. In the inductive step, we consider $\mathbf{f}_{\nu,i+1} = \langle f_1, \ldots, f_{i+1}\rangle$ and assume that $\mathbf{f}_{\nu,i}$ satisfies $(M_I, \alpha_0) \xrightarrow{\mathbf{f}_{\nu,i}} (M_i, \alpha_i)$. For the last transition firing

$f_{i+1} = (t_j, \beta)$ there must be some j such that $1 \leq j \leq |T|$ and $\nu(\mathtt{S}_{i+1}) = j$, by construction and requirement (a) above. Since ν is a solution to (Φ), it satisfies $\varphi_{enabled}$ so that t_j is enabled in M_i. Moreover, as ν satisfies φ_{mark} and φ_{data}, we have $(M_i, \alpha_i) \xrightarrow{f_{i+1}} (M_{i+1}, \alpha_{i+1})$. This concludes the induction proof. For the case where $i = n$, we thus obtain $(M_I, \alpha_0) \xrightarrow{\mathbf{f}_{\nu,n}} (M_n, \alpha_n)$, and $\mathbf{f}_{\nu,n} = \mathbf{f}_\nu$. Finally, given that ν satisfies φ_{fin}, the last marking M_n must be final and hence $\mathbf{f}_\nu \in \mathit{Runs}(\mathcal{N})$.

(*ii*) Let $\{ue_1, \ldots, ue_k\}$ be all events $ue \in \mathbf{ue}$ such that $\nu(\mathtt{drop}_{ue}) = \bot$. By requirement (d), all events in $\mathbf{ue} \setminus \{ue_1, \ldots, ue_k\}$ are uncertain. By construction, $\mathbf{e}_\nu = \langle e_1, \ldots, e_k \rangle$ is obtained from $\{ue_1, \ldots, ue_k\}$ by taking for each ue_i the timestamp value $t_i = \nu(\mathtt{T}_{ue_i})$ in a way such that $t_1 \leq t_2 \leq \cdots \leq t_k$. For all $1 \leq i \leq k$, t_i is an admissible timestamp for ue_i by requirement (g). We have $lab(e_i) = \nu(\mathtt{A}_{ue_i})$, which is admissible by requirement (e), and $\hat{\alpha}(e_i)(v) = \nu(\mathtt{D}_{v,ue_i})$ for all $v \in V$, which is admissible by requirement (f). Thus $\mathbf{e}_\nu$ is a realization of $\mathbf{ue}$ according to Def. 4.3. □

This lemma shows that the decoding provides both a valid process run and a trace realization. Next we demonstrate that the decoded alignment is optimal. To this end, we assume for the sake of simplicity that the final marking is non-empty, and admits a silent transition to itself; however, this restriction could be avoided by encoding refinements. The following proof of correctness uses our likelihood cost function, but it can be adapted to different costs.

Theorem 6.1. *Let $\mathcal{N}$ be a DPN, $\boldsymbol{ue}$ a log trace with uncertainty and ν a solution to (Φ) as in Sec. 6.5. Then $\gamma_{m,n}$ is an optimal alignment for $\boldsymbol{ue}$ of cost $\mathfrak{K}^{lh}(\gamma_{m,n}, \boldsymbol{ue}) = \nu(\mathtt{d}_{m,n})$.*

Proof. By Lem. 6.2, $\mathbf{f}_\nu \in \mathit{Runs}(\mathcal{N})$ and $\mathbf{e}_\nu$ is a realization of $\mathbf{ue}$. Let $ue_1, \ldots, ue_m$ be the sequence of events in $\mathbf{ue}$ ordered in a way such that $\nu(\mathtt{T}_{ue_1}) \leq \cdots \leq \nu(\mathtt{T}_{ue_m})$. Moreover, let $\mathbf{ue}_i$ be the subset of $\mathbf{ue}$ such that $\mathbf{ue}_i = \{ue_1, \ldots, ue_i\}$ for all $0 \leq i \leq m$. Let moreover $\widehat{\mathbf{e}}_i$ be the projection of $\mathbf{e}_\nu$ to $\mathbf{ue}_i$, i.e., the prefix of $\mathbf{e}_\nu$ such that for all events in $\widehat{\mathbf{e}}_i$ the respective event with uncertainty is in $\mathbf{ue}_i$. This subtrace with uncertainty is needed to perform the induction proof below. Using the observations in the proof of Lem. 6.2 (b), it is easy to see that $\widehat{\mathbf{e}}_i$ is a realization of $\mathbf{ue}_i$ for all i, $0 \leq i \leq m$. Note that the length of the sequence $\widehat{\mathbf{e}}_i$ is smaller or equal to i.

Let $d_{i,j} = \nu(\mathtt{d}_{i,j})$, for all i, j with $0 \leq i \leq m$ and $0 \leq j \leq n$. We show now the following ($\star$): $\gamma_{i,j}$ is an optimal alignment of $\widehat{\mathbf{e}}_i$ and $\mathbf{f}_\nu|_j$ with cost

$\mathfrak{K}^{lh}(\gamma_{i,j}, \mathbf{ue}_i) = d_{i,j}$, by induction on (i, j). In the following, we freely use the fact that $[P_=]$, $[P_L]$, and $[P_M]$ are correct encodings of $P_=$, P_L, and P_M from Exa. 5.3, cf. [20].

Base case. If $i = j = 0$, then $\mathbf{ue}_i = \emptyset$ and $\gamma_{i,j}$ is the empty sequence, which is the optimal alignment of an empty log trace and an empty process run. We have $d_{i,j} = 0$ by (φ_δ), and also $\mathfrak{K}^{lh}(\gamma_{i,j}, \mathbf{ue}_i) = 0$.

Step case 1. If $i = 0$ and $j > 0$, then the only possibility to match the last transition f_j of $\mathbf{f}_\nu|_j$ is a model step with f_j. By the induction hypothesis, $\gamma_{0,j-1}$ is an optimal alignment of the empty trace and $\mathbf{f}_\nu|_{j-1}$ of cost $\mathfrak{K}^{lh}(\gamma_{0,j-1}, \emptyset) = d_{0,j-1}$. Thus, also $\gamma_{0,j} = \gamma_{0,j-1} \cdot \langle(\gg, f_j)\rangle$ is optimal. We have $d_{0,j} = d_{0,j-1} + \nu([P_M]_j)$ by (φ_δ), and by the choice of our cost functions, $\mathfrak{K}^{lh}(\gamma_{0,j}, \emptyset) = \mathfrak{K}^{lh}(\gamma_{0,j-1}, \emptyset) + P_M(f_j) = d_{0,j-1} + \nu([P_M]_j)$.

Step case 2. If $j = 0$ and $i > 0$, then according to (φ_δ) either (i) $d_{i,0} = \nu([P_L]_i + [P_L]_i \cdot [\theta_{\mathbf{ue}}]_i) + d_{i-1,0}$ and $\gamma_{i,0} = \gamma_{i-1,0} \cdot \langle(e_i, \gg)\rangle$, or (ii) $d_{i,0} = \nu([\kappa_{\mathbf{ue}}]_i) + d_{i-1,0}$ and $\gamma_{i,0} = \gamma_{i-1,0}$. Let ue_i be the event with uncertainty in $\mathbf{ue}$ that matches e_i, and p be such that $(lab(e_i) \colon p) \in \textsc{la}(ue_i)$. By the induction hypothesis, $\gamma_{i-1,0}$ is an optimal alignment of $\widehat{\mathbf{e}}_{i-1}$ and the empty run with cost $\mathfrak{K}^{lh}(\gamma_{i-1,0}, \emptyset) = d_{i-1,0}$. In case (i), $\nu([P_L]_i + [P_L]_i \cdot [\theta_{\mathbf{ue}}]_i) = 3 - ue_i.conf - p$, and a similar case distinction as Eq. (4) but for $[\theta_{\mathbf{ue}}]_i$ ensures that $\nu(\mathtt{drop}_{ue_i}) = \bot$, so that $\mathfrak{K}^{lh}(\gamma_{i,0}, \emptyset) = d_{i-1,0} + \kappa(e_i, \gg) \otimes \theta(e_i, \mathbf{ue}_i)$ as desired, according to our choices for the cost function and realization cost from Sec. 5. If case (ii) applies, we can assume that $\nu([\kappa_{\mathbf{ue}}]_i) < \infty$, so by Eq. (4) we must have $\nu(\mathtt{drop}_{ue_i}) = \top$, and $d_{i,0} = d_{i-1,0} + conf(ue_i)$, by our choice for the realization cost. Requirement (d) implies that $conf(ue_i) < 1$, so ue_i is uncertain. Therefore, $\widehat{\mathbf{e}}_i = \widehat{\mathbf{e}}_{i-1}$ is a realization of $\mathbf{ue}_i$ where e_i is dropped, and $\gamma_{i,0} = \gamma_{i-1,0}$ a valid alignment. According to (φ_δ), $d_{i,0}$ is assigned the minimum of the values corresponding to cases (i) and (ii), so since $\gamma_{i-1,0}$ is optimal, also $\gamma_{i,0}$ is optimal.

Step case 3. If $i, j > 0$, then, since ν satisfies (φ_δ), we can distinguish four cases: (i) $d_{i,j} = \nu([P_L]_i + [P_L]_i \cdot [\theta_{\mathbf{ue}}]_i) + d_{i-1,j}$, (ii) $d_{i,j} = \nu([\kappa_{\mathbf{ue}}]_i) + d_{i-1,j}$, (iii) $d_{i,j} = \nu([P_M]_j) + d_{i,j-1}$, and finally, (iv) $d_{i,j} = \nu(ite([P_=]_{i,j} = 0, [\theta_{\mathbf{ue}}]_i, [P_=]_{i,j} + [P_=]_{i,j} \cdot [\theta_{\mathbf{ue}}]_i)) + d_{i-1,j-1}$. In cases $(i) - (iii)$, we reason similarly as for cases (i) and (ii) in the Step Case 2, and as in Step Case 1, respectively, to show that $\gamma_{i,j}$ is an alignment of $\widehat{\mathbf{e}}_i$ and $\mathbf{f}_\nu|_j$ with cost $\mathfrak{K}^{lh}(\gamma_{i,j}, \mathbf{ue}_i) = d_{i,j}$. In case (iv), by the induction hypothesis, $d_{i-1,j-1}$ is the cost of the optimal alignment for $\widehat{\mathbf{e}}_{i-1}$ and $\mathbf{f}_\nu|_{j-1}$. By construction, $\gamma_{i,j} = \gamma_{i-1,j-1} \cdot (e_i, f_j)$. A similar case dis-

tinction as Eq. (4) but for $[\theta_{\mathbf{ue}}]_i$ ensures that $\nu(\mathtt{drop}_{ue_i}) = \bot$, so e_i is included in $\widehat{\mathbf{e}}_i$, so $\gamma_{i,j}$ is a valid alignment for $\widehat{\mathbf{e}}_i$. By Eq. (1), we have $\mathfrak{K}^{lh}(\gamma_{i,j}, \mathbf{ue}_i) = d_{i-1,j-1} + \kappa(e_i, f_i) \otimes \theta(e_i, \mathbf{ue}_i)$, and by Eq. (2), $\kappa(e_i, f_i) \otimes \theta(e_i, \mathbf{ue}_i) = \nu(ite([P_{=}]_{i,j} = 0, [\theta_{\mathbf{ue}}]_i, [P_{=}]_{i,j} + [P_{=}]_{i,j} \cdot [\theta_{\mathbf{ue}}]_i))$, so $\mathfrak{K}^{lh}(\gamma_{i,j}, \mathbf{ue}_i)$ is the cost of $\gamma_{i,j}$.

According to (φ_δ), $d_{i,j}$ is assigned the minimum of the values corresponding to cases $(i) - (iv)$, so $\gamma_{i,j}$ is optimal, which concludes the induction proof.

We can assume that an optimal alignment γ exists where the process run $\gamma|_M$ has exactly length n. While Lem. 6.1 guarantees that there is some γ such that $\gamma|_M \leq n$, we can assume $\gamma|_M \geq n$ if for all α the final marking M_F admits a step $(M_F, \alpha) \xrightarrow{(t,\beta)} (M_F, \alpha)$ with a silent transition t. Such transitions can always be added to the net $\mathcal{N}$. Thus, the claim of the theorem follows from case $i = m$, $j = n$ of $(\star)$, as $\mathbf{ue}_m = \mathbf{ue}$ and hence $\widehat{\mathbf{e}}_m = \mathbf{e}_\nu$. □

As explained in Sec. 5, our cost model can also accommodate the best-realization cost function from [11]. Then, a solution to (Φ) yields a realization of **ue** that has minimal optimal alignment cost wrt. κ. The following lemma formalizes this property, it is proven in a similar way as Thm. 6.1.

Lemma 6.3. *For $\mathcal{N}$, $\boldsymbol{ue}$ as above and $\gamma_{m,n}$ the alignment decoded from a satisfying assignment ν for (Φ) as in Sec. 6.5, there is no realization $\mathbf{e}$ of $\boldsymbol{ue}$ and alignment γ for $\mathbf{e}$ such that $\kappa(\gamma) < \kappa(\gamma_{m,n})$.*

7. Implementation

We implemented our approach on top of `cocomot` [20]. Though some basic functionalities could be reused, the encoding and decoding was designed and written from scratch, and an additional solver was added as backend. For clarity, we will in the sequel refer to the extended tool by $\mathtt{cocomot}_{\mathtt{u}}$. Its source code, as well as all data sets, are publicly available.[1]

$\mathtt{cocomot}_{\mathtt{u}}$ implements conformance checking for traces with the four types of uncertainty: uncertain events, uncertain activities, uncertain data (both discrete or continuous), and uncertain timestamps. The tool `cocomot` is a Python command line script: it takes as input a DPN (as `.pnml` file) and a log (as `.xes`) as input and computes the optimal alignment distance for

[1] https://github.com/bytekid/cocomot

every trace in the log; in verbose mode, it additionally prints an optimal alignment for every trace. Input files with uncertainty must conform to the extension of the XES standard proposed in [42]. The option `-u` activates the uncertainty mode, and takes an argument to select one of two cost functions that are implemented:

– the *likelihood* cost function specified in Fig. 2, which is selected by setting the parameter `-u like`; and
– the *best-realization* cost function from Ex. 5.2 that computes an optimal alignment for a best-case realization, which is selected by setting the parameter `-u real`. With this cost function, $\texttt{cocomot}_\texttt{u}$ implements the conformance checking task introduced in [11].[2]

SMT solvers. Our tool offers three solvers as backends: Z3 [22], Yices 2 [21], and OptiMathSAT [39]. All solvers are interfaced via their Python bindings. While Z3 and OptiMathSAT have an optimization scheme built in, Yices does not. For Yices, we thus implemented a simple custom optimization strategy, which subsequently tries to find an alignment of cost 0, 1, 2, etc. In the sequel, we call this the *incremental optimization* strategy. However, this strategy is only feasible if the optimal alignment cost is guaranteed to be a natural number. Thus, we use incremental optimization only with the best-realization cost function. Based on the performance comparison described in the next section, the following solver backends are used by default: with the best-realization cost function, we use Yices with incremental optimization; with the likelihood cost function instead Z3 as is. For Z3 we activate two options that are offered by the library: we use the *symba* optimization engine, as the *basic* optimization engine caused timeouts on some traces; and we set the option called *incremental* since this resulted in a slight speedup. The solver can be controlled with the command line option `-s`, which takes one of the values `yices`, `z3`, `z3-inc`, `optimathsat`, `optimathsat-inc`, where the suffix `inc` triggers incremental optimization.

Length bound. As described in Sec. 6, our encoding-based approach crucially relies on the fact that an upper bound on the length of the model run in an optimal alignment can be computed (cf. Lem. 6.1). In the implementation, we heuristically approximate this bound by taking the sum of the length of

[2] In fact the tool Proved [11] computes the optimal alignment for both the best-case and the worst-case realization; the latter is not implemented in $\texttt{cocomot}_\texttt{u}$.

the trace, the length of the shortest model run of the Petri net without data, and a constant depending on the DPN size.

Performance optimizations. The implementation is based on the encoding described in Sec. 6, though with two optimizations:

For SMT solvers, it is often beneficial to deal with less complex expressions, even at the price of more variables. We thus used additional variables to abbreviate complex subexpressions, adding their defining equations separately. It can be easily shown that this only changes the formula structure and that the resulting formula is equisatisfiable. Thus, correctness is not compromised. Additional variables are especially useful to create shorthand variables for expressions that occur multiple times in the encoding, for instance the cost of a log step at index i or the cost of a model step at index j, which occur j resp. i times in the encoding of the alignment matrix.

Moreover, in the encoding we add the additional constraint that a log step never occurs right before a model step. This eliminates symmetries in the search space, removing redundant solutions, but it is easily to prove that it does not hamper correctness. Indeed, the cost of an alignment where a log step occurs before a model step is the same as the cost of the alignment where the two steps are swapped. This reduces the conformance checking time by about 5%.

8. Experiments

In this section we address the following questions experimentally:

(Q1) **Comparison to behavior-net-based approach**: How does the SMT-based approach of `cocomot`$_{\mathtt{u}}$ perform in comparison with the approach from [11] to find optimal alignments for the best-case realization?

(Q2) **Impact of uncertainty on performance**: How does the performance of `cocomot`$_{\mathtt{u}}$ change with varying degrees of uncertainty of different types in the input, for both the best-realization and likelihood cost functions?

(Q3) **Comparison to explicit unfolding**: How does the SMT-based approach of `cocomot`$_{\mathtt{u}}$ to find best-realization alignments compare with the SMT-based approach without uncertainty (standard `cocomot`) run on all possible realizations?

(Q4) **Solver backends**: Which SMT solvers perform best as backends with the two cost functions, respectively?

	# traces	max. length	DPN size	# vars
(a) road fines	4290	20	9/19	8
(b) sepsis	94	8	48/36	3
(c) hospital billing	621	12	34/36	4
(d) artificial	100	10	15/16	0

Table 1: Characteristics of data sets.

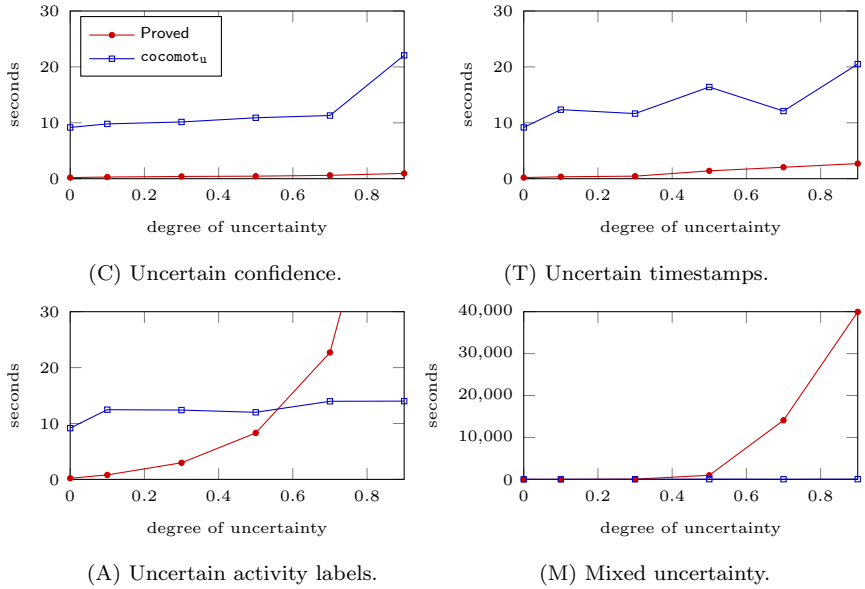
Data sets. Before addressing these questions, we describe the data sets used in experiments. We used the following real-world event logs with data, for which DPN models are known: (a) the road fine data set [13], (b) the sepsis data set [15], and (c) the hospital billing data set [15]. For (a), we restricted to a subset that takes a representative from each cluster, using the trace clustering technique from [20], to reduce the number of traces while maintaining a representative subset. For the latter two, `cocomot` times out on some traces of the original data set. To keep evaluation time within bounds, we thus considered subsets of the original logs, restricting to traces up to a fixed length. To compare with Proved, we also used the artificial data set (d) generated within the experiment script[3] provided by the authors of [11]. Tab. 1 summarizes these data sets, reporting its number of traces, the maximal trace length, the size of the respective DPN in terms of places/transitions, and the number of data variables in the DPN.

To obtain event logs with uncertainty, we augmented data sets (a)–(c) by adding random uncertainty features of different kinds. We consider four types of uncertainty, namely (C) uncertain events as indicated by confidence values, (A) uncertain activities, (D) uncertainty with respect to data values, (T) uncertain timestamps, and (M) the combination of all these. By the *degree of uncertainty* of a log, we refer in the sequel to the ratio of events that exhibit uncertainty of one type, which is thus a value between 0 and 1. To inject uncertainty of degree r into a given log, we modified the log as follows, depending on the uncertainty type:

(C) A random confidence value was added with probability r to each event.

(A) With probability r, the activity b of an event was replaced by a set of activity labels $\{b\colon p_0, b_1\colon p_1, b_2\colon p_2\}$, where b_1 and b_2 are two other

[3] https://github.com/proved-py/proved-core/blob/Conformance_Checking_over_Uncertain_Event_Data/experiments



(C) Uncertain confidence.

(T) Uncertain timestamps.

(A) Uncertain activity labels.

(M) Mixed uncertainty.

Figure 3: Comparison of cocomot$_u$ (best-realization cost) and Proved.

labels that occur in the log, and p_0, p_1, p_2 are chosen randomly such that they add up to 1.

(D) For every variable x in the domain of the assignment α associated with an event, with probability r, the value $v = \alpha(x)$ is replaced by $\{v, v_0, v_1\}$, where v_1 and v_2 are two additional values randomly chosen from the value range of x in the log.

(T) With probability r, the timestamp t of an event e was replaced by a range $[t - \Delta/2, t + \Delta/2]$, where Δ is the timespan of e's trace.

(M) All the above modifications are combined. Note that as a result, the probability that *some* kind of uncertainty is attached to an event e is much higher than r (namely $1 - (1 - r)^4$).

All of the below experiments were run single-threaded on a 12-core Intel i7-5930K 3.50GHz machine with 32GB of main memory.

(Q1) Comparison with behavior-net based approach. We first investigate how cocomot$_u$ compares with the tool Proved [11] for the task considered therein, i.e., finding the optimal alignment for the best-case realization. To this end, we used the experiment script provided by the authors of [11], which first creates benchmark (d) mentioned above and then performs conformance checking. In this script, the benchmarks are created as follows: starting from

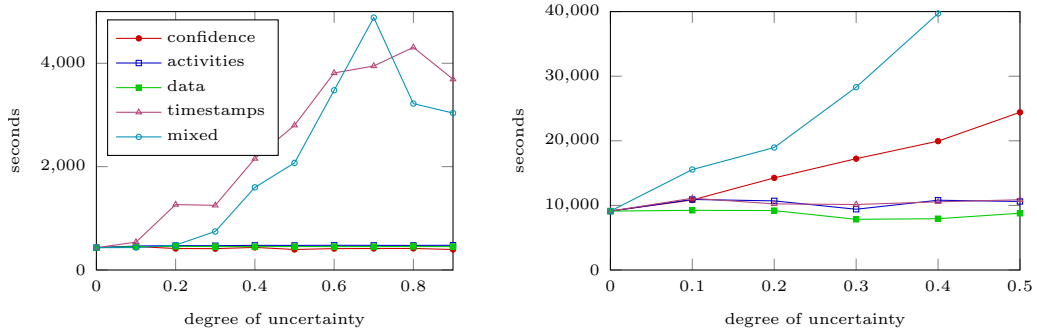
a fixed set of artificially created Petri nets, the script generates playouts to obtain logs of desired size. It then adds random "deviations" to obtain not only perfectly fitting traces, and injects random uncertainty of a desired kind (uncertain events, activities, timestamps, or all of these). In this way, a fixed-size data set with a desired degree of uncertainty is obtained.

We instrumented this script to export the generated logs with uncertainty in xes format, ran `cocomot`$_{\mathtt{u}}$ with the best-realization cost function on these logs, and compared the results. Most parameters of the script were left as they were: we again used the set of nets of size 10 and generated 100 traces per net. We fixed one deviation type, though, namely a mix of the three available features (new activity labels, swaps, and extra events), with a coefficient of 0.1 each; since the comparison of different deviation types is not of interest here. We considered the same four types of uncertainty as in [11], namely (C), (A), (T), and (M) as described above. (The approach of [11] does not support data, so (D) is not applicable.) The results are shown in Fig. 3, where the runtimes of the two tools are compared for the four types of uncertainty. The y-axes give the runtimes of the tools on the entire log in seconds, while the x-axes give the degree of uncertainty between 0 and 0.9. For each data point, the results are averaged over three runs, i.e., over three different logs that are random-generated as described above.

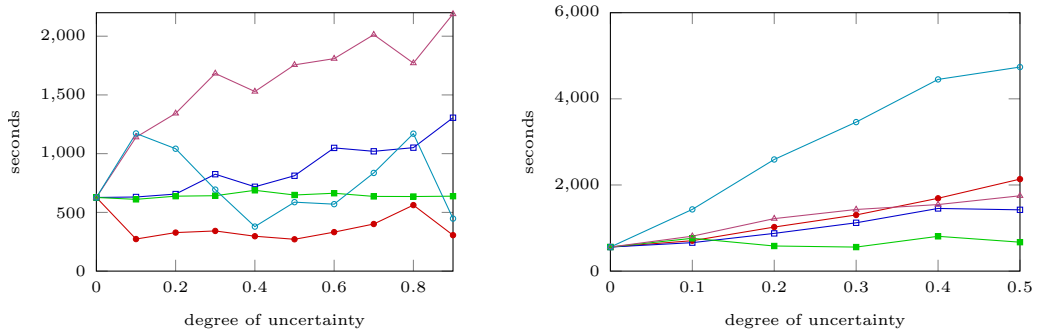
It should be noted that Proved computes the optimal alignment of both the best-case realization and the worst-case realization within the given time, by inspecting all realizations. The latter cannot be computed by `cocomot`$_{\mathtt{u}}$, since the problem is one of max-min optimization, rather than minimization only. Overall, it can be noted that Proved is faster if the log has only confidence values, or low degrees of uncertainty, as they were evaluated in [11]. On the other hand, `cocomot`$_{\mathtt{u}}$ scales much better for high degrees of uncertainty. For instance, to conformance check 100 traces with mixed uncertainty of degree 0.7, Proved required on average almost four hours, whereas `cocomot`$_{\mathtt{u}}$ needs only 38 seconds. Although Proved solves a more comprehensive task within this time, the magnitude of the difference can hardly be explained by this fact alone. A possible reason for the reduced performance is that behaviour nets get more complex if more events have some kind of uncertainty, so that also the product nets to compute alignments become larger. In the experiments of [11], only comparatively low degrees of uncertainty were explored (up to 0.15), so this effect was likely not observed.

(Q2) Impact of uncertainty on performance. Next, we investigate how injection of uncertainty of differents kinds into real-world logs with data affects performance of $\texttt{cocomot}_\texttt{u}$. To this end, we consider the data sets (a)–(c) and uncertainty types (C), (A), (D), (T), and (M) described above, with both cost functions.

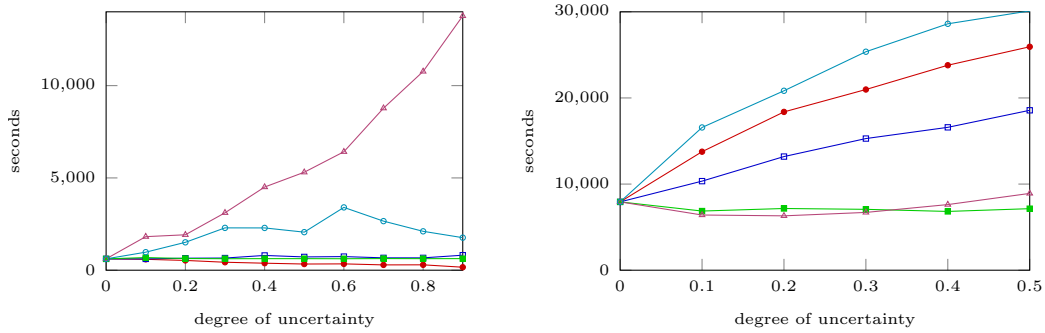
In Fig. 4 we show the results for $\texttt{cocomot}_\texttt{u}$ with cost function best-realization (left), and likelihood (right) for the three data sets. The y-axes show the conformance checking time for the entire data set in seconds. The x-axes show the degree of uncertainty added, ranging from 0 to 0.9 for best-realization, and from 0 to 0.5 for likelihood. The reason for less uncertain events being added in the second case is that the performance impact is already considerable for 0.5. We see that the impact of the different types of uncertainty is very different, and depends also on the cost function. For best-realization, we observe the following: (C) Adding confidence values tends to improve performance, likely because the best-realization cost can only be smaller when adding uncertain events, so that the solver's search time decreases. However, there is also a tradeoff with a slightly more complex encoding. With degree of uncertainty 0.9, the required time decreases by 13% (road fines) and 70% (hospital billing), respectively, though for the sepsis data we noted a very slight increase. (A) Uncertain activity labels somewhat increase conformance checking time, probably due to an increased search space and a somewhat more complex encoding, though the increase is limited. With degree of uncertainty 0.1, the required time increases by only 3-5%. Even with degree of uncertainty 0.9, the required time increases by less than 5% (road fines), 50% (sepsis), and 30% (hospital billing). (D) Uncertainty with respect to data values hardly makes a difference, probably because changes in data have no impact on optimal alignments with the cost function best-realization. (T) Adding uncertain timestamps considerably or even drastically deteriorates performance. This is to be expected, since the encoding with uncertain timestamps is considerably more complex (cf. Sec. 6.3). With degree of uncertainty 0.1, the required time increases by 25% (road fines), 80% (sepsis), and more than 150% (hospital billing). With degree of uncertainty 0.9 (i.e., the event order in traces is basically lost), the required time multiplies by a factor of 6 (road fines), a factor of 3 (sepsis), and a factor of 7 (hospital billing). (M) Interestingly, performance deteriorates considerably less when all types of uncertainties are combined. We assume that in this case the deterioration caused by uncertain timestamps is mitigated by the positive effect of confidence values.



(a) Road fine data set with best-realization cost (left) and likelihood cost (right).

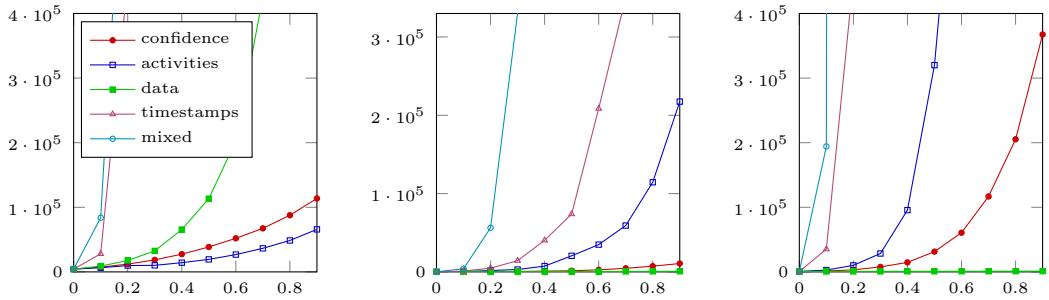


(b) Sepsis data set with best-realization cost (left) and likelihood cost (right).



(c) Hospital billing data set with best-realization cost (left) and likelihood cost (right).

Figure 4: Conformance checking with increasing degree of uncertainty.



(a) Road fines data set. (b) Sepsis data set. (c) Hospital billing data set.

Figure 5: Number of realizations with increasing degree of uncertainty.

With the likelihood cost function, the impact of uncertainty on performance is generally much higher. (C) In contrast to best-realization, with likelihood the presence of confidence values in events increases the required time, likely because in contrast to best-realization, confidence values do not allow for simpler alignments as events can be skipped; instead, the cost function gets considerably more complex since other expressions in the cost function get multiplied by a value depending on the confidence. For degree of uncertainty 0.5, the required time is multiplied by a factor of 2-3 on all data sets. (A) Performance deteriorates much less by adding uncertain activity labels, by only 10-20%, probably because uncertain activities in the cost function just add more alternatives, but do not introduce multiplication. (D) Uncertain data has a neglible effect on performance even with the likelihood cost function, likely because (mis)matching data values are hardly determinative for optimality of an alignment on these data sets. (T) Adding uncertain timestamps increases the conformance checking time for all data sets, which is to be expected since the encoding gets more complex. The effect varies for the data sets though, a degree of uncertainty 0.5 causes an increase by 20% for the road fine benchmark, by 200% for the sepsis benchmark, and by 12% for the hospital benchmark. (M) As is to be expected, introducing all kinds of uncertainty causes even more slowdown.

(Q3) Comparison to explicit unfolding. In order to compute the minimal optimal alignment among all realizations of an uncertain trace, instead of employing the SMT-based approach for uncertain traces presented in this paper with the best-realization cost function, one could also compute all possible realizations, and apply to these the SMT-based approach for traces

		cocomot			cocomot$_u$	
	uncertainty	*# realizations*	*time*	*TMO*	*time*	*TMO*
(a) road fines	(C) 0.1	7360	9886	166	454	0
	(A) 0.1	6505	5115	184	464	0
	(T) 0.1	27923	∞		539	0
(b) sepsis	(C) 0.1	185	5122	1	273	0
	(A) 0.1	392	44646	1	631	0
	(D) 0.1	295	1062	1	611	0
	(T) 0.1	1076	∞		1140	0
(c) hospital billing	(C) 0.1	1346	1638	0	615	0
	(A) 0.1	2797	6099	0	614	0
	(D) 0.1	675	1629	0	685	0
	(T) 0.1	38191	∞		1818	0

Table 2: Comparison of cocomot with cocomot$_u$ with best-realization cost.

without uncertainty from [20], i.e., the standard cocomot tool. We next study how these two approaches compare with respect to performance.

We again consider the data sets (a)–(c) described above. First, Fig. 5 illustrates how many realizations the traces in these three logs exhibit when augmented with varying degrees of uncertainty. Indeed, the number of realizations grows exponentially, and soon becomes hard to compute explicitly, especially for uncertain timestamps and mixed uncertainty. It is thus not surprising that cocomot run on all realizations is much slower than cocomot$_u$, even for comparatively low degrees of uncertainty. Tab. 2 illustrates this observation for some data sets and uncertainty of different kinds. Here, the columns refer to the type of uncertainty added to the log, the number of realizations, the conformance checking time in seconds, and the number of SMT solver timeouts (600 seconds). An ∞ symbol indicates that cocomot ran for more than 24h without completing the conformance check of the log. The table clearly shows that cocomot$_u$ outperforms by far the approach applying cocomot to all realizations.

(Q4) Solver backends. Tab. 3 and Tab. 4 show the performance of cocomot$_u$ using different solver backends with the best-realization and likelihood cost functions. The columns OptiMathSAT and Z3 refer to the respective solvers using their built-in optimization schemes. In the columns labeled (inc), the incremental optimization search strategy was employed for Yices, OptiMathSAT, and Z3, respectively (cf. Sec. 7). The columns *enc* and *slv* refer to the total encoding and solving time in seconds, while *mem* gives the approx-

	Yices (inc)			OptiMathSAT			Z3			OptiMathSAT (inc)			Z3 (inc)		
	enc	*slv*	*mem*	*enc*	*slv*	*mem*	*enc*	*slv*	*mem*	*enc*	*slv*	*mem*	*enc*	*slv*	*mem*
(a) baseline	142	295	2.7	130	1673	16	1269	1530	1.5	133	1105	1.5	1325	1292	1.5
(b) baseline	9	608	2	12	1846	2.5	143	1628	1.5	11	1682	1.5	152	6713	1.5
(c) baseline	40	538	2	54	3177	4.8	595	2886	1.5	53	3634	1.6	709	6381	1.5

Table 3: Comparison of solvers (best-realization cost).

	OptiMathSAT				Z3			
	enc	*slv*	*mem*	*TMO*	*enc*	*slv*	*mem*	*TMO*
(a) baseline	–	–	30	–	1757	10459	1.6	5
(b) baseline	6270	23980	2.5	18	185	254	1.5	0
(c) baseline	1227	173363	5.6	214	770	6335	1.5	0

Table 4: Comparison of solvers (likelihood cost).

imate memory consumption in GB. For the evaluation with the likelihood cost function, we imposed a timeout of 600 seconds for every trace (with the best-realization cost function, solvers generally need much less time, and no timeout was needed). We used the data sets (a) road traffic billing, (b) sepsis, and (c) hospital billing described above without uncertain events (but using the uncertainty encoding, as opposed to the one from standard `cocomot`).

The tables illustrate that overall the encoding time is negligible compared to the solving time (except for Yices perhaps, but in this case also the solving time is small). With the best-realization cost function, it turns out that Yices clearly outperforms the other tools despite the quite naive incremental optimization scheme, being at least a factor of two faster. Thus, $\texttt{cocomot}_\texttt{u}$ defaults to Yices with this cost function. While OptiMathSAT performs comparably with Z3 with best-realization, it is by far slower than Z3 with the likelihood cost function. In fact, on data set (a), OptiMathSAT had to be stopped after having processed about 75% of the trace because the available memory was consumed. We tried to adjust solver settings but could not improve this behavior. By default, $\texttt{cocomot}_\texttt{u}$ thus uses Z3 with the likelihood cost function.

9. Conclusions

In this work we extend the theoretical framework for alignment-based conformance checking of (deterministically known) data-aware processes over logs with uncertain data studied in [23]. We consider four types of uncertain data following the taxonomy of uncertainty types proposed in [17, 11]: uncertain events, uncertain timestamps, uncertain activities and uncertain

(generic) data values. Such uncertainties are accounted for as additional annotations provided for respective attributes in the log. Using data Petri nets [13, 14] as the reference process model, the alignment computation per trace is then encoded as an SMT instance with an objective function (describing the optimal distance between the trace and a model run) to be minimized. This theoretical framework is, to the best of our knowledge, generalizes all existing approaches dealing with uncertain log data in the context of conformance checking.

The extension provides three essential results demonstrating the feasibility of the approach proposed in [23]. First, we present formal guarantees of the correctness of the problem encoding in SMT. Specifically, we show that our approach is capable of producing trace realizations and, given a concrete distance function, the alignments it produces are optimal. Second, we implement the approach by extending CoCoMoT – our previously developed tool for SMT-based conformance checking [20]. Third, we provide a thorough experimental evaluation to assess how feasible is our approach in practice. To this end, we use publicly available and synthetically generated event logs and DPN models. As a side result, the artificially generated models and respective logs can be used as benchmarks for other process mining approaches for data-aware processes. We also compare our tool to the one from [11], test its performance using various SMT solvers and demonstrate experimentally the advantage of the direct encoding of uncertainty realization by comparing the tool runtime against the results produced by the tool from [20] ran over traces with resolved uncertainties.

The results presented in this paper open up multiple future work directions. Given the feasibility and relatively good computational speed, the approach can be extended towards the support of process models with uncertainties. Like that, similarly to [32], one can study the conformance checking problem for a stochastic extension of data Petri nets and use an augmented SMT encoding to compute (optimal) alignments.

The application of machine learning techniques also seems like a promising future direction. While our encoding allows for concrete, automatic realization of uncertain event data, such uncertainties are resolved while computing the optimal alignments using the SMT encoding, and such optimality is only subject to the objective function encoding the distance between a given trace and the process model. However, one cannot guess the "real" realization, that is, the most preferred one in the given system enactment context. Moreover, although SMT solvers are very efficient and our approach

offers a unified way for resolving all types of mentioned uncertainties in logs, the experimental evaluation demonstrated that the performance drops when dealing with timestamp uncertainties. To address these two issues the usage of automated reasoning techniques is not enough and one might need to resort to classification approaches coming from the machine learning domain. Like that, our tool can be used in the process of training a chosen classifier as follows. Given a trace, the conformance checker proposes a concrete realization for the best alignment, which is then decoded and presented to the domain expert. They evaluate the correctness/suitability of the realization and send their feedback back to the classifier. Feedback results can be also accommodated in the shape of concrete attribute realizations that can be suitably added to the SMT encoding. The process can be repeated until the domain expert accepts the best alignment and the realization (pattern) is recorded by the machine learning framework at hand.

Acknowledgments. This research has been partially supported by the Italian Ministry of University and Research (MUR) under the PRIN project PINPOINT Prot. 2020FNEB27, and by UNIBZ under the projects ADAPTERS, MENS, and SMART-APP.

References

[1] van der Aalst Wil M. P. et al., Process mining manifesto, in: Proc. of BPM Workshops 2011, Springer, 2011, pp. 169–194.

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

[3] A. Adriansyah, B. F. van Dongen, W. M. P. van der Aalst, Conformance checking using cost-based fitness analysis, in: Proc. EDOC 2011, IEEE Computer Society, 2011, pp. 55–64.

[4] A. Adriansyah, N. Sidorova, B. F. van Dongen, Cost-based fitness in conformance checking, in: B. Caillaud, J. Carmona, K. Hiraishi (Eds.), Proc. ACSD 2011, IEEE Computer Society, 2011, pp. 57–66.

[5] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, W. M. P. van der Aalst, Alignment based precision checking, in: M. L. Rosa, P. Soffer (Eds.), Proc. BPM Workshops 2012, Vol. 132 of Lecture Notes in Business Information Processing, Springer, 2012, pp. 137–149.

[6] M. T. Wynn, S. W. Sadiq, Responsible process mining - A data quality perspective, in: Proc. BPM 2020, Springer, 2019, pp. 10–15.

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

[8] A. Polyvyanyy, A. A. Kalenkova, Conformance checking of partially matching processes: An entropy-based approach, Inf. Syst. 106 (2022) 101720.

[9] G. Bergami, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic trace alignment, in: Proc. of ICPM 2021, IEEE, 2021, pp. 9–16.

[10] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, Inf. Syst. 109 (2022) 102033.

[11] M. Pegoraro, M. S. Uysal, W. M. P. van der Aalst, Conformance checking over uncertain event data, Inf. Syst. 102 (2021) 101810.

[12] F. Chesani, P. Mello, R. De Masellis, C. Di Francescomarino, C. Ghidini, M. Montali, S. Tessaris, Compliance in business processes with incomplete information and time constraints: a general framework based on abductive reasoning, Fundam. Informaticae 161 (1-2) (2018) 75–111.

[13] F. Mannhardt, M. de Leoni, H. Reijers, W. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437.

[14] M. de Leoni, P. Felli, M. Montali, A holistic approach for soundness verification of decision-aware process models, in: Proc. 37th International Conference on Conceptual Modeling, Vol. 11157 of LNCS, 2018, pp. 219–235.

[15] F. Mannhardt, Multi-perspective process mining, Ph.D. thesis, Technical University of Eindhoven (2018).

[16] I. Cohen, A. Gal, Uncertain process data with probabilistic knowledge: Problem characterization and challenges, in: Proc. BPM Workshops 2021, Vol. 2938 of CEUR Workshop Proceedings, CEUR-WS.org, 2021, pp. 51–56.

[17] M. Pegoraro, W. M. P. van der Aalst, Mining uncertain event data in process mining, in: ICPM 2019, IEEE, 2019, pp. 89–96.

[18] C. W. Barrett, C. Tinelli, Satisfiability modulo theories, in: Handbook of Model Checking, Springer, 2018, pp. 305–343.

[19] S. Abiteboul, P. C. Kanellakis, G. Grahne, On the representation and querying of sets of possible worlds, in: U. Dayal, I. L. Traiger (Eds.), Proc. SIGMOD Conference 1987, ACM Press, 1987, pp. 34–48.

[20] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Cocomot: Conformance checking of multi-perspective processes via SMT, in: Proc. BPM 2021, Springer, 2021, pp. 217–234.

[21] B. Dutertre, Yices 2.2, in: Proc. CAV 2014, 2014, pp. 737–744.

[22] L. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: Proc. TACAS 2008, 2008, pp. 337–340.

[23] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Conformance checking with uncertainty via SMT, in: C. D. Ciccio, R. M. Dijkman, A. del-Río-Ortega, S. Rinderle-Ma (Eds.), Proc. BPM 2022, Vol. 13420 of Lecture Notes in Computer Science, Springer, 2022, pp. 199–216.

[24] C. C. Aggarwal, Managing and Mining Uncertain Data, Vol. 35 of Advances in Database Systems, Kluwer, 2009.

[25] L. Li, H. Wang, J. Li, H. Gao, A survey of uncertain data management, Front. Comput. Sci. 14 (1) (2020) 162–190.

[26] J. Chandra, J. C. Bose, R. S. Mans, W. M. P. van der Aalst, Wanna improve process mining results?, in: CIDM 2013, IEEE, 2013, pp. 127–134.

[27] H. van der Aa, H. Leopold, M. Weidlich, Partial order resolution of event logs for process conformance checking, Decis. Support Syst. 136 (2020) 113347.

[28] K. Goel, S. J. J. Leemans, N. Martin, M. T. Wynn, Quality-informed process mining: A case for standardised data quality annotations, ACM Trans. Knowl. Discov. Data 16 (5) (2022) 97:1–97:47.

[29] X. Lu, D. Fahland, W. M. P. van der Aalst, Conformance checking based on partially ordered event data, in: F. Fournier, J. Mendling (Eds.), Proc. BPM Workshops 2014, Vol. 202 of Lecture Notes in Business Information Processing, Springer, 2014, pp. 75–88.

[30] S. J. Leemans, S. J. van Zelst, X. Lu, Partial-order-based process mining: a survey and outlook, Knowledge and Information Systems (2022) 1–29.

[31] W. M. P. van der Aalst, L. F. R. Santos, May I take your order? - on the interplay between time and order in process mining, in: A. Marrella, B. Weber (Eds.), Proc. BPM Workshops 2021, Vol. 436 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 99–110.

[32] E. Bogdanov, I. Cohen, A. Gal, Conformance checking over stochastically known logs, in: C. D. Ciccio, R. M. Dijkman, A. del-Río-Ortega, S. Rinderle-Ma (Eds.), Proc. BPM Forum 2022, Vol. 458 of Lecture Notes in Business Information Processing, Springer, 2022, pp. 105–119.

[33] M. Pegoraro, B. Bakullari, M. S. Uysal, W. M. P. van der Aalst, Probability estimation of uncertain process trace realizations, in: J. Munoz-Gama, X. Lu (Eds.), Process Mining Workshops - ICPM 2021, Revised Selected Papers, Vol. 433 of Lecture Notes in Business Information Processing, Springer, 2021, pp. 21–33.

[34] M. Solé, J. Carmona, Encoding process discovery problems in SMT, Softw. Syst. Model. 17 (4) (2018) 1055–1078.

[35] M. Boltenhagen, T. Chatain, J. Carmona, Encoding conformance checking artefacts in SAT, in: Proc. BPM Workshops 2019, 2019, pp. 160–171.

[36] M. Boltenhagen, T. Chatain, J. Carmona, Optimized SAT encoding of conformance checking artefacts, Computing 103 (2021) 29–50.

[37] M. de Leoni, W. M. P. van der Aalst, Data-aware process mining: discovering decisions in processes using alignments, in: Proc. 28th SAC, 2013, pp. 1454–1461.

[38] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.6, Tech. rep., Available at: http://smtlib.cs.uiowa.edu/language.shtml (2018).

[39] R. Sebastiani, P. Trentin, OptiMathSAT: A tool for optimization modulo theories, Journal of Automated Reasoning (2018).

[40] R. Sebastiani, S. Tomasi, Optimization modulo theories with linear rational costs, ACM Trans. Comput. Log. 16 (2) (2015) 12:1–12:43.

[41] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Data-aware conformance checking with SMT, Information Systems 117 (2023).

[42] M. Pegoraro, Process mining on uncertain event data (extended abstract), in: Proc. ICPM-D 2021, CEUR, 2021, pp. 1–2.