International Journal of Innovative Computing, Information and Control
Volume **8**, Number **7(B)**, July **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 5317–5339

A GENERAL FRAMEWORK FOR PRECISION CHECKING

Jorge Munoz-Gama and Josep Carmona

Department of Software
Universitat Politecnica de Catalunya
C. Jordi Girona, 1. 08034, Barcelona, Spain
{ jmunoz; jcarmona }@lsi.upc.edu

Received March 2011; revised October 2011

Abstract. *Process conformance is the domain within Process Mining that addresses the adequacy between a model and a log of a system. It has four different dimensions: fitness, precision, generalization and structure. This paper presents a metric to evaluate the precision dimension: the extra behavior a formal model tolerates when confronted to a log. Additionally, two important factors are presented together with the metric value: confidence (an estimation of the stability of the metric value for a future window), and a severity assessment to the imprecisions detected. Several techniques are described to accomplish this, including a log-guided traversal of the model, the optimization of binary linear programming models to estimate the confidence, and multi-factored methods to determine the severity. The approach is implemented within an open-source Process Mining platform, and experimental results certify both the significance of the approach and the usefulness of the new factors proposed.*

Keywords: Process mining, Conformance checking, Business process management

1. Introduction.

Process mining is a discipline that combines formal techniques that address the process perspective of a system, and data mining [1]. It has been an emerging area in the last decade. The reason for the success of process mining is simple: there is a lack of automation and formal analysis in the way nowadays information systems are designed and maintained. We live in a world where there is an exponential growth of data, and where software engineers must necessarily adapt to an always evolving situation. Process mining techniques are based on extracting information from *logs*, where a log is a set of traces observed in the execution of a system. These techniques address the process perspective of an information system in the following scopes: *process discovery* (find a formal model to represent a log), *process conformance* (evaluate the adequacy of a formal model in representing a log) and *process extension* (incorporate enhancements into the model). The work presented in this paper presents techniques for process conformance.

Process conformance comprises four dimensions [2]. The first dimension is called *fitness*, and addresses how much of the log is captured by the process model. The *precision* dimension instead prioritizes these models that represent as closely as possible the log. In contrast, the *generalization* dimension prioritizes models having some degree of abstraction in representing the log. Finally, the *structure* dimension refers to models minimal in structure which reflect the log.

In this work we propose techniques for estimating the precision of a formal model in representing a log. Importantly, the approach presented in this paper estimates the effort needed to achieve a better model (more precise), and identifies the mismatches between log and model. This may speed-up the modeling stage by iterative evaluation and enhancement of the processes within a system. Additionally, we incorporate discussions on how to adapt the techniques to models that are popular in *Process-Aware Information*

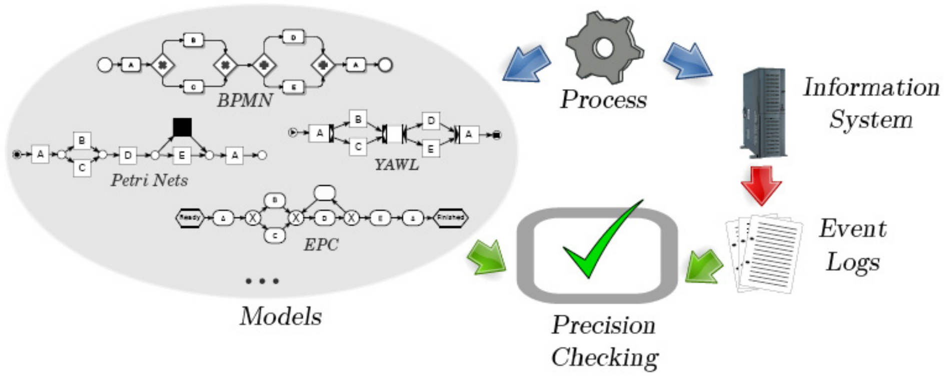


FIGURE 1. A general framework for precision checking

Systems such as Petri nets [3], YAWL [4], EPC [5] and BPMN [6]. In the remaining of the section, the general setting and a simple example motivating the technique of this paper are provided respectively.

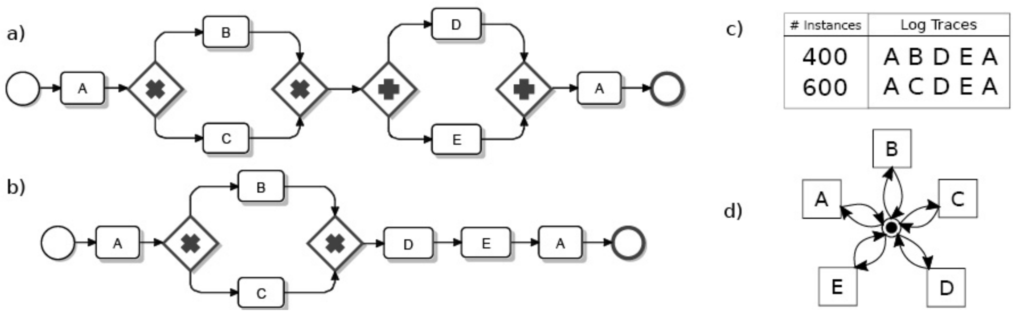
1.1. **Checking precision.** Figure 1 describes the general scenario for checking precision: the system under consideration has some processes that represent its modus operandi. On top of these processes, there is an information system that is meant to coordinate their execution according to a predefined specification. An information system can be monitored in order to record its activities in a log, from which the techniques presented in this paper will start. Additionally to logs, there are the models used to formally represent the processes in a system. These models can be obtained through manual design, or from process discovery techniques [7]. Once a model and a log are available, precision checking will quantify how precise the model is in describing a log. This metric is crucial in evaluating the real gap between specification and implementation of a system [8].

The contributions of this paper are the following[1].

- A novel technique for robust precision checking which evaluates the effort needed in order to reach a more precise model, and which can be instantiated on different models (Petri nets, YAWL, EPCs, BPMN). The technique is based on a *log-guided traversal* of the model behavior, thus avoiding the full exploration of the state space of the model. This is a significant alleviation with respect to related approaches.
- The metric is equipped with a *confidence interval*, based on an estimation of how the precision will change in the future.
- Imprecisions in the model are detected, and their *severity* quantified. This enables selecting the deviations in the model that should be tackled first, but also provides an elegant approach for reporting the precision dimension.
- For the case of Petri nets, all the techniques of this paper have been fully implemented as a ProM 6 plugin.[2] Together with the experiments reported in [9, 10], the new experiments reported in this paper witness the importance and generality of the precision checking techniques proposed.

1.2. **Motivating example.** The example illustrates a possible payment process in a billing context. The process is composed of 5 possible activities or *tasks*: set checkpoint (A), pay by check (B), pay by cash (C), print bill (D) and close payment (E). Figure 2(c) reflects a possible log generated by the information system managing this process. The system has recorded two different *traces*: 400 instances of the trace ABDEA, and 600 for

[1] The work presented in this paper is an extension of the work presented in [9, 10].
[2] http://www.promtools.org/prom6



# Instances	Log Traces
400	A B D E A
600	A C D E A

FIGURE 2. Three possible process models describing the same log

ACDEA, reflecting the real behavior of the process. There are two models, described in Figures 2(a) and 2(b), as potential descriptions of the process. Along this paper, we will use different modeling languages for describing a process: for the example at hand, we are using BPMN (Figures 2(a) and 2(b)) and Petri nets (Figure 2(d)). The question addressed in the precision dimension is: which model describes more precisely the behavior reflected in the log? When considering the log of Figure 2(c), the model of Figure 2(b) is more precise because model of Figure 2(a) includes a concurrency between D and E, an extra behavior not reflected in the log (the precision metric proposed in this paper would reflect 1 and 0.85 respectively). The extreme case is known as the *flower model* (modeled as a Petri net in Figure 2(d)): this model contains all the behavior in the log (i.e., the fitness dimension is perfect), but actually it allows any interleaving of the tasks, and thus does not provide any information about the workflow of the process.

This paper is organized as follows. In Section 2, we introduce the basic notation necessary for formally defining the problem in Section 3. In Section 4.1, we present a metric for precision. In Section 4.2, we present a technique to compute the confidence interval over the precision metric. In Section 5, we provide a method to detect the imprecisions and an approach to assess the severity of each imprecision. Finally, experimental results are provided in Section 6 while related work is reported in Section 7. Conclusions are presented in Section 8.

2. Preliminaries.

Event logs, or simply *logs*, contain executions of a system called *traces* [7]. These traces represent the ordering between different tasks, but may also contain additional information, such as the task originator or its timestamp. The work presented in this paper focuses on the control-flow perspective of the processes, and therefore, all this additional information is abstracted, simplifying the log as simply sequences of tasks. A formal definition of trace and log can be seen in Definition 2.1.

Definition 2.1 (Trace, Log)**.** *Let T be the set of tasks of a process, and let $\mathcal{P}(S)$ denote the powerset over S, i.e., the set of possible subsets of elements of S. A* trace *σ is defined as $\sigma \in T^*$. A* log *is a set of traces, i.e.,* $\mathsf{EL} \in \mathcal{P}(T^*)$.

Additional to the definitions above, we define the concept of *prefix* of a log, and the *occurrence* of a prefix.

Definition 2.2 (Prefixes, Occurrence of a Prefix)**.** *Given a log* EL*, define* $pre(\mathsf{EL})$ *as the set of prefixes of* EL*, i.e.,* $pre(\mathsf{EL}) = \{p \mid \sigma = px \in \mathsf{EL} \wedge p, x \in T^*\}$*. Note that the empty trace ϵ is a prefix of any log. Given a prefix* $p \in pre(\mathsf{EL})$*,* $p_{\#}^{\mathsf{EL}}$ *denotes the occurrences of that prefix in the log, i.e.,* $p_{\#}^{\mathsf{EL}} = |\{\sigma \text{ s.t. } \sigma = px \in \mathsf{EL}\}|$*. The subscript* EL *can be omitted whenever it is clear by the context.*

The behavior of logs and models defined in Section 3 will be mapped onto an extended version of *transition system*. The basic form of transition system is defined below.

Definition 2.3 (Transition system [11]). *A* transition system *(or* TS*) is a tuple* (S, T, A, s_{in})*, where* S *is a set of* states*,* T *is an alphabet of* actions*,* $A \subseteq S \times T \times S$ *is a set of* (labeled) transitions*, and* $s_{in} \in S$ *is the* initial state.

3. Problem Statement and Approach.

The approach presented in this work aims at comparing the behaviors of the model and the log in order to determine the precision of the pair. The procedure is divided into three phases: i) exploration of log behavior, ii) traversal of model behavior and iii) comparison of both behaviors, to detect the imprecisions. The three phases are explained in the following three sections. Finally, in Section 3.4, we address the problem of dealing with duplicate or invisible tasks and non fitting models.

3.1. Determining log's behavior.

The first step is to describe formally the behavior of the process reflected in the log. This description must have the same language (set of traces) of the log, but it should also include state information in order to compare it with the model behavior. For this purpose, the *prefix automaton* of a log is defined.

Definition 3.1 (Prefix Automaton). *Given a log* EL $\in \mathcal{P}(T^*)$*, the* prefix automaton *of* EL *is the transition system* TS $= (S, T, A, s_{in})$ *such that* $S = pre(\mathsf{EL})$*,* $A = \{(s, e, s') | s' = se\}$*, and* $s_{in} = \epsilon$*.*

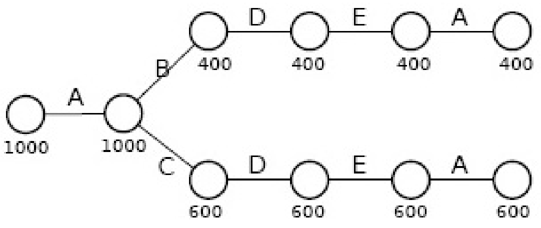


FIGURE 3. TS

Figure 3 shows the prefix automaton construction for the log in Figure 2(c). The number below each state s represents the occurrences of the prefix s in the log, i.e., $s_{\#}$. Note that all the states have an occurrence number greater than zero given that all them are log prefixes.

3.2. Log-based exploration of model behavior.

An important element in any process mining approach is the model that describes formally a process. These models can be developed by domain experts, or produced automatically by some process discovery technique [7]. In this work we present a general approach for precision checking, that can be instantiated on different modeling languages. We will consider $L(M)$ denote the set of traces that can be observed in model M, and avail(σ, M) to define the set of available tasks according to the model M after executing the sequence of tasks σ. In the last part of the section we present five modeling languages, *Petri nets and Reset nets*, *YAWL*, *EPC* and *BPMN*, and show how the approach is applied to them.

Once the log behavior has been determined, the second step is to compare it with the model behavior. However, the exhaustive exploration of model state space could be costly in terms of complexity, or even infinite. The approach presented in this work, unlike other conformance approaches such as [8], avoids incurring into this state explosion problem by performing a log-guided traversal of model behavior. The prefix automaton is used to guide the model state space exploration, extending it at each step of the traversal with information about behavior allowed by the model. This *extended prefix automaton*, which contains information of both log and model behavior, will be used later on to detect the imprecisions between them. Formally:

Definition 3.2 (Extended Prefix Automaton)**.** *Given the prefix automaton* $\mathsf{TS} = (S, T, A, s_{in})$ *of* EL *and the model* M*, we define the set of* extended states *and the set of* extended transitions *as:*

$$\begin{aligned} S^+ &= \{s | s = s't \wedge s' \in S \wedge t \in avail(s', M) \wedge s \notin S\} \\ A^+ &= \{(s', t, s) | s = s't \wedge s \in S^+\} \end{aligned}$$

The extended prefix automaton *is* $\widehat{\mathsf{TS}} = (\widehat{S}, T, \widehat{A}, s_{in})$ *with* $\widehat{S} = S \cup S^+$ *and* $\widehat{A} = A \cup A^+$*.*

Notice that the definition above makes the implicit assumption that, given a trace in the log, the set of available tasks after replaying that trace in the model always can be computed. However, there might be states of the log behavior that fall out of the model behavior, and therefore, the set of available tasks cannot be computed. These situations, known as *non fitting* are addressed in Section 3.4. In the rest of the section we will present intuitively the extension of the log prefix automaton for the most used process modeling languages: *Petri and Reset nets*, *YAWL*, *BPMN* and *EPC*.

3.2.1. *Petri nets and reset nets.* Petri nets [3] are widely used for modeling processes given its adequacy for reflecting concurrency, its intuitive graphical format, its formal semantics, and the vast number of approaches for Petri nets in Process Mining [12].

Informally, Petri nets are bipartite graphs, with two kinds of nodes, *transitions*, that represent the tasks of the process, and *places*, used to define the state of the system. The behavior of a process modeled using Petri nets is defined by the *tokens* and the *firing rule*. Tokens, represented by black dots, flow through places. A transition is *enabled* if there are tokens in all its input places. A distribution of tokens over places is called *marking*, and determines the state of the system. An enabled transition can *fire* (i.e., the task has been performed), and it results in the removal of the tokens in the preceding places and the creation of tokens in the succeeding places of the transition. Firing sequence of transitions determines the behavior represented by the model. The set of available tasks after firing a trace σ (avail(σ,M)) corresponds to the set of enabled transitions in the marking reached after firing σ in the Petri net. For more details, the reader can refer to [9].

Figure 4(a) shows a possible Petri net for the log of Figure 2(c), where tasks in the log are associated with transitions in the model. Apart from the simple 1-to-1 mapping between transitions and the tasks in the EL, other mappings are also possible, i.e., a several transitions can be associated with the same task (known as *duplicate tasks* and represented with the same label in all the transitions), or a transition can be associated with no task (known as *invisible tasks*, and represented as a black filled or unlabeled square). In this example, the model starts and ends with two transitions associated with the same log event A (i.e., it has duplicate tasks), and it contains an invisible task, used to model the possibility of skipping the execution of the task E. Figure 4(b) shows the extension of the log prefix automaton with the information on the Petri net, e.g., after the execution of ACD the model allows the execution of A directly (through firing the invisible transition), without executing E first, a behavior not reflected in the log.

In [13] the authors present a set of patterns that any real-world workflow modeling language should support. Although Petri nets have a great expressive power, they are not able to capture in an intuitive way all these patterns (e.g., the cancellation of an activity).

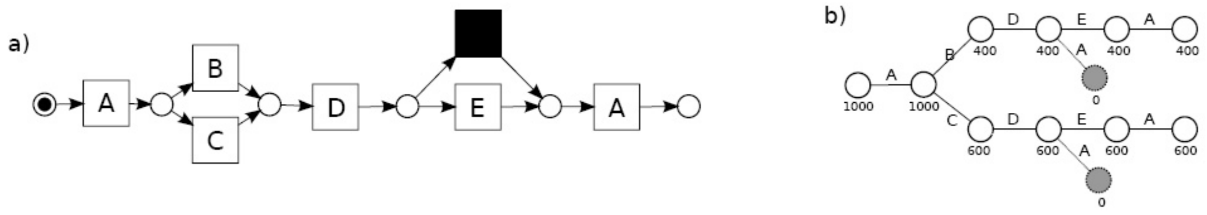
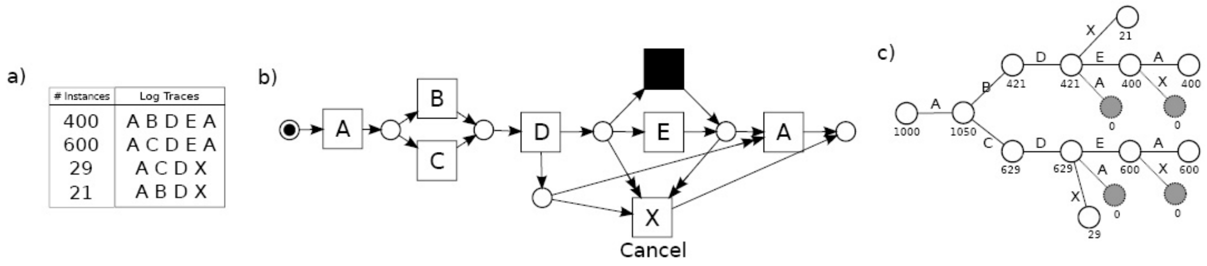


FIGURE 4. Petri net for the log of Figure 2(c) and its extended prefix automaton



# Instances	Log Traces
400	A B D E A
600	A C D E A
29	A C D X
21	A B D X

FIGURE 5. Example of system modeled with reset nets

Reset nets [14] are meant to cover this gap: they are Petri nets with the inclusion of a new element called reset arcs. A reset arc connects a place and a transition, but it does not restrict the enabling of the transition. Furthermore, when the transition to which a reset arc is connected fires, any place connected to the transition by the reset arc is emptied of all their tokens. Reset arcs are represented by two headed arrow ($\twoheadrightarrow$).

In order to illustrate the application of the approach presented in this paper for reset nets, we consider a simple modification of the example of Figure 4, shown in Figure 5. In this process we introduced an extra task *cancel* labeled as X. In the model, X reflects the possibility of canceling properly the procedure (removing all the tokens) before closing the process (E) or setting the checkpoint (A). In case that the check point is set (A) without canceling the procedure, the option of canceling the process becomes unavailable. However, the log only reflects the cancellation of the procedure before closing the process (E). Therefore, the cancellation after E becomes a behavior modeled but not reflected in the log, and the prefix automaton is extended accordingly.

3.2.2. *YAWL (yet another workflow language).* YAWL [4] was developed in order to capture easily some of the workflow patterns that Petri nets were not able to reflect in an easy way, e.g., cancellation, synchronization of active branches only, and multiple concurrently executing instances of the same task. YAWL has formal semantics based on transition systems, and also a complete support environment.[3]

YAWL models are mostly composed from the following elements: tasks, represented by squares, and conditions drawn by circles (as in Petri nets). However, in YAWL models tasks can be connected directly with other tasks (assuming an implicit condition between them). Furthermore, YAWL supports the join and split constructs shown in Figure 6(a). Informally, the semantics behind each construct is the following: AND-join and XOR-join receive the thread of control when *all* or *one* of the incoming branches are enabled, respectively. Similarly, AND-split and XOR-split pass the thread of control to *all* or *precisely one* of its outgoing branches. When OR-split is enabled, the thread of control is passed to one or more of the outgoing branches. Finally, OR-join receives the control when either each active incoming branch has been enabled or it is not possible that any

[3]More details on www.yawlfoundation.org.

branch that has not yet been enabled will be enabled at any time in the future.[4] YAWL allows not only to define atomic tasks, but also composite tasks (tasks that are YAWL models itself), multiple instances tasks (the same tasks with different parameters), and composite multiple instance tasks (a combination of the previous two). For the sake of clarity in this paper we only consider atomic tasks.

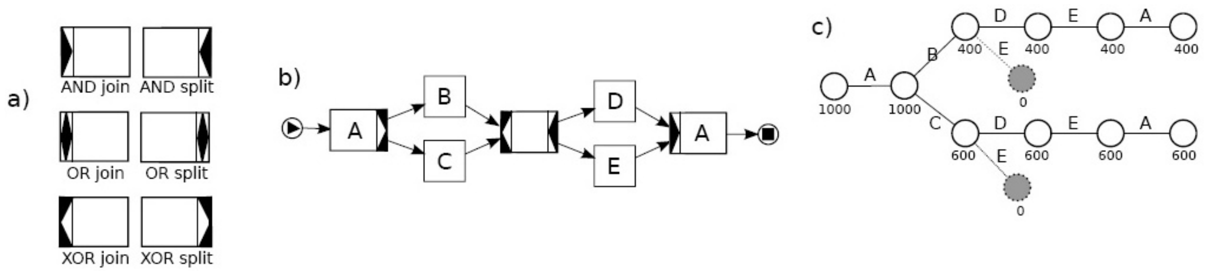


FIGURE 6. YAWL for the log of Figure 2(c) and its extended prefix automaton

Hence, a task is enabled in YAWL if it receives the thread of control. Given a sequence σ, we can define the available tasks (avail(σ,M)) as the tasks with the thread of control after replaying σ in the YAWL model M. For instance, Figure 6(b) defines a YAWL model for the log of Figure 2(c). The extended prefix automaton of the YAWL model based in that log is shown in Figure 6(c). For example, for the state AB, we reproduce the trace AB in the YAWL model, and the tasks that receive the thread of control after AB are D (present in the log) and E (not present in the log).

3.2.3. *BPMN (business process modeling notation).* BPMN [6] models are composed from three types of nodes[5]: events, activities and gateways.[6] *Events* (represented as circles) denotes something that happens (e.g., time, messages, ...), rather than *Activities* which are something that is done (represented as rounded-corner rectangles). Finally, the *gateways*, represented as diamond shapes, are used to route the control flow. We can classify the gateways in six types: *exclusive decision* and *exclusive merge* gateways (denoted by a "×"), *parallel fork* and *parallel join* gateways (denoted by "+") and finally, *inclusive decision* and *inclusive merge gateways* (denoted by "O"). Exclusive decision routes the control flow to exactly one of its outgoing branches, while exclusive merge receives the thread of control when one of the incoming branches is enabled. Parallel fork splits the thread of control among all its outgoing branches, and parallel join waits for all their incoming branches. Finally, inclusive decision and inclusive merge, instead of synchronizing all the branches (as parallel gates) or only one (as exclusive gates), synchronize *any* number of the branches. However, unlike Petri nets or YAWL, the semantic behind these gateways is underspecified in the standard specifications [15].

Figure 7(a) shows a BPMN model corresponding with the log shown in Figure 2(c). The first part of the diagram models the choice between task B and task C, while the second part reflects the concurrency between tasks D and E. The process is started and ended with two activities both associated with the log event A. Figure 7(b) shows the extended prefix automaton of the log and the BPMN model. Note that, the extension of this model and the YAWL model seen previously reflect the same extended prefix automaton.

[4]Or-join semantics in YAWL is a complex topic, as described in [4].

[5]BPMN 1.0 is considered.

[6]BPMN may contain other more advanced elements such as complex gateways, triggers and subprocesses which are not discussed in this paper. For more information, see www.bpmn.org.

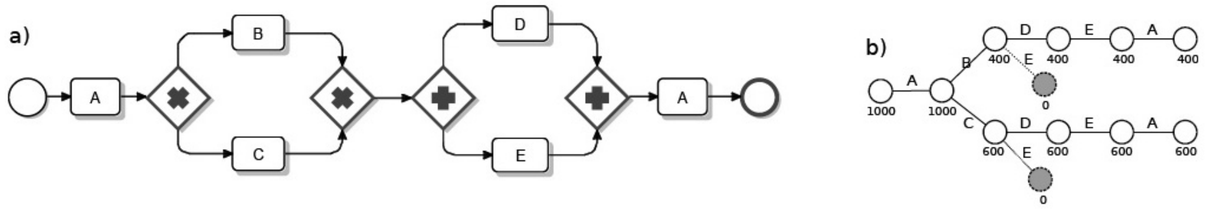


FIGURE 7. BPMN for the log of Figure 2(c) and its extended prefix automaton

3.2.4. *EPC (event-driven process chain).* EPC [5] uses a chain of elements to define the workflow of a process. An event-driven process chain may contain three types of elements: functions, events and logical connectors. *Functions* (rounded rectangles) are the tasks to execute, while *events* (hexagons) describe the situation before and/or after executing a function. Finally, logical connectors are used to route the flow of the process. There exist three types of connectors: AND (or $\wedge$), OR (or $\vee$) and XOR (or X), with a semantic similar to the one presented for BPMN. Like in BPMN notation, the underspecified and non-local semantics behind EPC is something one should be aware of when using this modeling language [16].

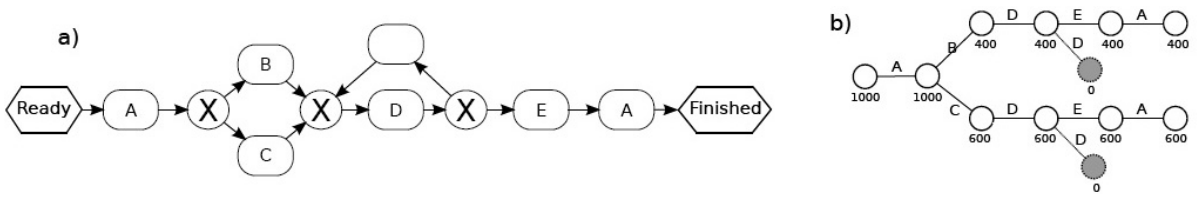
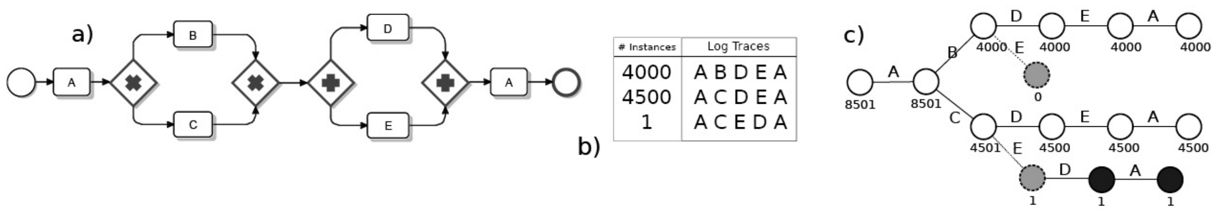


FIGURE 8. EPC for the log of Figure 2(c) and its extended prefix automaton

Figure 8(a) shows a possible EPC model for the process reflected in the log in Figure 2(c). The first part of the model reflects the choice between tasks B and C. The second part contains a loop that models the possibility of executing the tasks D several times, an extra behavior not reflected in the log, but which appears in the extended prefix automaton (cf, Figure 8(b)). Notice that, EPC events represent conditions but not actions, and therefore they are not considered tasks of the process. Note also that, although the model contains a loop (thought an invisible task) and the behavior modeled is infinite, our approach makes a log-guided exploration of the model behavior, bounding it, and limiting the complexity of the exploration.

3.3. **Detecting precision discrepancies.** Once the prefix automaton contains the information about log and model behaviors, it has all the necessary information to detect the *imprecisions* of the model in describing the log, i.e., the points where the model reflects more behavior than the one represented in the log. Intuitively, these points are the states in the automaton with an occurrence value of zero, denoting that they are allowed by the model but they do not appear in the log. As expected, these imprecisions (filled in gray) correspond with the new states introduced in the extension of the prefix automaton, denoting behavior not reflected in the log but modeled anyway. For instance, the imprecisions in the Figure 4(b) are produced by the possibility of skipping the task E through an invisible task. This extra behavior in the model was never present in the log.

However, in order to add an extra layer of robustness to the detection of imprecisions, we extend the definition of imprecision to include, not only the states with occurrence value zero, but also the ones below a certain threshold. With this threshold we avoid that a noisy trace will affect the computation of the precision value. The example in Figure



# Instances	Log Traces
4000	A B D E A
4500	A C D E A
1	A C E D A

FIGURE 9. Motivation for robust imprecision detection

9 illustrates intuitively the idea. In the example, a slightly different version of the log of Figure 2(c) is used, where a noisy trace ACEDA has been inserted, denoting an erroneous and infrequent behavior that should not be considered. In the simple approach presented above, where only states with occurrence value equal to zero are regarded as imprecisions, only one imprecision would be detected (ABE), due to the fact that the other imprecision (ACE) will be masked by the noisy trace. However, by considering a threshold related to the frequency of each location (cf. the remaining of this section), we will be able to consider the noisy trace as an imprecision.

In order to define formally the imprecision points, first we need to define the imprecisions (called *escaping states*) for a given state. Then, we use this definition to formalize the *escaping states* of the whole system, splitting the prefix automaton into three parts: *inner*, *escaping states* and *outer* part (denoted as white, gray and black circles respectively).

Definition 3.3 (Escaping States of a state). *Let* $\widehat{\mathsf{TS}} = (\widehat{S}, T, \widehat{A}, s_{in})$ *be the given extended prefix automaton for the log* EL *and the model* M, *and consider a threshold parameter* $\gamma \in [0,1]$. *Given a state* $s \in \widehat{S}$, *the set of* escaping states *at* s *is defined as:*

$$E_S^\gamma(s) = \left\{ s' | (s, e, s') \in \widehat{A} \wedge (\gamma \cdot s_\#) \geq s'_\# \right\}$$

In other words, the occurrence value of the state s' *(i.e.,* $s'_\#$*) must be inferior to the threshold defined for this point (i.e.,* $\gamma \cdot s_\#$*) in order to be an escaping state.*

Definition 3.4 (Escaping , Inner and Outer States of a system). *Let* $\widehat{\mathsf{TS}} = (\widehat{S}, T, \widehat{A}, s_{in})$ *be the given extended prefix automaton for the log* EL *and the model* M, *and consider a threshold parameter* $\gamma \in [0,1]$. *The set of escaping states of a system is defined as:*

$$E_S^\gamma = \left\{ s \mid \exists s_a \in \widehat{S} : s \in E_S^\gamma(s_a) \wedge \nexists s_b, s_c \in \widehat{S} : s = s_b x \wedge s_b \in E_S^\gamma(s_c) \right\}$$

The set E_S^γ *defines the* border *between the log and the model behavior. The states that fall out of this border are called* outer states *and are defined as:*

$$O_S^\gamma = \left\{ s \mid s \in \widehat{S} \wedge \exists s' : s = s'x \wedge s' \in E_S^\gamma \right\}$$

The states that fall in of this border are called inner states *and are defined as:*

$$I_S^\gamma = \left\{ s \in [\widehat{S} \setminus (E_S^\gamma \cup O_S^\gamma)] \right\}$$

Figure 9 shows the escaping states (gray), the inner (white) and the outer (black) part of the example system, for $\gamma = 0.02$. Notice that, given the definitions above, there may exist states belonging to the escaping states set of a given state that do not belong to the escaping states set of the whole system, i.e., they fall into the outer part.

3.4. **Dealing with non-fitting models, duplicate and invisible tasks.** Although the approach presented has a general nature, there are some special cases that should be analyzed in detail. The first situation are the so called *non fitting traces.* In the second step of the technique presented above, each state of the prefix automaton is extended with the information about the tasks available according to the model representation. This process was assuming implicitly that it was always possible to determine the set of available tasks for any prefix in the log, replaying that prefix in the model. In other words, the behavior of the log is assumed to be a subset of the behavior modeled in the model. However, in the real-case scenarios, it is common to find traces that, at some point, fall out of the model behavior, i.e., traces that do not fit with the model. The analysis and measuring of the conformance issues produced by this kind traces fall directly under the scope of another of the conformance dimensions: fitness. The fitness dimension and the approaches proposed to measure it (e.g., [8, 17, 18]) should analyze that part of the conformance independently from the precision dimension. However, the prefixes of non fitting traces that fall within the model behavior can be considered for the precision computation: given a non-fitting trace σ for a model M (i.e., $\sigma \notin L(M)$), we can characterize the set of *fitting states* of σ as $\{\sigma' \mid \sigma = \sigma' x \ \wedge \ \sigma' \in L(M)\}$. In the presence of non-fitting traces of an EL, the approach can still compute precision by focusing on the fitting states of every trace in EL.

The second scenario to consider is the inclusion of duplicate and invisible tasks. The approach presented above is able to deal with duplicate tasks. Figures in Section 3.2 show examples of these situations, i.e., the task A appears twice, at the beginning and at the end of the models in that section. The approach presented is also able to deal with invisible tasks. Invisible tasks are usually represented as black-filled boxes (e.g., in Petri nets) or simply as tasks without label. Examples of invisible tasks and how they are handled can be seen in Figures 4, 6 and 8. However, all that examples satisfy a crucial property: they are *deterministic.* In other words, the inclusion of duplicate tasks could produce cases of indeterminism, i.e., it is not possible to determine a unique replay of a trace in the model. Figure 10(a) shows an example of these situations (in that case, using Petri nets as modeling language). In the example, if we try to replay the sequence AB, given the information reflected in the log, we are not able to determine which of the two tasks in the model (B on top or B at bottom) is the one that should be fired, because both have the same footprint in the log. A similar situation could be produced by the introduction of invisible tasks, as it is shown in Figure 10(b).

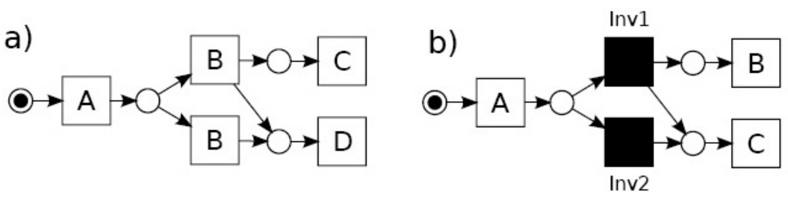


FIGURE 10. Indeterminism produced by duplicate and invisible tasks

Choosing one or another invisible/duplicate task in the model may produce differences in the behavior, and therefore the computation of the precision would be strongly related with the choices made. In such cases, the application of heuristics and best effort strategies becomes necessary, e.g., *n-lookahead* (i.e., consider the next n tasks in the trace to make a decision), *lazy invisible* (i.e., invisible tasks only considered if task is already available), or *shortest sequence of invisible* (i.e., consider the shortest invisible tasks necessary to avail the task) [8]. The latter two assume that the simplest reproduction of the trace must be the correct one (Occam's razor). Finally, we can apply more complex strategies based in the cost-replayed of traces in order to determine the sequence of tasks to be considered in each case [19]. The inclusion of invisible tasks in a deterministic scenario could also be problematic: in the presence of invisible tasks, to determine the set of available tasks may require the exploration of the state space, given that for invisible tasks there is not reflex in the log to guide that traversal. For the case of Petri nets, a finite exploration of this state space can be performed through the notion of *Invisible Coverability Graph* [9].

4. **Evaluation of the Precision.**

In this section we propose a new metric for measuring precision based on the escaping states defined in the previous section. Furthermore, a confidence value over the metric value is presented in the second part of the section.

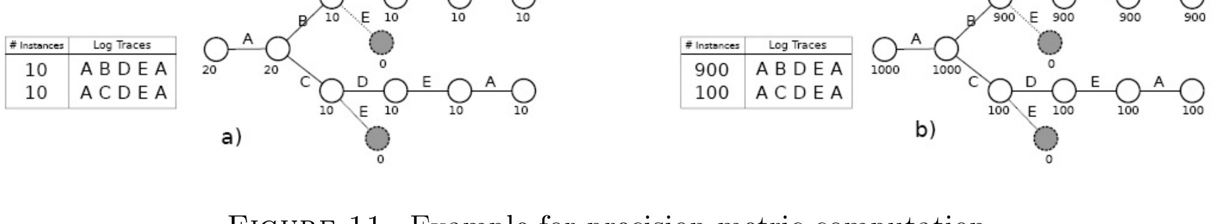
4.1. **Precision metric.**

The metric presented in this section uses the extended prefix automaton to compare model and log behaviors. For evaluating the metric we count the imprecisions, weigh them according to their frequency, and compare them with the behavior allowed in every state. This way of estimating precision is strongly linked with the idea of assessing the effort needed to achieve a model 100% precise.

Definition 4.1 (ETC Precision metric)**.** *Let* EL *be a log, a model* M*, the extended prefix automaton* $\widehat{\mathsf{TS}} = (\widehat{S}, T, \widehat{A}, s_{in})$ *with inner and escaping states* (I_S^γ, E_S^γ) *from Definition 3.4 on a threshold parameter* $\gamma \in [0,1]$*. The metric is defined as follows:*

$$etc_P(\gamma) = 1 - \frac{\sum\limits_{s \in I_S^\gamma} (|E_S^\gamma(s)| \cdot s_\#)}{\sum\limits_{s \in I_S^\gamma} (|avail(s,M)| \cdot s_\#)} \tag{1}$$

The metric evaluates the amount of *overapproximation* in each state from I_S^γ, by dividing the set of escaping states by the set of allowed tasks. Note that $|E_S^\gamma(s)| \leq |\text{avail}(s,M)|$, and therefore $0 \leq \text{etc}_P \leq 1$.[7] To illustrate the metric we consider the payment process modeled in Figure 2(a) and two possible logs shown in Figure 11. The corresponding extended prefix automata determining the escaping states (considering a $\gamma = 0.02$) is shown in the same figure. For each case, the metric value would be $\text{etc}_P(0.02) = 1 - \frac{20}{140} = 0.86$ and $\text{etc}_P(0.02) = 1 - \frac{1000}{7000} = 0.86$, respectively. These results reflex that both pairs model-log have the same high precision of 0.86, as it was expected, regardless of the size of the log. However, note that imprecisions with different weight would affect differently the metric. Both examples do not achieve the perfect precision, given the concurrent behavior between activities D and E (such concurrency is not observed in the log). On the other hand, if we consider the same pair of logs with the model of Figure 2(b), the resulting extended prefix automaton will contain no escaping state, and the metric will report a 100% of precision (as it would be expected).



# Instances	Log Traces
10	A B D E A
10	A C D E A

# Instances	Log Traces
900	A B D E A
100	A C D E A

FIGURE 11. Example for precision metric computation

[7]Only states included in model behavior are considered (cf. Section 3.4).

Note that in these examples we are considering a γ value of 0.02. Actually, changing γ slightly has no effect in the metric for these particular examples. However, there are situations where it could be interesting to consider a greater γ, e.g., computing only the precision of the frequent parts of the process. For example, in Figure 11(b), considering a γ of 0.15, the infrequent part of the trace ACDEA is cut, considering only the most frequent trace ABDEA. In that case, the state AC is considered as an escaping state, and the rest of the succesors states are considered as outer states. The precision metric changes to 0.71.

4.2. **Confidence over the precision metric.** Together with the precision metric presented in the previous section, we provide an estimation of the confidence over this metric, i.e., an estimation of the stability of the metric value for a certain future window to consider. In this work, we propose a *confidence interval*, i.e., and upper and a lower value over the metric value. The amplitude of the interval determines the confidence over the metric (i.e., the narrower the interval is, the more confidence one has). The confidence interval described in this work is strongly linked with the future to consider, i.e., the confidence that the metric vary in a short period of time (and therefore with only few new executions of the process) should not be the same as considering a long period of time where much more new incoming behavior is considered. This future is specified using a parameter k, which defines the number of new extra traces to consider in order to analyze the possible variations of the metric value. By estimating an upper and a lower metric value, we are providing a confidence interval over the metric itself.

4.2.1. *Upper confidence value.* In order to estimate an upper confidence value, we must try to maximize the metric value using the k new traces that determine the future assuming always a best case scenario, i.e., minimizing the number of imprecisions. In the best case scenario we can assume that each of these k traces is able to reach an escaping state, i.e., each of them are traces starting in the initial state of the prefix automaton and ending exactly in an escaping state. If the escaping state receives enough traces to overpass the threshold (γ, cf. Section 3.3), it will not be an imprecision anymore, and therefore, the metric value will be increased. Notice that, given the way the precision metric is computed (see Section 4.1), the increase achieved covering different imprecisions may be different, i.e., imprecisions concerning frequents parts have a greater weight in the metric than other imprecisions less frequent, and therefore, covering them produces a greater increase.

In summary, the problem of computing an upper confidence value can be reduced to an optimization problem: maximize the total increase (gain) of covering imprecisions, given k new traces and a set of imprecisions, each one with a function that determines the cost and gain of covering it. The cost of covering an escaping state is determined by the number of traces needed to overpass the threshold. The gain of covering a escaping state is estimated as the increase it will produce in the metric. Formally:

Definition 4.2 (Cost and Gain of Covering an imprecision)**.** *Let $s' \in \widehat{S}$ be an escaping state such that $(s, e, s') \in \widehat{A}$, and let γ be the parameter used to define the escaping states. The* cost *of covering s', denoted as $C(s') = l$ with $l \in \mathbb{N}$, is the minimum l that satisfies $(s_{\#} + l) \cdot \gamma < (s'_{\#} + l)$. The* gain *of s' is defined as $G(s') = s_{\#}$, i.e., the gain of reducing in one the number of escaping states of the parent state s.*

By inspecting the fraction part of Formula (1), one can see why the gain of covering the escaping state s' is $s_{\#}$: if in state s one escaping state is removed, then the new escaping estates in s are $|E^{\gamma}_{S}(s)|-1$. Since this number is multiplied by $s_{\#}$ in the numerator part of the fraction, the numerator will be reduced exactly in $|E^{\gamma}_{S}(s)| \cdot s_{\#} - (|E^{\gamma}_{S}(s)| - 1) \cdot s_{\#} = s_{\#}$.

Given the cost and gain functions, and the maximization to achieve, the problem can be reduced to the *Knapsack* problem [20]. Here we propose the formulation of the problem as a *Binary Integer Programming (BIP)* problem. This problem can be solved using linear programming techniques [21]. The formulation is described below:

BIP model to solve:

1. *Variables:* denoting if the escaping state i is covered or not.
$$\forall i \in E_S^\gamma : X_i \in \{0,1\} \tag{2}$$
2. *Constraints:* the total cost cannot exceed the number of new traces seen.
$$\sum_{i \in E_S^\gamma} C(i) \cdot X_i \leq k \tag{3}$$
3. *Cost function:* maximize the gain.
$$\max \sum_{i \in E_S^\gamma} G(i) \cdot X_i \tag{4}$$

In the simple formulation of the problem, the number of variables is $|E_S^\gamma|$. However, characteristics of the problem and properties over the domain can be exploited to reduce significantly this number. For example, being $\mathcal{C}$ the set of possible costs for a given model, for each possible $l \in \mathcal{C}$, at most $\lfloor k/l \rfloor$ can be used in order to satisfy (3). Hence, the real upper bound to the number of variables needed is $\sum\limits_{l \in \mathcal{C}} \lfloor k/l \rfloor$.

With the result of the maximization of the possible total gain obtained using at most k traces, we can define the upper confidence interval value as follows.

Definition 4.3 (Upper Confidence Value). *Let N and D be the numerator and denominator of the metric etc_P as defined in (1), i.e., $etc_P(\gamma) = 1 - (N \backslash D)$. Let $G_{\max}$ be the result obtained using the optimization problem modeled above. The* upper confidence value *is defined as follows:*
$$U(k) = 1 - \frac{N - G_{\max}}{D} \tag{5}$$

To illustrate the confidence value we use again the example of Figure 11. In the example 11(a) there are 2 imprecisions. Given $\gamma = 0.02$, the cost of covering each imprecision is the same, i.e., 1 trace, and the gain is also the same, i.e., 10 in each case. Taking $k = 1$, only 1 trace could be covered, and then the upper bound for this k would be $U(1) = 1 - \frac{20-10}{140} = 0.93$. Considering a greater k (e.g., $k = 2$), both imprecisions can be covered, achieving an upper confidence value of 1. Note that, even considering a near future the metric could increase substantially, denoting a low confidence over the metric. On the other hand, in example 11(b) there are also two imprecisions, but the cost and the gain of covering them are not the same. Given $\gamma = 0.02$, the cost of covering them is 19 (upper) and 3 (lower), and the gain is 900 and 100, respectively. In this case, considering $k = 1$ no imprecision can be covered. With $k = 3$ only the lower imprecision can be covered, and the resulting upper bound would be $U(3) = 1 - \frac{1000-100}{7000} = 0.87$ (only a minimum variation over the metric). With a greater k of 20, only one of the two imprecisions can be covered. In that case, and taking into account that we are considering the best case scenario, we maximize the gain (i.e., covering the imprecision with gain of 900), and hence the upper bound achieved is $U(20) = 1 - \frac{1000-900}{7000} = 0.99$. Note that, covering only the other imprecision will achieve an upper bound considerably inferior, i.e., 0.87. For values of k greater than 22, both imprecisions can be covered, achieving precision 1.0. Notice than, in this second example, more executions of the process have been reflected in the log, and then the confidence over the metric computed is higher (i.e., the upper bound is closer to the metric value).

4.2.2. *Lower confidence value.* The approach behind the lower bound assumes a situation determined by the k new traces that lower as much as possible the precision value (worst case scenario). Hence, in contrast to the upper bound computation where we maximize the imprecisions to cover, the appearance of new imprecisions in the system is considered instead. By estimating the new imprecisions that may appear caused by the k traces, we will be able to determine the confidence over the metric decrease in the future.

The type and number of new imprecisions arising from observing k new traces can vary significantly. In this work we propose an approach based on the size of the traces seen in the log, and the most generic and worst case scenario possible according to the model. If m is the average size of the traces in the log, we consider k traces of size m, representing new behavior. Therefore, this k new traces produce $m \cdot k$ new states in the prefix automaton. At the same time, each one of this new states contains θ escaping states. This parameter θ is defined by the user depending on the scenario to be considered. For instance, this parameter could be extracted statistically from the log as final representation of the process (e.g., the average of escaping states per state, per trace). Alternatively, given that we are considering always the worst case scenario, we may consider that each of these new states contains the maximum number of possible escaping states, i.e., $|T| - 1$ where T are the set of tasks. In other words, all the tasks are available but only one is followed by the trace. Given these considerations, the lower confidence value is defined as follows.

Definition 4.4 (Lower Confidence Value)**.** *Let N and D be the numerator and denominator of the metric etc_P as defined in (1), i.e., $etc_P(\gamma) = 1 - (N \backslash D)$. The* lower confidence value *is*

$$L(k, \theta) = 1 - \frac{N + (m \cdot k \cdot \theta)}{D + (m \cdot k \cdot |T|)} \tag{6}$$

Following with the example of Figure 11 and considering θ as $|T| - 1$, the average size of the traces in both logs (i.e., m) is 5, and the number of tasks of the process (i.e., T) is 5. Considering $\gamma = 0.02$, $k = 1$, the lower bound of log 11(a) would be computed as $L(1, 4) = 1 - \frac{20+(5\cdot1\cdot4)}{140+(5\cdot1\cdot5)} = 0.76$. On the other hand, considering the log 11(b) with the same parameters, the lower bound would be $L(1, 4) = 1 - \frac{1000+(5\cdot1\cdot4)}{7000+(5\cdot1\cdot5)} = 0.85$, i.e., really close to the metric value denoting the low confidence of changing in the near future. Notice again that in the second example, more executions of the process have been reflected in the log, and therefore, the confidence over the metric computed is higher. This is even clearer if we take both upper and lower confidence value. For instance, with $k = 1$ and $\gamma = 0.02$, the metric and the interval of example 11(a) are 0.86 and $0.76 - 0.93$ (the size of the interval is 0.17). For example 11(b), they are 0.86 and $0.85 - 0.86$ (the amplitude of the interval is 0.01, considerably smaller than example 11(a)).

5. **Locating and Assessing Imprecisions.**

As important as evaluating the precision of a system in a global way is locating where the precision problems are and assessing their importance. This is the goal of this section: in the first part we present the *minimal disconformant traces* used to locate the imprecisions, while in the second part we present an approach to estimate the severity of these imprecisions.

5.1. **Locating the imprecisions.** In this section we provide *paths* to imprecisions, i.e., the path followed by the system to reach an escaping state. We call these paths *Minimal Disconformant Traces (*MDT*)*. MDT main property is their minimality, i.e., they are the shortest trace in the model such that it deviates from the log behavior [9]. Therefore, MDT represents a succint mechanism to report imprecisions, identifying the situations to tackle in order to reach a more precise model.

Definition 5.1 (Minimal Disconformant Traces (MDT)). *Given T the set of tasks of the process, a log* EL*, a model M, define the Minimal Disconformant Traces of* EL *and M as:*

$$\mathsf{MDT} = \{\sigma' \in T^* \mid \sigma' = \sigma t \wedge \sigma' \in L(M) \wedge \sigma \in pre(\mathsf{EL}) \wedge \sigma' \notin pre(\mathsf{EL})\}$$

By the construction of the extended prefix automaton and the definition of MDT above, there is only one MDT for every imprecision, corresponding to the sequence associated with the escaping state. For instance, Figure 12(a) shows the two existing MDT for the process in the examples of Figure 11, one for each imprecision. Furthermore, the set of MDT could be considered as an event log, i.e., it is indeed a set of sequence of tasks.

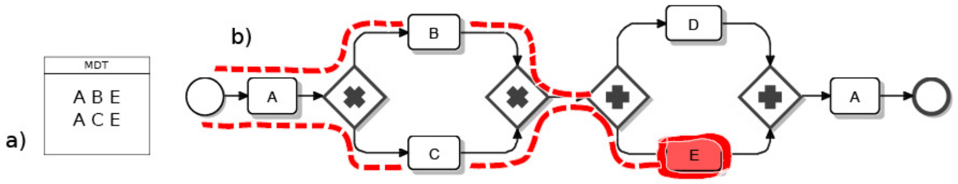


FIGURE 12. Example of minimal disconformant traces (MDT)

5.2. **Severity of an imprecision.** Associating a severity value to each imprecision will assess the urgency for fixing it. Moreover, the imprecisions can be sorted according to their severity. Below we provide a set of factors to estimate the severity of an imprecision.

Assessing the severity of an imprecision is a complex and subjective operation, which may change depending on the scenario and the needs of the analyst of the system. For that reason, in this work we present a multi-factored severity approach, where each of the four factors may be weighed according to the importance each scenario requires. The four factors considered are *frequency*, *alternation*, *stability* and *criticality*.

Definition 5.2 (Severity of an Imprecision). *Let E_S^{γ} be the set of escaping states denoting the imprecisions of a system. Given $s \in E_S^{\gamma}$, the* severity *of the imprecision at s is*

$$sev(s) = f(F_{ft}, A_{ft}, S_{ft}, C_{ft}) \tag{7}$$

where F_{ft}, A_{ft}, S_{ft}, C_{ft} correspond to the frequency*,* alternation*,* stability *and* criticality *factors of the imprecision at s, and f is a user-defined function that weighs these four factors.*

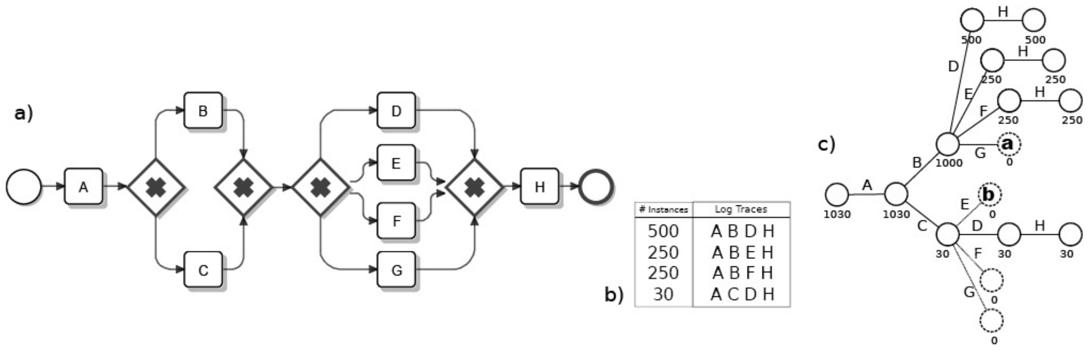
In the rest of the section, the four factors are described. In the definitions below, the following context is assumed: a model M, a log EL, the extended prefix automaton of EL on M $\widehat{\mathsf{TS}} = (\widehat{S}, T, \widehat{A}, s_{in})$, and $(s', e, s) \in \widehat{A}$ where $s \in E_S^{\gamma}$ is the imprecision to be assessed.

5.2.1. *Frequency.* The first factor assesses the severity of an imprecision according to its frequency. An imprecision reached frequently should have more importance than another located in infrequent parts that barely have any repercussion in the process execution. Therefore, we define the *frequency factor* as follows:

Definition 5.3 (Frequency factor). *Let* $\max_{\#} = \max(x_{\#} \mid x \in \widehat{S})$ *be the maximum occurrence value of all the states in* $\widehat{S}$. *The* frequency factor *of the imprecision at s is defined as follows:*

$$F_{ft}(s) = \frac{s'_{\#}}{\max_{\#}} \tag{8}$$

To illustrate the frequency factors (and the rest of factors) we consider the process of Figure 13. The process is composed by a model (13(a)) and a log (13(b)). The extended prefix automaton (considering a $\gamma = 0.01$) is shown in 13(c). We consider two of the four imprecisions of the system: imprecision a is reached after the sequence ABG, while imprecision b is reached after ACE. Imprecision a has a high frequency factor of $\frac{1000}{1030} = 0.97$, (where 1030 is the maximum occurrence value in the whole prefix automaton). On the other hand, imprecision b has a extremely low frequency factor of $\frac{30}{1030} = 0.03$.



# Instances	Log Traces
500	A B D H
250	A B E H
250	A B F H
30	A C D H

FIGURE 13. Example for determining the severity of the imprecisions

5.2.2. *Alternation.* The second factor of the severity assessment addresses the probability of choosing an incorrect option. In case of the precision dimension, this incorrect choice refers to the possibility of, given a state, choosing a task allowed by the model but not reflected in the log, i.e., an imprecision. There are situations where choosing badly is really unlikely because almost all the possible paths reflect correct process behavior. However, there are cases where the ratio between the tasks that should be enabled (i.e., lead to an escaping state) and the ones that are really allowed by the model is quite high. These latter situations are the ones that should be tackled first (i.e., have high severity) given that they may produce problematic situations in the process. The *alternation factor* is the one that assets that issue, and can be defined as follows:

Definition 5.4 (Alternation factor). *Let* $P_E(x)$ *be the probability of choosing an escaping state being in the state* x. *The* alternation factor *of the imprecision at s is defined as* $A_{ft}(s) = P_E(s')$, *being* s' *the predecessor of s. The distribution of* $P_E(x)$ *may be different depending on the assumptions taken. If no assumption is considered, a uniform distribution, where each possible path has the same probability, must be applied. Therefore, the alternation factor can be estimated as:*

$$A_{ft}(s) = \frac{|E_S^{\gamma}(s')|}{|avail(s', M)|} \tag{9}$$

Following with the example of Figure 13, the alternation factors of both a and b imprecisions are completely different. For the imprecision a the factor value is $\frac{1}{4} = 0.25$, denoting a low probability of choosing an erroneous path in that location. On the other hand, imprecision b has a high alternation factor of $\frac{3}{4} = 0.75$, reflecting the high probability of deviating from the behavior observed in the log.

5.2.3. *Stability.* The third factor considers the probability of an imprecision to become a non-imprecision in the future. This concept is strongly linked with the robustness layer presented when defining the escaping states (cf. Section 3.3). In order to measure the stability or equilibrium of the imprecision, we apply a little perturbation to the system and the results are analyzed.[8] In our setting, the perturbation consists on considering a small number of extra traces (denoted by z) that reach the imprecision. For stable imprecisions, these z extra traces will not produce any effect, i.e., the imprecision will still be an imprecision. These are the cases that should be tackled first (i.e., high severity degree) because the probability of changing in the future is low. However, in unstable situations, these z extra traces may produce a sufficient change in the occurrence value of a predecessor state making it to overpass the threshold defined (γ, cf. Section 3.3), and thus become non-imprecisions. Repairing these unstable imprecisions may be a waste of resources given the high probability that they disappear in the future represented by the window of z traces. The intensity of the perturbation (i.e., the number of extra traces z considered) may be proportional to the *size* of the case (in our setting, the total number of traces seen in that location), i.e., the perturbation in frequent parts of the process must be greater than the perturbation made in other less frequent parts.

Definition 5.5 (Stability factor). *Let z be a percentage of the occurrence value of the location, i.e., $z = \lceil s'_{\#} \cdot \tau \rceil$ where $\tau \in [0, 1]$ is the parameter indicating the percentage of new traces to be considered in order to determine the stability of an imprecision. Moreover, let $l \in \mathbb{N}$ be the smallest number such that the equation $(s'_{\#} + z) \cdot \gamma < (s_{\#} + l)$ is satisfied, i.e., l defines the minimum number of traces the state s must receive in order to change from escaping to non-escaping state after considering z new traces in that point. The* stability factor *of the imprecision at s is the probability of s still being an escaping state after considering z new traces, i.e.,*

$$S_{ft}(s, \tau) = P^z_s(< l) = \sum_{i=0}^{l-1} P^z_s(= i) \tag{10}$$

where $P^z_s(< x)$ and $P^z_s(= x)$ represent the probability that the state s receives less than x (or exactly x) of the new z traces considered in this point.

For instance, for imprecision a in the example of Figure 13, and considering $\gamma = 0.01$ and $\tau = 0.06$, z would be $\lceil 1000 \cdot 0.06 \rceil = 60$ and l would be $\lceil ((1000 + 60) \cdot 0.01) - 0 \rceil = 11$. On the other hand, the z considered for imprecision b would be $\lceil 30 \cdot 0.06 \rceil = 2$ and l would be $\lceil ((30 + 2) \cdot 0.01) - 0 \rceil = 1$.

Let p_s define the probability that a new trace in s' follows the state s, and let $1 - p_s$ be the probability that the trace follows one of the other successor states of s'. According to the *Binomial distribution* [23], the stability factor can be expressed as:

$$S_{ft}(s, \tau) = \sum_{i=0}^{l-1} \binom{z}{i} (p_s)^i (1 - p_s)^{z-i} \tag{11}$$

Formula (11) can be understood as follows: in order for s to still be an escaping state, i successes $(p_s)^i$ and $z-i$ failures $(1 - p_s)^{z-i}$ are needed. However, the i successes can occur anywhere among the z traces, and there are $\binom{z}{i}$ different ways of distributing i successes in a sequence of z traces. The probability p_s may depend on the assumptions taken. Again, if no knowledge regarding the distribution of the log is assumed, a uniform distribution

[8]The idea of applying perturbations in order to estimate some properties has been used successfully in other fields, such as the measurement of community robustness [22].

is taken. Therefore, if c is the number of successor states of s', the probability of each successor state is $1/c$, and Formula (11) can be rewritten as:

$$S_{\mathit{ft}}(s, \tau) = \sum_{i=0}^{l-1} \binom{z}{i} \left(\frac{1}{c}\right)^{i} \left(1 - \frac{1}{c}\right)^{z-i} \tag{12}$$

Following with the running example of Figure 13, the stability factor of imprecision a would be 0.04, while value for the imprecision b is 0, denoting that it is really unstable.

5.2.4. *Criticality.* The last severity factor addresses the criticality of the tasks involved in a imprecision. Informally, this factor will measure the importance given to the tasks that deviate from the log: it is not the same to execute an unnecessary check of a field in an online form, than making twice the payment of a 3 million dollar grant. Thus, each task must be assessed with a criticality value strongly linked with the cost (economic, legal, temporal, ...) of executing the task in an inappropriate moment. This is a subjective factor, strongly related with the process analyzed, and requires a domain expert to provide the tasks criticality values. This factor does not appear in preliminary versions of severity computation presented in [10], and was inspired by the work in [19].

Definition 5.6 (Criticality factor)**.** *Let the function* $crit : T \to [0..1]$ *define the criticality for each task. The critical factor at imprecision* s *is* $C_{\mathit{ft}}(s) = crit(e)$*.*

In the example of Figure 13 we set a criticality value of 0.5 to all the tasks in the process, except D, G and E. D and E would correspond to complex checking actions (e.g., human revision of a 80 pages grand application form) and therefore their criticality value is set in 0.9. On the other hand, G would correspond to a simple checking (e.g., a milliseconds query of non-sensitive information in a database), and therefore it is assessed with a low value of 0.2. In such possible scenario, the criticality factor for imprecision a would be 0.9, while the imprecision b would be assessed as 0.2.

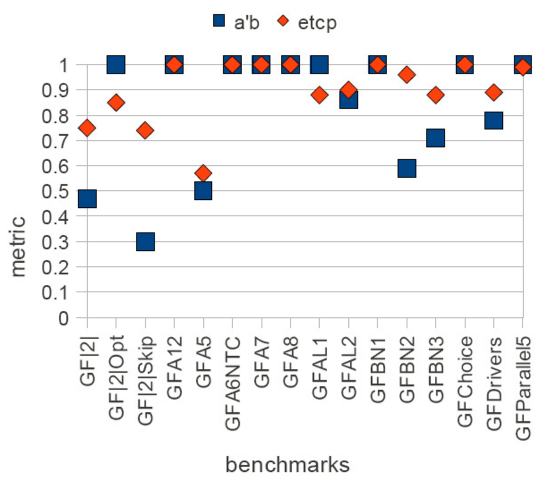
6. **Experiments.**

We start by comparing the approach of this paper with the metric $a'b$ presented in [8], one of the most popular techniques for precision checking. Figure 14(a) shows a chart illustrating the comparison. In order to confront both metrics, a γ value of 0 has been used in our tool. The benchmarks used are small logs, publicly available.[9] The models reported are Petri nets obtained from the log using the *ILPMiner* [24], a process discovery tool. The chart shows that, in the cases where the models are precise, both metrics report the highest value. In the other cases, the values of both metrics may be substantially different, due to the different emphasis both metrics have: while $a'b$ penalizes precision by observing the tasks relations, metric etc_P will penalize the situations where the model deviates from the log (a thorough comparison can be found in [9]).

Table 14(b) compares the precision metric of different modeling languages. The processes are based on *Describing planning processes in the production by using ARIS and SAP* and *New Technology Introduction Process (NTIP) for Solution Architects* ARIS Community reference models, and the example of Section 3. Each one of the three benchmarks comprises an EPC, YAWL, and BPMN model (created by hand), and a Petri net model (generated from the log using ILPMiner ProM plugin). The table illustrates how inter-model precision comparison can be done when several modeling languages are used.

Table 14(c) contains the experiments using larger benchmarks, from the same public available repository. These benchmarks cannot be handled by the current implementation of the $a'b$ metric. Our experimental setting is based on variations of the *a32f0n00_5* and *t32f0n00_5* examples: the experiments focus on illustrating how the growth of a

[9]www.promtools.org/prom5



(a)

Bench	**PN**	**YAWL**	**BPMN**	**EPC**
payment	.857	.857	.857	.857
new_tech	.775	.911	.861	.939
planning	.824	.810	.980	.887

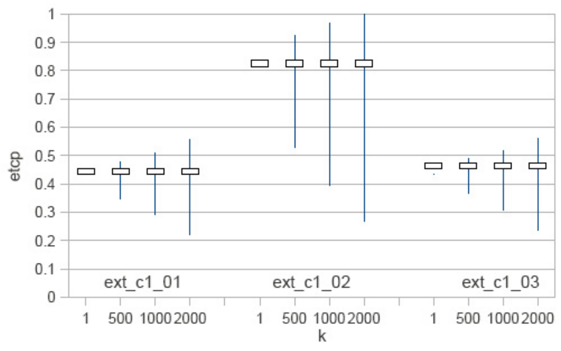
(b)

Bench		**|Log|**	γ	**k**	**etc$_P$**	**Confidence**		**time(s)**
	p20	180			.543	.246 - .553	(.307)	1 / 3 / 5
	p40	360			.564	.345 - .570	(.225)	1 / 5 / 6
	p60	540			.576	.403 - .582	(.179)	1 / 7 / 11
	p80	720			.583	.441 - .587	(.146)	1 / 12 / 17
a32	p100	900	.05	20	.592	.470 - .595	(.125)	1 / 15 / 24
	p150	1350			.591	.504 - .595	(.091)	2 / 16 / 23
	p200	1800			.591	.523 - .595	(.072)	2 / 17 / 23
	p250	2250			.590	.534 - .594	(.060)	2 / 16 / 24
	p300	2700			.591	.544 - .594	(.050)	2 / 16 / 24
	p20	360			.385	.250 - .387	(.137)	2 / 67 / 121
	p40	720			.391	.305 - .392	(.087)	4 / 180 / 229
	p60	1080			.392	.330 - .393	(.063)	5 / 295 / 339
	p80	1440			.393	.345 - .394	(.049)	6 / 336 / 496
t32	p100	1800	.05	20	.393	.353 - .394	(.041)	6 / 390 / 550
	p150	2700			.393	.365 - .393	(.028)	6 / 411 / 562
	p200	3600			.393	.371 - .393	(.022)	7 / 429 / 572
	p250	4500			.393	.376 - .393	(.017)	9 / 440 / 579
	p300	5400			.393	.379 - .393	(.014)	9 / 443 / 581

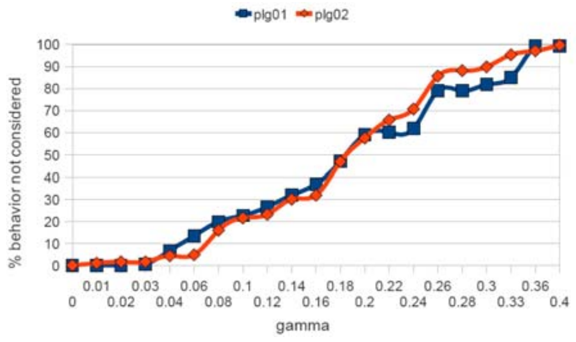
(c)

FIGURE 14. Experimental results (1)

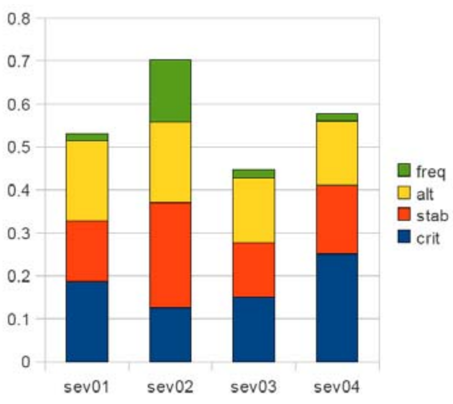
log influences the metric and its confidence, given a particular selection of the stability and confidence parameters presented in this paper. The column with *pX* reports the percentage of the log considered in each case, i.e., *p100* represents the original *a32f0n00_5* log, while logs *pX* with $X < 100$ correspond to slices of the original log, e.g., *p20* contains the first 20% of the original log traces. Logs *pX* with $X > 100$ are obtained by choosing with uniform distribution among the existing traces in the log the extra traces needed to achieve the desired size. As in the previous experiment, the models used are the ones obtained from discovering a Petri net through the ILPMiner ProM plugin. The wide spectrum of the set of benchmarks presented makes it possible to illustrate the evolution of the approach presented in this paper and can be considered as real situations in an information system where trace sets are evaluated on a regular basis, e.g., monthly. A



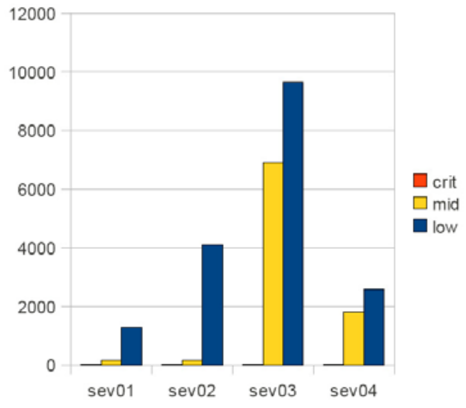
(a)



(b)



(c)



(d)

FIGURE 15. Experimental results (2)

first conclusion on the table is the stability of the approach with respect to the size of the log. Notice that the etc$_P$ value tends to increase as new behavior is considered, e.g., between *p20* and *p100* there is a difference of 0.05. However, this difference is extremely small considering that between *p20* and *p100* there is a 500% increment in the observed behavior. In addition, the more traces are included in the previously observed behavior, the closer the metric value is to stabilizing. The second conclusion to extract from this table is the dependency between the traces considered and the confidence in the metric, i.e., increasing the size of the trace set considered results in a narrower confidence interval.

The influence of parameters k (confidence, see Section 4.2) and γ (precision metric, see Section 4.1) in the approach are considered in the following experiment. In chart 15(a), three process models are considered: *ext_c1_01*, *ext_c1_02* and *ext_c1_03*.[10] These benchmarks have been created using the PLG [25] tool. This tool allows to create configurable and generic benchmarks, containing all the common patterns appearing in any workflow model, e.g., choice, parallelism and sequence. For the experiment, each one of the logs considered contains 15000 traces. Benchmarks 01 and 03 denote standard processes, with great difference between model and log behavior (and thus exhibiting low precision). On the other hand, process model *ext_c1_02* is a simpler model which describes accurately the behavior reflected in the log, i.e., the precision value is high. The chart illustrates the influence in the approach when considering diverse future windows, i.e., four different k values: 1, 500, 1000 or 2000 new traces to consider. In these experiments the worst case version of θ is considered, i.e., $|T| - 1$ (cf. Section 4.2). As it is reflected in the experiments, the future considered has no influence on the metric value, but it is relevant

[10]All experiment benchmarks are available in http://www.lsi.upc.edu/ jmunoz/software.html.

on the confidence value over the metric. The possibility of variation for the metric considering a really near future (i.e., $k = 1$) is practically zero. However, when considering further futures, this possibility increases, e.g., considering a k value of 2000 (approx. 15% of the log) the confidence in the metric is substantially low. Notice that, as expected, the confidence interval is not symmetric.

Chart 15(b) illustrates the relation between the γ parameter and the percentage of the process behavior considered to compute the precision. Two generic and representative process, *plg_01* and *plg_02*, have been created using PLG tool, and different values of γ have been tested. The conclusion we obtain is that for these processes, lower values of γ (i.e., less than 0.04) can be used to *polish* the effects produced by noisy traces, while greater values of γ (not considering more than 10% of the process behavior for computing the precision) should be used if the emphasis is in computing precision on the most frequent parts of the process. Values greater than 0.4 do not make any sense, due to the fact that the 100% of process is ruled out.

In addition, charts 15(c) and 15(d) are used to illustrate the part of the approach concerning the severity. Four generic models (sev01, sev02, sev03 and sev04) have been created using the PLG tool, which contain the most common structures in workflow models. For each model, a severity analysis has been performed, where each factor in Equation (7) has received the same weight. The same values of γ and τ have also been assigned for all the models (0.01 and 0.06, respectively). The criticality value assigned to each of the tasks has been different depending on the task and the model. In chart 15(c), we selected the most sever imprecision of each process, and showed the distribution of the imprecision for each one of the four factors. This chart illustrates the distribution of the weight of each factor in the final severity value, in this particular setting. In addition, it also illustrates that, given the normalization introduced in the definition of each one of the factors, it is possible to compare the severity between imprecision of different processes, e.g., given a system containing the four process models, and given the current setting, the imprecision shown in the second bar (which corresponds to model sev02) should be tackled first. In chart 15(d), the same four processes are considered. In this case, imprecisions of each process are classified in three categories: *low* (less than 0.3), *mid* (between 0.3 and 0.4) and *critical* (greater than 0.4). Notice that, in this particular scenario, the number of critical imprecisions (the ones that should be tackled urgently) is small (approx. 10 imprecisions for each process) compared with the total number of imprecisions.

Finally, the approach presented in this paper has been tested in a real world scenario. The scenario is taken from a Dutch Academic Hospital, and the log contains about 150.000 events in 1100 cases, where each case corresponds with a patient of the Gynecology department.[11] The goal of the experiment is to measure the quality of the models obtained using different discovery algorithms. The process miners used in the experiment are *RBMiner*[26], *Genet*[27], *ILPMiner* [24] and α-*miner*[1]. The results illustrate that the precision of the models obtained using such miners focusing on the whole process is very low. The models generated allow almost all the tasks most of the time, decreasing drastically the precision and consequently the quality of the models. For instance, the etc_P value of the models generated by α-miner and RBMiner are 0.03 and 0.01 respectively. However, that precision increases when we apply partition and clustering techniques over the log, to focus the mining on specific parts of the process. For instance, mining a model projecting the process over the set of the 10 most frequent events will result in a precision

[11]Log file DOI is doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54 and can be found in the 3TU Datacenter (http://data.3tu.nl).

of 0.397, 0.386 and 0.386 for *Genet*, *ILPMiner* and *α-miner* respectively. In the case of *RBMiner*, the precision is slightly greater, i.e., 0.423.

The approach presented has been implemented as *ETConformance* plug-in within ProM 6 tool. The current approach considers models in Petri net modeling language, the dominating model in ProM 6. Other models will be considered in the future.

7. Related Work.

The seminal paper in [8] represents the closer work to the techniques of this paper. In that paper, the authors present metrics for fitness, structural and also precision (called $a'b$). This precision metric is based on comparing the ordering relations between events in the model with the ones from the log. Due to this, the exhaustive exploration of model state space is required, becoming impractical for real life cases. In [18], the authors present an approach based on measuring the percentage of potential traces in the model that are in the log, while in [17] a technique for comparing the precision of two models with respect to a log is presented. In [28], a technique for measuring the precision of a model without a log (through *causal footprints*) is proposed. The approach is applied to Petri net models but also EPCs, and it can be applied to BPEL and UML models too. In [29], the authors propose also an independent-model approach based on the *minimal description length* principle. A complete study and evaluation of precision checking approaches, and conformance techniques in general, can be found in [2, 30].

8. Conclusions.

In this work, we have presented a metric to measure the precision between a log and a process model. The technique is designed to be model independent, being possible to instantiate it for different modeling languages. Together with the metric, the approach incorporates important and new aspects in conformance checking: the measurement of the confidence over the precision metric, and the multi-factored assessment of the imprecisions severity. The approach has been implemented as plug-in within ProM open-source process mining framework. Finally, a set of experiments has been performed to reveal the properties and the relevance of the concepts introduced in this paper.

Acknowledgment. This work is supported by the project FORMALISM (TIN2007-66523), and by the Spanish Ministerio de Educación with the FPU scholarship (AP2009-4959). The authors would like to thank Wil van der Aalst for his helpful comments and suggestions and Marc Solé for his help with the experiments section.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Springer, 2011.

[2] A. Rozinat, A. K. A. de Medeiros, C. W. Günther, A. J. M. M. Weijters and W. M. P. van der Aalst, Towards an evaluation framework for process mining algorithms, *BPM Center Report BPM-07-06*, BPMcenter. org, 2007.

[3] T. Murata, Petri nets: Properties, analysis and applications, *Proc. of the IEEE*, vol.77, no.4, pp.541-580, 1989.

[4] A. H. M. ter Hofstede, W. M. P. van der Aalst and M. Adams, *Modern Business Process Automation: YAWL and Its Support Environment*, Springer-Verlag, New York Inc, 2009.

[5] G. Keller, M. Nüttgens and A. Scheer, Semantische prozeßmodellierung auf der basis ereignisgesteuerter prozeßketten, *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, vol.89, 1992.

[6] S. White and D. Miers, *BPMN Modeling and Reference Guide: Understanding and Using BPMN*, Future Strategies Inc, 2008.

[7] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm and A. J. M. M. Weijters, Workflow mining: A survey of issues and approaches, *Data Knowl. Eng.*, vol.47, no.2, pp.237-267, 2003.

[8] A. Rozinat and W. M. P. van der Aalst, Conformance checking of processes based on monitoring real behavior, *Information Systems*, vol.33, no.1, pp.64-95, 2008.

[9] J. Munoz-Gama and J. Carmona, A fresh look at precision in process conformance, *Business Process Management, LNCS*, pp.211-226, 2010.

[10] J. Munoz-Gama and J. Carmona, Enhancing precision in process conformance: Stability, confidence and severity, *IEEE Symposium Series in Computational Intelligence 2011*, Paris, France, 2011.

[11] A. Arnold, *Finite Transition Systems*, Prentice Hall, 1994.

[12] B. F. van Dongen, A. K. A. de Medeiros and L. Wen, Process mining: Overview and outlook of petri net discovery algorithms, *T. Petri Nets and Other Models of Concurrency*, vol.2, pp.225-242, 2009.

[13] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst and N. Mulyar, Workflow control-flow patterns: A revised view, *Tech. Rep., BPM Center Report*, BPMcenter.org, 2006.

[14] C. Dufourd, A. Finkel and P. Schnoebelen, Reset nets between decidability and undecidability, *ICALP, LNCS*, vol.1443, pp.103-115, 1998.

[15] M. Dumas, A. Großkopf, T. Hettel and M. T. Wynn, Semantics of standard process models with OR-joins, *OTM Conferences, LNCS*, vol.4803, pp.41-58, 2007.

[16] E. Kindler, On the semantics of EPCs: Resolving the vicious circle, *Data Knowl. Eng.*, vol.56, no.1, pp.23-40, 2006.

[17] A. K. A. de Medeiros, *Genetic Process Mining*, Ph.D. Thesis, Technische Universiteit Eindhoven, 2006.

[18] G. Greco, A. Guzzo, L. Pontieri and D. Saccà, Discovering expressive process models by clustering log traces, *IEEE Trans. Knowl. Data Eng.*, vol.18, no.8, pp.1010-1027, 2006.

[19] A. Adriansyah, N. Sidorova and B. van Dongen, Cost-based fitness in conformance checking, *Application of Concurrency to System Design Conference*, Kanazawa, Japan, 2011.

[20] T. H. Cormen, C. Stein, R. L. Rivest and C. E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2001.

[21] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, 1998.

[22] B. Karrer, E. Levina and M. E. J. Newman, Robustness of community structure in networks, *Phys. Rev. E*, vol.77, no.4, p.046119, 2008.

[23] G. E. P. Box, W. G. Hunter and J. S. Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, Wiley, New York, 1978.

[24] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens and A. Serebrenik, Process discovery using integer linear programming, *Petri Nets, LNCS*, vol.5062, pp.368-387, 2008.

[25] A. Burattin and A. Sperduti, PLG: A framework for the generation of business process models and their execution logs, *Business Process Intelligence (BPI)*, 2010.

[26] M. Solé and J. Carmona, Process mining from a basis of state regions, *Petri Nets*, vol.6128, pp.226-245, 2010.

[27] J. Carmona, J. Cortadella and M. Kishinevsky, Genet: A tool for the synthesis and mining of petri nets (tool paper), *Proc. of the 2009 Application of Concurrency to System Design Conference*, Augsburg, Germany, pp.181-185, 2009.

[28] B. F. van Dongen, J. Mendling and W. M. P. van der Aalst, Structural patterns for soundness of business process models, *EDOC*, pp.116-128, 2006.

[29] T. Calders, C. W. Günther, M. Pechenizkiy and A. Rozinat, Using minimum description length for process mining, in *SAC*, S. Y. Shin and S. Ossowski (eds.), ACM, 2009.

[30] J. D. Weerdt, M. D. Backer, J. Vanthienen and B. Baesens, A critical evaluation study of model-log metrics in process discovery, *Business Process Intelligence*, 2010.