Quantifying process equivalence based on observed behavior


Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Accepted 12 June 2007
Available online 6 July 2007

Abstract

In various application domains there is a desire to compare process models, e.g., to relate an organization-specific process model to a reference model, to find a web service matching some desired service description, or to compare some normative process model with a process model discovered using process mining techniques. Although many researchers have worked on different notions of equivalence (e.g., trace equivalence, bisimulation, branching bisimulation, etc.), most of the existing notions are not very useful in this context. First of all, most equivalence notions result in a binary answer (i.e., two processes are equivalent or not). This is not very helpful because, in real-life applications, one needs to differentiate between slightly different models and completely different models. Second, not all parts of a process model are equally important. There may be parts of the process model that are rarely activated (i.e., “process veins”) while other parts are executed for most process instances (i.e., the “process arteries”). Clearly, differences in some veins of a process are less important than differences in the main arteries of a process. To address the problem, this paper proposes a completely new way of comparing process models. Rather than directly comparing two models, the process models are compared with respect to some typical behavior. This way, we are able to avoid the two problems just mentioned. The approach has been implemented and has been used in the context of genetic process mining. Although the results are presented in the context of Petri nets, the approach can be applied to any process modeling language with executable semantics.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Process mining; Petri nets; Genetic algorithms; Process discovery; Business process intelligence; Process equivalence

1. Introduction

Nowadays, large organizations typically have a wide variety of process models [17]. Some examples are: reference models (e.g., the EPC models in the SAP R/3 reference model [24]); workflow models (e.g., models used for enactment in systems like Staffware, FLOWer, FileNet, Oracle BPEL, etc. [2]); business process models/simulation models (e.g., using tools such as ARIS, Protos, Arena, etc. [17]); interface/service descriptions (e.g., the Partner Interface Processes in RosettaNet [30], the abstract BPEL processes in the context of web...
services [9], choreography descriptions using WSCDL [23], or other ad hoc notations [33]); and/or process models discovered using process mining techniques [5,6]. Given the co-existence of different models and different types of models, it is interesting to be able to compare process models.

This comparison of process models applies to different levels ranging from models at the business level to models at the level of software components (e.g., when looking for a software component matching some specification). To compare process models in a meaningful manner, we need to assume that these models have semantics. Moreover, we need to assume some equivalence notion (When are two models the same?). People working on formal methods have proposed a wide variety of equivalence notions [1,19,26], e.g., two models may be identical under trace equivalence but are different when considering stronger notions of equivalence (e.g., bisimulation). Unfortunately, most equivalence notions provide a “true/false” answer. In reality there will seldom be a perfect fit. Hence, we are interested in the degree of similarity, e.g., a number between 0 (completely different) and 1 (identical). In other to do so, we need to quantify the differences. Here it seems reasonable to put more emphasis on the frequently used parts of the model.

To clarify the problem, let us consider Fig. 1 where four process models (expressed in terms of Petri nets [14,22,29]) are depicted. These models describe the booking of a trip, see the legend for the interpretation of the various transitions in the Petri nets, e.g., C refers to the booking of a flight. Clearly, these models are similar. However, using classical equivalence notions all models are considered different. For example, in process (a) it is possible to have the execution sequence $ABDE$ while this sequence is not possible in (b) and (c). Moreover, the Petri net in Fig. 1d allows for $ACDFDE$ which is not possible in any of the other models. Note that we focus on the active parts of the net (i.e., the transitions) rather than passive things such as places. Although classical equivalence notions consider the four models to be different, it is clear that some are more similar than others. Therefore, we want to quantify “equality”, i.e., the degree of similarity. A naive approach could be to simply compare the sets of transition labels, e.g., nets (a) and (b) have the same transition labels: $\{A, B, C, D, E\}$ while (c) has a smaller set (without B) and (d) has a bigger set (with $F$). However, models with similar labels can have completely different behaviors (cf. a and b in Fig. 1). Therefore, it is important to consider causal dependencies and the ordering of activities, e.g., to distinguish between parallelism and choice. Another approach could be to consider the state spaces or sets of possible traces of both models. However,
in that case the problems are that there may be infinitely many traces/states and that certain paths are more probable.

In this paper, we investigate these problems and propose a completely new approach. The main idea is to compare two models relative to an event log containing “typical behavior”. This solves several problems when comparing different models. Even models having infinitely many execution sequences can be compared and automatically the relevance of each difference can be taken into account. Moreover, as we will show, we can capture the moment of choice and analyze causalities that may not be explicitly represented in the log.

To give some initial insights in our approach, consider the set of traces listed in Fig. 1f. Each trace represents an execution sequence that may or may not fit in the models at hand. Moreover, frequencies are given, e.g., in the event log trace $ABDE$ occurred 40 times, i.e., there were 40 process instances having this behavior. Fig. 1f represents some “typical behavior”. This may be obtained using simulation of some model or it could be obtained by observing some real-life system/process. All 160 traces fit into the first Petri net (cf. Fig. 1a), moreover, this Petri net does not allow for any execution sequences not present in the log. In this paper, we will quantify a notion of fitness. However, our primary objective is not to compare an event log and a process model, but to compare models in the presence of some event log as shown in Fig. 1f. Compare for example models (a) and (b): in a substantial number of cases (35) $D$ precedes $B$ or $C$. If we compare (a) and (c) based on the log, we can see that for 55 cases there is a difference regarding the presence of $B$. We will show that we can quantify these differences using the event log. It is important to note that we do not only consider full traces, e.g., if we compare Fig. 1a with a Petri net where $D$ is missing in the model, there is still some degree of similarity although none of the traces still fits (they all contain $D$).

To quantify differences between two models, we introduce precision and recall measures. Precision measures are used to define whether the second model’s behavior is possible according to the first model’s behavior. Recall measures are used to quantify how much of the first model’s behavior is covered by the second model.

This paper extends the results presented in [4] in three ways. Firstly, we investigate what can be inferred about the values of the defined precision and recall measures for two models – say $model_1$ and $model_2$ – and some typical behavior – say $log$ – when we know the measure values for these models when compared to another model – say $model_3$ – with respect to $log$. In other words, if the precision and recall values for $(model_1, model_2, log)$ and $(model_2, model_3, log)$ are known, what can be inferred about the values for $(model_1, model_3, log)$? Secondly, we reason if two models can be considered behaviorally equivalent when the precision and recall measures defined in this paper indicate that these models have the same behavior with respect to a given typical behavior. Thirdly, this paper also discusses the implementation of the precision and recall measures in the context of genetic mining (i.e., discovering process using genetic algorithms), where there is a need to compare models in an approximate manner. Moreover, in this domain the assumption of having example behavior in terms of event logs is very natural.

The remainder is organized as follows. After providing a brief overview of related work, we introduce some preliminaries required to explain our approach. Although we use Petri nets to illustrate our approach, any other process model with some local execution semantics (e.g., EPCs, activity diagrams, BPMN, etc.) could be used. In Section 4, we present two naive approaches (one based on the static structure and one based on a direct comparison of all possible behaviors) and discuss their limitations. Then, in Section 5 we present the core results of this paper. We will show that we can define precision and recall measures using event logs containing typical behavior. These notions have been implemented in ProM [16]. In sections 6 and 7 we reason about what can be inferred based on the results of the precision and recall measures over different models with respect to a same given log. Finally, we discuss the application of these results to a genetic mining approach and conclude the paper.

2. Overview of various equivalence notations and related work

In the literature, many equivalence notions have been defined for process models. Most equivalence notions focus on the dynamics of the model and not on the syntactical structure (e.g., trace equivalence and bisimulation [1,19,26]).

This paper uses Petri nets as a theoretical foundation [14,22,29]. In [28] an overview is given of equivalence notions in the context of Petri nets. See also [10] for more discussions on equivalence in the context of
nets. Most authors translate a Petri net to a transition system to give it semantics. However, there are also authors that emphasize the true-concurrency aspects when giving Petri nets semantics. For example, in [13] the well-known concept of occurrence nets (also named runs) are used to reason about the semantics of Petri nets.

Any model with formal/executable semantics (including Petri nets) can be translated to a (possibly infinite) transition system. If we consider transition systems, many notions of equivalence have been identified. The weakest notion considered is trace equivalence: two process models are considered equivalent if the sets of traces they can execute are identical. Trace equivalence has two problems: (1) the set of traces may be infinite and (2) trace equivalence does not capture the moment of choice. The first problem can be addressed in various ways (e.g., looking at finite sets of prefixes or comparing transition systems rather than traces). The second problem requires stronger notions of equivalence. Bisimulation and various kinds of observation equivalence [26] attempt to capture the moment of choice. For example, there may be different processes having identical sets of traces \{ABC, ABD\}, e.g., the process where the choice for C or D is made after executing A or the process where the same choice is made only after executing B. Branching bisimilarity [19] is a slightly finer equivalence notion than the well-known observation equivalence [26]. A comparison of branching bisimilarity, observation equivalence, and a few other equivalences on processes with silent behavior can be found in [19]. Branching bisimilarity can be checked in polynomial time (in terms of the size of the transition system) as shown in [20]. Based on these equivalence relations also other relations have been introduced, e.g., the four inheritance relations in [1] are based on branching bisimilarity.

All references mentioned so far, aim at a “true/false” answer. Moreover, they do not take into account that some parts of the process may be more important than others. Few people (e.g., [15]) have been working on probabilistic bisimulation using labeled Markov processes rather than labeled transition systems. See [15] for an excellent overview of this work and also links to the probability theory community working on metrics on spaces of measures. In this paper, we use a different approach. We do not assume that we know any probabilities. Instead we assume that we have some example behavior than can serve as a basis for a comparison of two models. Also related is the work on metric labeled transition systems where the “behavioral difference” between states is a non-negative real number indicating the similarity between those states [11]. This way one can define a behavioral pseudometric to compare transition systems as shown in [11]. Note that this approach very much depends on an explicit notion of states and it is not clear how this can be applied to a practical, mainly activity oriented, setting.

As far as we know, this paper and our work in [4] are among the first to propose the use of “typical behavior” recorded in event logs as an aid for comparison. This makes the work quite different from the references mentioned in this section. Moreover, we show that this can be used in the context of process mining [3,5,6].

3. Preliminaries

This section introduces some of the basic mathematical and Petri net related concepts used in the remainder.

3.1. Multi-sets, sequences, and matrices

Let \( A \) be a set. \( \mathcal{B}(A) = A \rightarrow \mathbb{N} \) is the set of multi-sets (bags) over \( A \), i.e., \( X \in \mathcal{B}(A) \) is a multi-set where for each \( a \in A \): \( X(a) \) denotes the number of times \( a \) is included in the multi-set. The sum of two multi-sets \( (X + Y) \), the difference \( (X - Y) \), the presence of an element in a multi-set \( (x \in X) \), and the notion of subset \( (X \subseteq Y) \) are defined in a straightforward way and they can handle a mixture of sets and multi-sets. The operators are also robust with respect to the domains of the multi-sets, i.e., even if \( X \) and \( Y \) are defined on different domains, \( X + Y \), \( X - Y \), and \( X \subseteq Y \) are defined properly by extending the domain where needed. \( |X| = \sum_{a \in A} X(a) \) is the size of some multi-set \( X \) over \( A \).

For a given set \( A \), \( A^n \) is the set of all finite sequences over \( A \). A finite sequence over \( A \) of length \( n \) is a mapping \( \sigma \in \{1, \ldots, n\} \rightarrow A \). Such a sequence is represented by a string, i.e., \( \sigma = \langle a_1, a_2, \ldots, a_n \rangle \) where \( a_i = \sigma(i) \) for \( 1 \leq i \leq n \). \( hd(\sigma,k) = \langle a_1, a_2, \ldots, a_k \rangle \), i.e., the sequence of just the first \( k \) elements. Note that \( hd(\sigma,0) \) is the empty sequence.
Every multi-set can be represented as a vector, i.e., \( X \in \mathbb{B}(A) \) can be represented as a row vector \( (x(a_1), x(a_2), \ldots, x(a_n)) \) where \( a_1, a_2, \ldots, a_n \) enumerate the domain of \( X \). \( (x(a_1), x(a_2), \ldots, x(a_n))^T \) denotes the corresponding column vector \( (^T \) transposes the vector). Assume \( X \) is an \( k \times \ell \) matrix, i.e., a matrix with \( k \) rows and \( \ell \) columns. A row vector can be seen as \( 1 \times \ell \) matrix and a column vector can be seen as a \( k \times 1 \) vector. \( x(i,j) \) is the value of the element in the \( i \)-th row and the \( j \)-th column. Let \( X \) be an \( k \times \ell \) matrix and \( Y \) an \( \ell \times m \) matrix. The product \( X \cdot Y \) is the product of \( X \) and \( Y \) yielding a \( k \times m \) matrix, where \( X \cdot Y(i,j) = \sum_{1 \leq q \leq \ell} X(i,q) Y(q,j) \). The sum of two matrices having the same dimensions is denoted by \( X + Y \).

For any sequence \( \sigma \in \{1, \ldots, n\} \to A \) over \( A \), the Parikh vector \( \overline{\sigma} \) maps every element \( a \) of \( A \) onto the number of occurrences of \( a \) in \( \sigma \), i.e., \( \overline{\sigma} \in \mathbb{B}(A) \) where for any \( a \in A : \overline{\sigma}(a) = \sum_{1 \leq i \leq n} \text{if } \sigma(i) = a \text{ then } 1 \text{ else } 0 \).

### 3.2. Petri nets

This subsection briefly introduces some basic Petri net terminology [14,22,29] and notations used in the remainder.

**Definition 1** (Petri net). A Petri net is a triple \( (P, T, F) \). \( P \) is a finite set of places, \( T \) is a finite set of transitions \( (P \cap T = \emptyset) \), and \( F \subseteq (P \times T) \cup (T \times P) \) is a set of arcs (flow relation).

Fig. 1 shows four Petri nets. Places are represented by circles and transitions are represented by squares. For any relation/directed graph \( G \subseteq A \times A \) we define the preset \( a^G = \{ a_1 \mid (a_1, a_2) \in G \} \) and postset \( a^\bullet = \{ a_2 \mid (a_1, a_2) \in G \} \) for any node \( a \in A \). We use \( G(a) \) or \( a^G \) to explicitly indicate the context \( G \) if needed. Based on the flow relation \( F \) we use this notation as follows. \( \bullet t \) denotes the set of input places for a transition \( t \). The notations \( \bullet, \cdot p \) and \( p \bullet \) have similar meanings, e.g., \( p \bullet \) is the set of places sharing \( p \) as an input place. Note that we do not consider multiple arcs from one node to another. In the Petri net shown Fig. 1(d): \( p^5 \bullet = \{ E, F \}, \cdot p^5 = \{ D \}, A^\bullet = \{ p_2, p_3 \}, \bullet A = \{ p_1 \} \), etc.

At any time a place contains zero or more tokens, drawn as black dots. The state of the Petri net, often referred to as marking, is the distribution of tokens over its places, i.e., \( M \in \mathbb{B}(P) \). In each of the four Petri nets shown in Fig. 1 only one place is initially marked \( (p_1) \). Note that more places could be marked in the initial state and that places can be marked with multiple tokens.

We use the standard firing rule, i.e., a transition \( t \) is said to be enabled if and only if each input place \( p \) of \( t \) contains at least one token. An enabled transition may fire, and if transition \( t \) fires, then \( t \) consumes one token from each input place \( p \) of \( t \) and produces one token for each output place \( p \) of \( t \). For example, in Fig. 1a, \( A \) is enabled and firing \( A \) will result in the state marking place \( p_2 \) and \( p_3 \). In this state both \( B, C \), and \( D \) are enabled. If \( B \) fires, \( C \) is disabled, but \( D \) remains enabled. Similarly, if \( C \) fires, \( B \) is disabled, but \( D \) remains enabled, etc. After firing four transitions in Fig. 1a the resulting state marks \( p_6 \) with one token (independent of the order of \( B \) or \( C \)). In the following definition, we formalize these notions.

**Definition 2** (Firing rule). Let \( N = (P, T, F) \) be a Petri net and \( M \in \mathbb{B}(P) \) be a marking.

- \( \text{enabled}(N, M) = \{ t \in T \mid M \geq \bullet t \} \) is the set of enabled transitions,
- \( \text{result}(N, M, t) = (M - \bullet t) + \cdot t \) is the state resulting after firing \( t \in T \),
- \( (N, M)[t](N, M') \) denotes that \( t \) is enabled in \( (N, M) \) (i.e., \( t \in \text{enabled}(N, M) \)) and that firing \( t \) results in marking \( M' \) (i.e., \( M' = \text{result}(N, M, t) \)).

\( (N, M)[t](N, M') \) defines how a Petri net can move from one marking to another by firing a transition. We can extend this notion to firing sequences. Suppose \( \sigma = \langle t_1, t_2, \ldots, t_n \rangle \) is a sequence of transitions present in some Petri net \( N \) with initial marking \( M \). \( (N, M)[\sigma](N, M') \) means that there is also a sequence of markings \( \langle M_0, M_1, \ldots, M_n \rangle \) where \( M_0 = M, M_n = M' \), and for any \( 0 \leq i < n : (N, M_i)[t_{i+1}](N, M_{i+1}) \). Using this notation we define the set of reachable markings \( R(N, M) \) as follows: \( R(N, M) = \{ M' \in \mathbb{B}(P) \mid \exists \sigma \in (N, M)[\sigma](N, M') \} \). Note that \( M \in R(N, M) \) because \( M \) is reachable via the empty sequence.

Note that \( \text{result}(N, M, t) \) does not need to yield a multi-set if \( t \) is not enabled in marking \( M \) because some places may have a negative number of tokens. Although this is not allowed in a Petri net (only enabled transitions can...
fire), for technical reasons it is sometimes convenient to use markings that may have “negative tokens”. This becomes clear when considering the incidence matrix of a Petri net.

**Definition 3 (Incidence matrix).** Let \( N = (P, T, F) \) be a Petri net and \( M \in \mathbb{R}(P) \) be a marking.

- \( \tilde{N} \) is the incidence matrix of \( N \), i.e., \( \tilde{N} \) is a \([P] \times [T]\) matrix with \( \tilde{N}(p, t) = 1 \) if \( (p, t) \not\in F \) and \( (t, p) \in F \), \( \tilde{N}(p, t) = -1 \) if \( (p, t) \in F \) and \( (t, p) \not\in F \), and \( \tilde{N}(p, t) = 0 \) in all other cases,
- \( \text{result}(N, M, \sigma) = M + \tilde{N} \cdot \sigma \) is the state resulting after firing \( \sigma \in T^* \).
- \( \text{enabled}(N, M, \sigma) = \text{enabled}(N, \text{result}(N, M, \sigma)) \) is the set of enabled transitions after firing \( \sigma \in T^* \).

The incidence matrix of a Petri net can be used for different types of analysis, e.g., based on \( \tilde{N} \) it is possible to efficiently calculate place and transition invariants and to provide minimal (but not sufficient) requirements for the reachability of a marking. It is important to see that \( \text{result}(N, M, \sigma) \) does not need to yield a valid marking, i.e., there may be a place \( p \) such that \( \text{result}(N, M, \sigma)(p) < 0 \) indicating a negative number of tokens. If \( (N, M)[\sigma](N, M') \), then \( \text{result}(N, M, \sigma) = M' \). However, the reverse does not need to be the case. \( \text{enabled}(N, M, \sigma) \) calculates which transitions are enabled after firing each transition \( \sigma \) times using function \( \text{result} \) and the earlier defined function \( \text{enabled} \) (cf. **Definition 2**). It may be the case that while executing \( \sigma \) starting from \( (N, M) \), transitions were forced to be fired although they were not enabled. As a result, places may get a negative number of tokens. The reason we need such concepts is because we will later compare Petri nets with some observed behavior. In such situations, we need to be able to deal with transitions that were observed even if they were not enabled.

### 4. Naive approaches

In this paper we propose to compare two processes on the basis on some event log containing typical behavior. However, before presenting this approach in detail, we first discuss some naive approaches.

#### 4.1. Equivalence of processes based on their structure

When humans compare process models they typically compare the graphical structure, i.e., do the same activities (transitions in Petri net terms) appear in both models and do they have similar connections. Clearly, the graphical structure may be misleading: two models that superficially appear similar may be very different. Nevertheless, the graphical structure is an indicator that may be used to quantify similarity. Let us abstract from the precise split and join behavior (i.e., we do not distinguish between AND/XOR-splits/joins). In other words, we derive a simple graph where each node represents an activity and each arc some kind of connection. For example, the Petri net shown in **Fig. 1a** is reduced to a graph with nodes \( A, B, C, D \) and \( E \), and arcs \( (A, B), (A, C), (A, D), (B, E), (C, E) \) and \( (D, E) \). For the other Petri net models in **Fig. 1** a similar graph structure can be derived. It is easy to see that each of the four process models has a different graph structure. However, there are many overlapping connections, e.g., all models have arc \( (A, C) \). This suggests that from a structural point of view the models are not equivalent but similar. When quantifying the overlap relative to the whole model we can take the perspective of the first model or the second model. This leads to the definition of precision and recall as specified below.\(^2\)

**Definition 4 (Structural precision and recall).** Let \( N_1 = (P_1, T_1, F_1) \) and \( N_2 = (P_2, T_2, F_2) \) be two Petri nets. Using \( C_1 = \{(t_1, t_2) \in T_1 \times T_1 \mid t_1 \bullet N_1 \cap t_2 \neq \emptyset \} \) and \( C_2 = \{(t_1, t_2) \in T_2 \times T_2 \mid t_1 \bullet N_2 \cap t_2 \neq \emptyset \} \), we define:

\[
\text{precision}^S(N_1, N_2) = \frac{|C_1 \cap C_2|}{|C_2|}, \quad \text{recall}^S(N_1, N_2) = \frac{|C_1 \cap C_2|}{|C_1|}
\]

\(^1\)Note that \( \sigma \) does not need to be enabled, i.e., transitions are forced to fire even if they are not enabled. Also note that we do not explicitly distinguish row and column vectors.

\(^2\)These metrics are an adaptation of the precision and recall metrics in [27].
However, it is easy to extend Definition 1 to so-called labeled Petri nets where different transitions can have the same label.

Fig. 2. Although the connection structures of (a) and (b) are similar they are quite different in terms of behavior. Moreover, the connection structure of (a) and (c) differs while the corresponding behaviors are identical.

Let \( N_a, N_b, N_c, \) and \( N_d \) be the four Petri nets shown in Fig. 1.

\[
\text{precision}^S(N_a, N_b) = \frac{|\{(A, B), (A, C), (D, E)\}|}{|\{(A, B), (A, C), (B, D), (C, D), (D, E)\}|} = \frac{3}{5} = 0.6
\]

\[
\text{recall}^S(N_a, N_b) = \frac{|\{(A, B), (A, C), (D, E)\}|}{|\{(A, B), (A, C), (A, D), (B, E), (C, E), (D, E)\}|} = \frac{3}{6} = 0.5
\]

Note that \( \text{precision}^S(N_1, N_2) = \text{recall}^S(N_2, N_1) \) by definition for any pair of Petri nets \( N_1 \) and \( N_2 \). Therefore, we only list some precision values: \( \text{precision}^S(N_a, N_b) = 0.6, \text{precision}^S(N_a, N_c) = 4/4 = 1.0, \text{precision}^S(N_a, N_d) = 6/8 = 0.75, \text{precision}^S(N_b, N_d) = 3/6 = 0.5, \text{precision}^S(N_b, N_c) = 2/4 = 0.5, \text{precision}^S(N_b, N_d) = 3/8 = 0.375, \) etc.

If we consider \( N_a \) to be the initial model, then \( N_c \) has the best precision of the other three models because all connections in \( N_c \) also appear in \( N_a \). Moreover, if we consider \( N_a \) to be the initial model, then \( N_d \) has the best recall because all connections in \( N_a \) also appear in \( N_d \).

The precision and recall figures for the four process models in Fig. 1 seem reasonable. Unfortunately, models with nearly identical connections may be quite different as is shown in Fig. 2. Let \( N_a, N_b, N_c, \) and \( N_d \) be the four Petri nets shown in Fig. 2. Although \( \text{precision}^S(N_a, N_b) = \text{recall}^S(N_a, N_b) = 1 \), \( N_a \) and \( N_b \) are clearly different. In \( N_a \) transitions \( B \) and \( C \) are executed concurrently while in \( N_b \) a choice is made between these two transitions. However, although \( N_a \) and \( N_c \) are structurally different (\( \text{precision}^S(N_a, N_c) = 4/5 = 0.8 \)), they have identical behaviors. These examples show that Definition 4 does not provide a completely satisfactory answer when it comes to process equivalence. Nevertheless, \( \text{precision}^S(N_1, N_2) \) and \( \text{recall}^S(N_1, N_2) \) can be used as rough indicators for selecting a similar model, e.g., in a repository of reference models.

\(^3\) Note that strictly speaking \( N_d \) does not correspond to a Petri net as defined in Definition 1, because there are two transitions \( A \). However, it is easy to extend Definition 1 to so-called labeled Petri nets where different transitions can have the same label.
4.2. Equivalence of processes based on their state space or traces

Since process models with a similar structure may have very different behaviors and models with different structures can have similar behaviors, we now focus on quantifying the equivalence of processes based on their actual behaviors. We start with a rather naive approach where we define recall and precision based on the full firing sequences of two marked Petri nets.

**Definition 5 (Naïve behavioral precision and recall).** Let \( N_1 = (P_1, T_1, F_1) \) and \( N_2 = (P_2, T_2, F_2) \) be two Petri nets having initial markings \( M_1 \) and \( M_2 \) respectively. Moreover, let the corresponding two sets of possible full firing sequences be finite:

\[
S_1 = \{ \sigma \in T_1 \mid \exists M' \in B(P_1)(N_1, M_1)[\sigma](N_1, M') \land \text{enabled } (N_1, M') = \emptyset \} \text{ and } \\
S_2 = \{ \sigma \in T_2 \mid \exists M' \in B(P_2)(N_2, M_2)[\sigma](N_2, M') \land \text{enabled } (N_2, M') = \emptyset \}.
\]

\[
\text{precision}^B((N_1, M_1), (N_2, M_2)) = \frac{|S_1 \cap S_2|}{|S_2|} \quad \text{and} \quad \\
\text{recall}^B((N_1, M_1), (N_2, M_2)) = \frac{|S_1 \cap S_2|}{|S_1|}.
\]

Clearly, the initial markings of \( N_1 \) and \( N_2 \) are highly relevant. However, if these are clear from the context, we do not explicitly mention these, i.e., \( \text{precision}^B(N_1, N_2) = \text{precision}^B((N_1, M_1),(N_2, M_2)) \) and \( \text{recall}^B(N_1, N_2) = \text{recall}^B((N_1, M_1),(N_2, M_2)) \).

Let \( N_a, N_b, N_c, \) and \( N_d \) be the four Petri nets shown in Fig. 2 and \( S_a, S_b, S_c, \) and \( S_d \) their corresponding full firing sequences. \( S_a = \{(A, B, C, D), (A, C, B, D)\}, \ S_b = \{(A, B, D), (A, C, D)\}, \ S_c = S_a, \) and \( S_d = S_b \). Hence, \( \text{precision}^B(N_a, N_b) = 0 \) and \( \text{recall}^B(N_a, N_b) = 0 \), i.e., the models are considered to be completely different because there are no identical full firing sequences possible in both models. However, \( \text{precision}^B(N_a, N_c) = 1 \) and \( \text{recall}^B(N_a, N_c) = 1 \) and \( \text{precision}^B(N_b, N_d) = 1 \) and \( \text{recall}^B(N_b, N_d) = 1 \).

We can also consider the four process models in Fig. 1. The fourth model \( (N_d) \) has an infinite set of full firing sequences. Therefore, we focus on the first three models: \( N_a, N_b, \) and \( N_c \). Let us first compare \( N_a \) and \( N_b \): \( \text{precision}^B(N_a, N_b) = 2/2 = 1 \) and \( \text{recall}^B(N_a, N_b) = 2/4 = 0.5 \), i.e., all full firing sequences in \( N_b \) are possible in \( N_a \) but not the other way around. Although \( N_c \) differs from \( N_b \), the precision and recall values are identical when comparing with \( N_a \), i.e., \( \text{precision}^B(N_a, N_c) = 1 \) and \( \text{recall}^B(N_a, N_c) = 0.5 \).

These examples show that **Definition 5** provides another useful quantification of equivalence quite different from **Definition 4**. However, also this quantification has a number of problems:

1. The set of full firing sequences needs to be finite. This does not need to be the case as is illustrated by the Petri net shown in Fig. 1d. For such models, the metric becomes useless.
2. The models need to be terminating, i.e., it should be possible to end in a dead marking representing the completion of the process. Note that models may have unintentional livelocks or are designed to be non-terminating. For such models, we cannot apply **Definition 5** in a meaningful way. It also does not make sense to look at all possible firing sequences (i.e., also firing sequences that are not full firing sequences), because this would include the prefixes of both terminating and non-terminating sequences. As a result, new problems are introduced, e.g., more emphasis on the behavior typically contained in prefixes and possibly infinite sets.
3. **Definition 5** does not take into account differences between important paths or parts versus unimportant paths or parts of the model. For example, certain full firing sequences may have a very low probability in comparison to other sequences that occur more frequent. There may be parts of the process model that are rarely activated (earlier named “process veins”) while other parts are executed for all process instances (earlier named “process arteries”). Clearly this should be taken into account.
4. Fourth, **Definition 5** appears to be too rigid, i.e., one difference in a full firing sequence invalidates the entire sequence. In Fig. 2 precision\(^B(N_a, N_b) = 0 \) and recall\(^B(N_a, N_b) = 0 \) although both models always start with \( A \) and end with \( D \).
5. The moment of choice is not taken into account in Definition 5, i.e., essentially trace equivalence is used as a criterion. Many authors [1,19,26] have emphasized the importance of preserving the moment of choice by defining notions such as observation equivalence, bisimilarity, branching/weak bisimilarity, etc. To illustrate the importance of preserving the moment of choice, consider \( N_b \) and \( N_d \) depicted in Fig. 2. Although precision \( B(N_b,N_d) = 1 \) and recall \( B(N_b,N_d) = 1 \), most environments will be able to distinguish both processes. In \( N_b \) in Fig. 2b there is a state where only \( B \) or just \( C \) is enabled. However, such states exist in \( N_d \) in Fig. 2d, e.g., there can be a token in \( p2 \) enabling only \( B \). Suppose that \( B \) and \( C \) correspond to the receipt of different messages sent by some environment. In this case, \( N_b \) potentially deadlocks, e.g., a message for \( B \) cannot be handled because the system is waiting for \( C \) (i.e., \( p3 \) is marked). Such a deadlock is not possible in \( N_b \).

The problems listed above show that similarity metrics based on criteria directly comparing all possible behaviors in terms of traces are of little use from a practical point of view. An alternative approach is to compare two models. This exemplary behavior can be obtained on the basis of real process executions (in case the process already exists), user-defined scenarios, or by simply simulating one of the two models (or both). We assume this exemplary behavior to be recorded in an event log.

Definition 6 (Event log). An event log \( L \) is a multi-set of sequences on some set of \( T \), i.e., \( L \in \mathbb{B}(T^*) \).

An event log can be considered as a multi-set of full firing sequences (cf. Definition 5). However, now these sequences may exist independent of some model and the same sequence may occur multiple times.

Before comparing two process models using an event log, we first define the notion of fitness. This notion is inspired by earlier work on genetic mining and conformance checking [25,31].

Definition 7 (Fitness). Let \((N,M)\) be a marked Petri net and let \( L \in \mathbb{B}(T^*) \) be a multi-set over \( T \).\(^4\)

\[
\text{fitness}(L) = \left( \frac{1}{|L|} \sum_{\sigma \in L} \frac{L(\sigma)}{|\sigma|} \cdot \left| \{i \in \{0, |\sigma| - 1\} | \sigma(i+1) \in \text{enabled}(N,M,hd(\sigma,i)) \} \right| \right)
\]

fitness\((N,M,L)\) yields a number between 0 and 1. Note that per sequence \( \sigma \in L \) we calculate the number of times that a transition that was supposed to fire according to \( \sigma \) was actually enabled. This is divided by \(|\sigma|\) to yield a number between 0 and 1 per sequence. This number shows the “fit” of \( \sigma \). This is repeated for all \( \sigma \in L \). Since the same sequence may appear multiple times in \( L \) (i.e., \( L(\sigma) > 1 \)), we multiply the result for \( \sigma \) with \( L(\sigma) \) and divide by \(|L|\). Definition 7 assumes that \(|L| > 0 \) and \(|\sigma| > 0 \). This is not a fundamental restriction, if such strange cases occur (empty event log or an empty sequence), then we can simply assume that \( 0/0 = 0 \).

As an example, consider the event log \( L \) shown in Fig. 1f containing 160 traces. Clearly, \( \text{fitness}(N_{a'},L) = 1 \) because all sequences in \( L \) can be reproduced by \( N_{a'} \).\(^5\) Moreover, \( \text{fitness}(N_{b'},L) = (40 + 85 + (15 * 3/4) + (20 * 3/4))/160 = 0.945 \); \( \text{fitness}(N_c,L) = (40 * 1/2) + 85 + (15 * 1/2) + 20)/160 = 0.828 \), and \( \text{fitness}(N_{a''},L) = 1 \). These examples show that Definition 7 matches our intuitive understanding of fitness. It is important to note that transitions are “forced” to fire even if they are not enabled, cf. Definition 3. Moreover,

\(^4\) Note that not all events in the log need to correspond to actual transitions. These events are simply ignored, i.e., we assume \( \text{enabled}(N,M,\sigma) \) to be defined properly even if not all transitions in \( \sigma \) actually appear in \( N \).

\(^5\) Note that again we omit the initial marking if it is clear from the context, i.e., \( \text{fitness}(N_{a'},L) = \text{fitness}(N_{a'}[p1],L) \).
a particular sequence can be “partly fitting”, e.g., if we parse sequence \( \langle A, B, D, E \rangle \) using \( N_c \) in Fig. 1c, half of the sequence fits. When forcing the execution of \( \langle A, B, D, E \rangle \) using \( N_c \), \( A \) is initially enabled. However, \( B \) is not enabled and does not even exist in the model. Nevertheless, in the resulting state \( D \) is still enabled. However, after firing \( D \), the last event in the sequence \( (E) \) is not enabled. Hence, only two of the four events in \( \langle A, B, D, E \rangle \) are actually enabled, resulting in a fitness of 0.5. Note that it is better to look at individual events rather than considering whole sequences like in Definition 5. Using Definition 7, \( \text{fitness}(N_c, L) = 0.828 \). However, if we would focus on completely fitting sequences, \( \text{fitness}(N_c, L) = (0 + 85 + 0 + 20)/160 = 0.656 \), i.e., considerably lower because partly fitting are ignored.

Inspired by the definition of fitness, we would like to compare two models on the basis of a log. A straightforward extension of Definition 7 to two models is to compare the overlap in fitting or partially fitting sequences. However, in this case one only considers the actual behavior contained in the log. Therefore, we go one step further and look at the enabled transitions in both models and compare these, i.e., we do not just check whether an event in some sequence is possible, but also take into account all enabled transitions at any point in the sequence. This idea results in the following definition of precision and recall.

**Definition 8 (Behavioral precision and recall).** Let \((N_1, M_1)\) and \((N_2, M_2)\) be marked Petri nets and let \( L \in \mathbb{B}(\mathbb{T}^* ) \) be a multi-set over \( \mathbb{T} \).\(^6\)

\[
\text{precision}((N_1, M_1), (N_2, M_2), L) = \left( \frac{\sum_{\sigma \in \mathbb{T}} \frac{L(\sigma)}{|T|} \left( \sum_{i=0}^{\sigma-1} \frac{|\text{enabled}(N_1, M_1, hd(\sigma, i)) \cap \text{enabled}(N_2, M_2, hd(\sigma, i))|}{|\text{enabled}(N_2, M_2, hd(\sigma, i))|} \right)}{|L|} \right)
\]

\[
\text{recall}((N_1, M_1), (N_2, M_2), L) = \left( \frac{\sum_{\sigma \in \mathbb{T}} \frac{L(\sigma)}{|T|} \left( \sum_{i=0}^{\sigma-1} \frac{|\text{enabled}(N_1, M_1, hd(\sigma, i)) \cap \text{enabled}(N_2, M_2, hd(\sigma, i))|}{|\text{enabled}(N_1, M_1, hd(\sigma, i))|} \right)}{|L|} \right)
\]

To explain the concept consider a log \( L = \{(A, B, C, D), 2\}, (A, C, B, D), 1\}\) and the first three Petri nets shown in Fig. 2. \( \text{precision}(N_a, N_b, L) = ((40/4 \times (1/1 + 2/2 + 1/1 + 1/1)) + (85/4 \times (1/1 + 2/2 + 1/1 + 1/1)) + (15/4 \times (1/1 + 2/2 + 2/3 + 1/1)) + (20/4 \times (1/1 + 2/2 + 2/3 + 1/1)))/160 = 0.98 \) and \( \text{recall}(N_a, N_b, L) = ((40/4 \times (1/1 + 2/2 + 1/1 + 1/1)) + (85/4 \times (1/1 + 1/2 + 2/3 + 1/1)) + (15/4 \times (1/1 + 2/2 + 2/3 + 1/1)) + (20/4 \times (1/1 + 2/2 + 1/1 + 1/1)))/3 = 0.75 \).

We can also consider the four process models in Fig. 1 with respect to the logs shown in Fig. 1f. \( \text{precision}(N_a, N_b, L) = ((40/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (85/4 \times (1/1 + 2/2 + 1/1 + 1/1)) + (15/4 \times (1/1 + 2/2 + 2/3 + 1/1)) + (20/4 \times (1/1 + 2/2 + 2/3 + 1/1)))/160 = 0.98 \) and \( \text{recall}(N_a, N_b, L) = ((40/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (85/4 \times (1/1 + 1/2 + 2/3 + 1/1)) + (15/4 \times (1/1 + 2/2 + 2/3 + 1/1)) + (20/4 \times (1/1 + 2/2 + 1/1 + 1/1)))/160 = 0.92 \). Note that both numbers would be lower if the sequences starting with \( \langle A, D, \ldots \rangle \) would be more frequent. Let us now compare \( N_a \) and \( N_d \) in Fig. 1 using \( L \). \( \text{precision}(N_a, N_d, L) = ((40/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (85/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (15/4 \times (1/1 + 3/3 + 2/3 + 1/2)) + (20/4 \times (1/1 + 3/3 + 2/3 + 1/2)))/160 = 0.75 \) and \( \text{recall}(N_a, N_d, L) = ((40/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (85/4 \times (1/1 + 3/3 + 1/1 + 1/1)) + (15/4 \times (1/1 + 3/3 + 2/3 + 1/1)) + (20/4 \times (1/1 + 3/3 + 2/3 + 1/1)))/160 = 1 \). Note that \( N_d \) allows for behavior not present in log \( L \) (i.e., executing \( F \)). Nevertheless, as we can see from \( \text{precision}(N_a, N_d, L) = 0.75 \), the enabling of \( F \) is taken into account. It is also easy to see that Definition 8 takes into account the moment of choice, i.e., the enabling of set of transitions is the basis of comparison rather than the resulting sequences. Hence, we can distinguish \( N_b \) and \( N_a \) in Fig. 2.\(^7\)

In Section 4.2 we listed five problems related to the use of Definition 5. It is easy to see that Definition 8 addresses each of these problems:

\[^6\] Note that the two denominators \(|\text{enabled}(N_2, M_2, hd(\sigma, i))|\) and \(|\text{enabled}(N_1, M_1, hd(\sigma, i))|\) may evaluate to zero. In these cases, the numerator is also zero. Again, we assume in such cases that \( 0/0 = 0 \).

\[^7\] Note that \( N_d \) contains duplicate labels, i.e., two transitions with label \( A \). However, it is possible to extend Definition 8 and the resulting approach for such models.
1. Even models with an infinite set of firing sequences can be compared using a finite, but representative, set of traces.
2. Models do not need to be terminating.
3. Differences between frequent and infrequent sequences can be taken into account by selecting a representative log.
4. Partial fits are taken into account, i.e., small local differences do not result in a complete “misfit”.
5. The moment of choice is taken into account because the focus is on enabling.

Given the attractive properties of the precision and recall metrics defined in Definition 8, we have implemented these metrics in the ProM framework [16]. Here it has been applied to a variety of process models as will be discussed in Section 8.

One of critical success factors is the availability of some log \( L \) that can serve as a basis for comparison. We propose to use existing event logs or to generate artificial logs using simulation.

Existing logs can be extracted from information systems but can also be obtained by manually describing some typical scenarios. It is important to realize that today’s information systems are logging a wide variety of events. For example, any user action is logged in ERP systems like SAP R/3, workflow management systems like Staffware, and case handling systems like FLOWer. Classical information systems have some centralized database for logging such events (called transaction log or audit trail). Modern service-oriented architectures record the interactions between web services (e.g., in the form of SOAP messages). Moreover, today’s organizations are forced to log events by national or international regulations (cf. the Sarbanes–Oxley (SOX) Act [32] that is forcing organizations to audit their processes).

An example application scenario where existing event logs are used is the comparison of an existing process and a set of possible redesigns. For each of the redesigns, we can measure the precision and recall taking an event log of the existing information system as a starting point. First of all, the existing process can be compared with this event log using the fitness notion presented in this section. This gives an indication of the quality of the initial model. Then, if the quality is acceptable, each of the redesigns can be compared with the existing process using this log.

Another approach would be to use simulation. This simulation could be based on both models or just the initial model. Note that the generated logs do not need to be complete, because Definition 8 also takes the enabling into account, because differences in the “process veins” are of less importance than differences in the “process arteries”.

6. When are behavioral precision and recall metrics “transitive”?

As illustrated in Fig. 3, this section explores what can be “transitively” inferred about the values of the behavioral precision and recall metrics (cf. Definition 8) for three models and a log. In other words, given that you have (i) three process models \( N_1, N_2 \) and \( N_3 \) with the respective initial markings \( M_1, M_2, \) and \( M_3 \), (ii) an event log \( L \) and (iii) the values for the precision and recall metrics for \( N_1 \) and \( N_2 \) and for \( N_2 \) and \( N_3 \) over the same log \( L \) and using the respective initial markings, we analyze what can be said about the precision and recall values for \( N_1 \) and \( N_3 \). The results for the different scenarios are summarized in Table 1.

To understand Table 1, we formulate our main question as follows. Suppose we have three marked nets \((N_1, M_1), (N_2, M_2)\) and \((N_3, M_3)\), and a log \( L \). Moreover, assume that the following behavioral precision and recall metrics are known: \( \text{precision}((N_1, M_1), (N_2, M_2), L) = z \), \( \text{recall}((N_1, M_1), (N_2, M_2), L) = w \), \( \text{precision}((N_2, M_2), (N_3, M_3), L) = x \) and \( \text{recall}((N_2, M_2), (N_3, M_3), L) = y \). The main question is then: What are the values of the precision \((N_1, M_1), (N_3, M_3), L\) and the recall \((N_1, M_1), (N_3, M_3), L\)? The results in Table 1 are motivated in the remainder. Note that rather than providing formal proofs we give core arguments. However, first we provide an obvious lemma (cf. Lemma 1). Using the insights from this lemma, we show the reasoning behind the 16 scenarios in Table 1.

---

8 ProM and the analysis plug-in implementing the precision and recall metrics can be downloaded from www.processmining.org.
Lemma 1. Let \((N_1, M_1)\) and \((N_2, M_2)\) be two marked Petri nets and let \(L\) be a log.

- If \(\text{precision}((N_1, M_1), (N_2, M_2), L) = 1.0\) then \(\forall \sigma \in L \forall i \in (0, |\sigma| - 1) \enabled(N_1, M_1, \text{hd}(\sigma, i)) \subseteq \enabled(N_2, M_2, \text{hd}(\sigma, i))\);
- If \(\text{recall}((N_1, M_1), (N_2, M_2), L) = 1.0\) then \(\forall \sigma \in L \forall i \in (0, |\sigma| - 1) \enabled(N_1, M_1, \text{hd}(\sigma, i)) \subseteq \enabled(N_2, M_2, \text{hd}(\sigma, i))\);
- If \(\text{precision}((N_1, M_1), (N_2, M_2), L) = \text{recall}((N_1, M_1), (N_2, M_2), L) = 1.0\) then \(\forall \sigma \in L \forall i \in (0, |\sigma| - 1) \enabled(N_1, M_1, \text{hd}(\sigma, i)) = \enabled(N_2, M_2, \text{hd}(\sigma, i))\).

Proof. Follows directly from Definition 8. If both nets do not enable the same set of transitions, the precision and recall values will be less then 1. \(\Box\)
Scenario 13, 14, 15 and 16. Since $N_1$ and $N_2$ have the exact same enabled transitions while replaying the log (i.e., $\text{precision}(N_1, M_1), (N_2, M_2), L = 1.0$ and $\text{recall}(N_1, M_1), (N_2, M_2), L = 1.0$), the intersection between the enabled transitions of $N_1$ and of $N_2$ while replaying the log will be the same as for $N_2$ and $N_3$. Therefore, we can conclude that $\text{precision}(N_1, M_1), (N_3, M_3), L = \text{precision}(N_2, M_2), (N_3, M_3), L$ and $\text{recall}(N_1, M_1), (N_3, M_3), L = \text{recall}(N_2, M_2), (N_3, M_3), L$.

Scenarios 4, 8 and 12. Scenarios 4, 8 and 12 are similar to scenarios 13, 14 and 15. Just swap the models $N_1$ and $N_3$ in Fig. 3 to get the scenarios already explained for 13, 14 and 15.

Scenario 6. For this scenario, $\text{recall}(N_1, M_1), (N_3, M_3), L = 1.0$ because, while replaying the log, (i) all transitions that are enabled in $N_1$ are also in $N_2$ (i.e., $\text{recall}(N_1, M_1), (N_2, M_2), L = 1.0$) and (ii) all transitions that are enabled in $N_2$ are also in $N_3$ (i.e., $\text{recall}(N_2, M_2), (N_3, M_3), L = 1.0$). Thus, we can conclude that all transitions that are enabled in $N_1$ are also going to be enabled in $N_3$ (i.e., $\text{recall}(N_1, M_1), (N_3, M_3), L = 1.0$) because these transitions are contained in the set of transitions that were enabled for $N_2$ while calculating the value of $\text{recall}(N_2, M_2), (N_3, M_3), L$. $\text{precision}(N_1, M_1), (N_3, M_3), L < \text{precision}(N_2, M_2), (N_3, M_3), L$ because, since $\text{recall}(N_1, M_1), (N_2, M_2), L = 1.0$ and $\text{precision}(N_2, M_2), (N_3, M_3), L \neq 1.0$, the intersection between the enabled transitions in $N_3$ and $N_1$ (cf. numerator for the precision formula in Definition 8) can only be smaller than the intersection for $N_3$ and $N_2$. Furthermore, the intersection is always divided by the same number (i.e., the denominator “$\text{enabled}(N_3, M_3, hd(\sigma,j))$” remains constant). Therefore, we can conclude that $\text{precision}(N_1, M_1), (N_3, M_3), L < \text{precision}(N_2, M_2), (N_3, M_3), L$.

Scenario 11. A similar reasoning for the precision (recall) in Scenario 6 is used for the recall (precision) in Scenario 11.

Scenario 10. Since all transitions enabled for $N_2$ during the log replay are also enabled in $N_1$ (cf. $\text{precision}(N_1, M_1), (N_2, M_2), L = 1.0$) and in $N_3$ (cf. $\text{recall}(N_2, M_2), (N_3, M_3), L = 1.0$), $N_1$ and $N_3$ cannot have fewer intersecting transitions that are enabled at a given moment than the enabled transitions for $N_2$. However, since we do not make any assumptions about the transitions in these models, it is possible that $N_1$ and $N_3$ have more enabled transitions in common while replaying the log than they have with $N_2$. Therefore, we can conclude that $\text{precision}(N_1, M_1), (N_3, M_3), L \geq \text{precision}(N_2, M_2), (N_3, M_3), L$ and $\text{recall}(N_1, M_1), (N_3, M_3), L \geq \text{recall}(N_1, M_1), (N_2, M_2), L$.

Scenario 9. For this scenario, $\text{precision}(N_1, M_1), (N_3, M_3), L \geq \text{precision}(N_2, M_2), (N_3, M_3), L$ because, while replaying the log, in the worst scenario, $N_3$ will have at least as many elements in common with $N_1$ as it has with $N_2$. Note that $N_3$ allows for more behavior than $N_2$ (cf. $\text{precision}(N_2, M_2), (N_3, M_3), L = \chi$), but all the behavior enabled in $N_2$ is also enabled in $N_1$ (cf. $\text{precision}(N_1, M_1), (N_2, M_2), L = 1.0$), thus $N_3$ cannot have fewer enabled transitions in common with $N_1$ than it has with $N_2$. However, nothing can be said about $\text{recall}(N_1, M_1), (N_3, M_3), L$ because $N_1$ can have more or fewer enabled transitions than $N_3$. Therefore, $\text{recall}(N_1, M_1), (N_3, M_3), L$ can assume any value between 0 (inclusive) and 1 (inclusive).

Scenario 5. For this scenario, $\text{precision}(N_1, M_1), (N_3, M_3), L \leq \text{precision}(N_2, M_2), (N_3, M_3), L$ and $\text{recall}(N_1, M_1), (N_3, M_3), L \leq 1.0$. $\text{precision}(N_1, M_1), (N_3, M_3), L \leq \text{precision}(N_2, M_2), (N_3, M_3), L$ because, since (i) all enabled transitions in $N_1$ are also enabled in $N_2$ while replaying the log (cf. $\text{recall}(N_1, M_1), (N_2, M_2), L = 1.0$) and (ii) $N_3$ has enabled transitions that are not enabled in $N_2$ (cf. $\text{precision}(N_2, M_2), (N_3, M_3), L = \chi$), $N_3$ can have at most as many enabled transitions in common with $N_1$ as it has with $N_2$. However, note that $N_3$ can have fewer transitions in common with $N_1$ than it has with $N_2$ because $N_2$ has behavior that is not in $N_1$ (cf. $\text{precision}(N_1, M_1), (N_2, M_2), L = \gamma$). $\text{recall}(N_1, M_1), (N_3, M_3), L \leq 1$ because, while replaying the log, $N_1$ may have enabled transitions that are not enabled in $N_3$. Note that, although $\text{recall}(N_1, M_1), (N_2, M_2), L = 1.0$, a fraction of these enabled transition may not intersect with the enabled ones for $N_3$ because $\text{precision}(N_2, M_2), (N_3, M_3), L \neq 1.0$.

Scenario 2. In this scenario, we can infer that $\text{recall}(N_1, M_1), (N_3, M_3), L \geq \text{recall}(N_1, M_1), (N_2, M_2), L$ and $\text{precision}(N_1, M_1), (N_3, M_3), L < 1.0$. Let us first have at look at why $\text{recall}(N_1, M_1), (N_3, M_3), L \geq \text{recall}(N_1, M_1), (N_2, M_2), L$. From the value of the recall metric for $N_2$ and $N_3$ (cf. $\text{recall}(N_2, M_2), (N_3, M_3), L = 1$), we know that all transitions enabled in $N_2$ are also enabled in $N_3$ when replaying the log. Thus, when assessing the recall metric for $N_1$ and $N_3$ (i.e., $\text{recall}(N_1, M_1), (N_3, M_3), L$), we know that $N_1$ has at least as many enabled elements in common with $N_3$ as $N_2$ has with $N_3$. However, since $\text{precision}(N_2, M_2), (N_3, M_3), L \neq 1$ (i.e., some of the enabled transitions in $N_3$ are not enabled in $N_2$), it can
be that \( N_1 \) has more elements in common with \( N_3 \) than with \( N_2 \). That is why \( \text{recall}(N_1, M_1), (N_3, M_3), L) \geq \text{recall}(N_1, M_1), (N_2, M_2), L) \). Now, let us analyze why \( \text{precision}(N_1, M_1), (N_3, M_3), L) \leq 1 \). This happens because we cannot be sure if \( N_3 \) will have more or less enabled elements in common with \( N_1 \) than it has with \( N_3 \). We only know that \( N_3 \) cannot have all enabled transitions in common with \( N_1 \) because \( \text{precision}(N_1, M_1), (N_2, M_2), L) \neq 1 \) and \( \text{recall}(N_2, M_2), (N_3, M_3), L) = 1.0 \). Thus, \( \text{precision}(N_1, M_1), (N_3, M_3), L) \) can assume any value but 1.

**Scenario 3.** The \( \text{recall}(N_1, M_1), (N_2, M_2), L) \leq \text{recall}(N_2, M_2), (N_3, M_3)) \) for this scenario because all enabled transitions for \( N_3 \) are also enabled for \( N_2 \) (cf. \( \text{precision}(N_2, M_2), (N_3, M_3) = 1 \)). Consequently, \( N_1 \) cannot have more enabled transitions with \( N_1 \) than it has with \( N_2 \). However, \( N_1 \) can have fewer enabled transitions in common with \( N_3 \) than with \( N_2 \). That is why \( \text{recall}(N_1, M_1), (N_3, M_3), L) \leq \text{recall}(N_2, M_2), (N_3, M_3)) \). The \( \text{precision}(N_1, M_1), (N_3, M_3), L) \leq 1.0 \) because the fact that \( \text{precision}(N_2, M_2), (N_3, M_3), L) \) is equal to 1 does not prevent that all enabled transitions in \( N_3 \) are also enabled in \( N_1 \). Thus, \( \text{precision}(N_1, M_1), (N_3, M_3), L) \) can assume any value between 0 (inclusive) and 1 (inclusive).

**Scenario 7.** In this scenario, we know that all the enabled transitions for \( N_1 \) and \( N_3 \) while replaying the log are also enabled for \( N_2 \) because \( \text{recall}(N_1, M_1), (N_2, M_2), L) = 1.0 \) and \( \text{precision}(N_2, M_2), (N_3, M_3), L) = 1.0 \). However, since we do not know how much behavior of \( N_2 \) intersects with the behavior of \( N_1 \) and \( N_3 \), it can be that \( N_1 \) and \( N_3 \) have the same enabled transitions or that they do not have a single transition in common while replaying the log. In other words, \( \text{precision}(N_1, M_1), (N_3, M_3), L) \) and \( \text{recall}(N_1, M_1), (N_3, M_3), L) \) can assume any value between 0 (inclusive) and 1 (inclusive).

**Scenario 1.** In this scenario, the precision and recall metrics for \( N_1 \) and \( N_3 \) can have any value between 0 (inclusive) and 1 (inclusive) because the fact that \( \text{precision}(N_1, M_1), (N_2, M_2), L) \neq 1 \), \( \text{recall}(N_1, M_1), (N_2, M_2), L) \neq 1 \), \( \text{precision}(N_2, M_2), (N_3, M_3), L) \neq 1 \), and \( \text{recall}(N_2, M_2), (N_3, M_3), L) \neq 1 \) does not prevent \( N_1 \) and \( N_3 \) of having \( \text{precision}(N_1, M_1), (N_3, M_3), L) = 1 \) and \( \text{recall}(N_1, M_1), (N_2, M_2), L) = 1 \). As an illustration, just think of the situation in which the two models \( N_1 \) and \( N_3 \) are the same, but have a different behavior than the model \( N_2 \).

The results presented in this section (cf. Table 1) are useful in situations in which one needs to get a rough indication about values for the precision and recall of two models, but does not really need/want to calculate these values.

### 7. Behavioral precision/recall and process equivalence in general

If two models \( N_1 \) and \( N_2 \) have behavioral precision and recall equal to 1 with respect to a given log \( L \), does this imply that these two models are behaviorally equivalent? In other words, does this mean that all the behavior generated by \( N_1 \) can be generated by the \( N_2 \), and vice-versa? The short answer is not always. In this section we illustrate situations in which \( \text{precision}(N_1, N_2, L) = 1 \) and \( \text{recall}(N_1, N_2, L) = 1 \), but \( N_1 \) and \( N_2 \) do not capture the exact same behavior. The situations are:

- **Model does not fit the log.** In this situation, the fitness (cf. Definition 7) of at least one of the models is not equal to 1 (i.e., \( \text{fitness}(N_1, M_1), L) < 1 \) or \( \text{fitness}(N_2, M_2), L) < 1 \). Fig. 4 shows one example for this situation. Note that none of the models in Fig. 4 fit the log, but they always have the same enabled transitions when replaying the traces of the log in 4c and with the respective initial markings in 4a and 4b.
- **Model has tasks that are not in the log.** Some of the tasks of one of the models are not in the log (i.e., \( T_{N_1} \not\subseteq T_L \) or \( T_{N_2} \not\subseteq T_L \)), the models may also have different behaviors even if their behavioral precision and recall are maximal. For instance, consider the example illustrated in Fig. 5. Note that these models have the same enabled transitions when parsing the log. Furthermore, both models completely fit the log. However, these models are not behaviorally equivalent. If the log would contain any trace with the tasks C or E, the behavioral differences would have been captured by the precision and recall metrics in Definition 8.
- **The log does not express enough behavior.** Even if two models completely fit a log, all of their tasks are in this log, and their precision and recall values are maximal for this log, these models can still behave differently. The reason is that the log may not contain enough behavior such that the precision and recall metrics can capture differences between models. This situation is illustrated in Fig. 6.
Our aim with showing the three situations above is to make the reader aware of the "limitations" of the behavioral precision and recall metrics in Definition 8. It is important to realize that the process equivalence quantification captured by these metrics is always with respect to a given log. That is why it is so important that the log reflects typical behavior. This is both the strength and weakness of the approach described in this paper.

8. Application to genetic mining

Process mining aims at extracting information from event logs to capture the business process as it is being executed. Process mining is particularly useful in situations where events are recorded but there is no system enforcing people to work in a particular way. Consider for example a hospital where the diagnosis and treatment activities are recorded in the hospital information system, but where health-care professionals determine the "careflow". A variety of process mining algorithms have developed [5–7,12,21], including our approach based on genetic process mining [3,8,25].
The goal of process mining is to extract information about processes from event logs [5], e.g., a log like the one shown in Fig. 1f. The result is a model, e.g., a Petri net. Using simple approaches such as the one presented in [6] it is possible to automatically discover the Petri net shown in Fig. 1a based on the log shown in Fig. 1f. Unfortunately, existing approaches for mining the process perspective have problems dealing with issues such as duplicate activities, hidden activities, non-free-choice constructs, noise, and incompleteness. The problem with duplicate activities occurs when the same activity can occur at multiple places in the process. This is a problem because it is no longer clear to which activity some event refers. The problem with hidden activities is that essential routing decisions are not logged but impact the routing of cases. Non-free-choice constructs are problematic because it is not possible to separate choice from synchronization. We consider two sources of noise: (1) incorrectly logged events (i.e., the log does not reflect reality) or (2) exceptions (i.e., sequences of events corresponding to “abnormal behavior”). Clearly noise is difficult to handle. The problem of incompleteness is that for many processes it is not realistic to assume that all possible behavior is contained in the log. The goal of genetic process mining [3,25] is to overcome these problems.

Genetic process mining is the application of genetic algorithms to process mining. Genetic algorithms are an adaptive search technique that mimics the process of evolution in biological systems [18]. Fig. 7 depicts the main steps of our genetic approach. Once the log is read (Step I), the algorithm randomly builds an initial population with a number of individuals (Step II). Every individual is a process model containing all the tasks in the log. After the initial population is generated, the genetic algorithms calculates the fitness of its individuals (Step III). In our case, the fitness of an individual is based on its ability to correctly replay the behavior in the log. Populations evolve by selecting the fittest individuals and generating new individuals using genetic operators such as crossover (combining parts of two or more individuals) and mutation (random modification of an individual) (Step V). This process continues until the “best fitting” model is discovered or other stop criteria (like maximum number of generations) are met (Step IV). The “best fitting” discovered model that the genetic algorithm targets at is complete (can parse all the traces in the log) and precise (does not allow for the parsing of too much extra behavior that cannot be derived from the behavior observed in the log). Fig. 8 shows a screenshot of the Genetic algorithm plug-in in the ProM framework. The ProM framework can be downloaded from www.processmining.org and supports the development of plug-ins to mine event logs.

One of the problems of doing research in this area is that it is difficult to judge the result. For testing our approach we do not only take real-life logs (e.g., from SAP, Staffware or FLOWer) but also generate logs from

![Fig. 6. Example why the log should contain enough behavior such that the behavioral precision and recall metrics could capture differences in the models.](image-url)
known models. In the later case, we can compare the “original model” (i.e., the initial model used to generate logs from) with the “discovered model” (i.e., the one obtained using process mining). Experience shows that typically the initial model and the discovered model are not identical: the structure is different and/or behaviorally there are also differences. In the genetic algorithm setting, these differences can occur because, for instance, there is more than one complete and precise model that can be discovered for a given log, or simply because the genetic algorithm did not find a very precise model. Hence, to measure the quality of mining result we need to consider the issues raised in this paper. Moreover, in this setting there are event logs that can serve as a basis for comparison.

Figs. 8 and 9 show some results obtained by applying the genetic algorithm to the log $L = \{(A, B, C, E, F, H, I), 141\}, \{(A, B, D, E, G, H, I), 159\}$). Fig. 8 shows the result of one run of the genetic algorithm in ProM. Fig. 9 shows the precision and recall values over 50 runs. The results show that the genetic algorithm found a model with the same structure and behavior as the original model for most of the runs. The Petri net representation of the mined model for these runs looks like the one in Fig. 8c. Note that this Petri net shows a non-free-choice construct involving the tasks $C, D, E, F$ and $G$. In other words, the choice of whether executing $F$ or $G$ is not done after executing $E$, but when executing $C$ or $D$. Note that the behavior in the log tell us that the task $F$ is only executed when the task $C$ has been executed. A similar observation.

---

### Fig. 7. Main steps of our genetic process mining algorithm.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Read event log</td>
</tr>
<tr>
<td>II</td>
<td>Build the initial population</td>
</tr>
<tr>
<td>III</td>
<td>Calculate fitness of the individuals in the population</td>
</tr>
<tr>
<td>IV</td>
<td>Stop and return the fittest individuals?</td>
</tr>
<tr>
<td>V</td>
<td>Create next population — use elitism and genetic operators</td>
</tr>
</tbody>
</table>

---

### Fig. 8. Screenshot of the ProM framework showing three aspects of the Genetic algorithm plug-in: (a) the configuration window allows for the setting of parameters, (b) the discovered model in terms of the internal representation, and (c) the corresponding Petri net.
holds for tasks \( G \) and \( B \). Fig. 9 shows that the genetic algorithm did pretty well for 88% (44/50) of the runs. However, for six runs (runs 6, 8, 19, 22, 41 and 45), the genetic algorithm mined a different model. In runs 6 and 8, the genetic algorithm found a model that is a substructure of the original model (\( \text{precision}^S = 1 \) and \( \text{recall}^S = 0.875 \)) and allows for more behavior than the original model (\( \text{precision} = 0.933 \) and \( \text{recall} = 1 \)). The mined model for both runs was exactly the same. The Petri net representation for this mined model is shown in Fig. 10a. Fig. 10b shows the mined model for runs 19, 22, 41 and 45. Note that, as indicated by the metrics, this mined model has the same behavior as the original one (\( \text{precision} = 1 \) and \( \text{recall} = 1 \)) and a different structure (\( \text{precision}^S = 0.875 \) and \( \text{recall}^S = 0.875 \)). The use of the structural and behavioral precision/recall metrics allows us to measure how well the genetic algorithm is doing when mining process models. The metrics make it possible to analyze the similarity of the original and mined models in terms of possible behavior and thus quantify their differences.

9. Conclusion

This paper has presented a novel approach to compare process models. Existing approaches typically do not quantify equivalence, i.e., models are equivalent or not. However, for many practical applications such an approach is not very useful, because in most real-life settings we want to distinguish between marginally different processes and completely different processes. We have proposed and implemented notions of fitness, precision, and recall in the context of the ProM framework. The key differentiator is that these notions take an event log with typical execution sequences as a starting point. This allows us to overcome many of the problems associated with approaches directly comparing processes at the model level. Although our approach is
based on Petri nets, it can be applied to other models with executable semantics, e.g., formalizations of EPCs, BPMN, or UML activity diagrams.

We have applied the approach in the context of process mining. However, the notions of precision and recall can be applied in a wide variety of situations, e.g., to measure the difference between an organization-specific process model and a reference model, to select a web service that fits best based on some description (e.g., PIPs or abstract BPEL), or to compare an existing process model with some redesign. In our future work, we would like to explore more of these applications, e.g., comparing clinical guidelines.

References


Dr. Ana Karla Alves de Medeiros is a research fellow of Information Systems at the Department of Technology Management at the Eindhoven University of Technology (TU/e). Her research interests include process mining, business process management, data mining, genetic algorithms, and Petri nets. She has published several articles on these topics.

Prof.dr.ir. Wil van der Aalst is a full professor of Information Systems at the Technische Universiteit Eindhoven (TU/e) having a position in both the Department of Mathematics and Computer Science and the department of Technology Management. Currently he is also an adjunct professor at Queensland University of Technology (QUT) working within the BPM group there. His research interests include workflow management, process mining, Petri nets, business process management, process modeling, and process analysis. Wil van der Aalst has published more than 70 journal papers, 12 books (as author or editor), 200 refereed conference publications, and 20 book chapters. Many of his papers are highly cited and his ideas have influenced researchers, software developers, and standardization committees working on process support. He has been a co-chair of many conferences including the International Conference on Cooperative Information Systems, the International conference on the Application and Theory of Petri Nets, and the Business Process Management conference, and is an editor/member of the editorial board of several journals, including the Business Process Management Journal, the International Journal of Business Process Integration and Management, the International Journal on Enterprise Modelling and Information Systems Architectures, and Computers in Industry.

Dr. A.J.M.M. (Ton) Weijters is an associate professor at the Technology Management Department of the Eindhoven University of Technology (TUE), and member of the BETA research group. His current research focuses on process mining (i.e. to extract knowledge from event logs recorded by an information system to analyze the underlying business processes). He is the author of many scientific publications in the mentioned research field. His papers appeared in journals such as Data Mining and Knowledge Discovering, Information Systems, IEEE Transactions on Knowledge and Data Engineering, Artificial Intelligence in Medicine, European Journal of Operational Research, Computers in Industry, Knowledge-Based Systems, AI-Review, etc.