Behavioural Quotients for Precision and Recall in Process Mining

Artem Polyvyanyy, Andreas Solti, Matthias Weidlich, Claudio Di Ciccio, and Jan Mendling

Abstract—The comparison of the languages of software systems, i.e., their behaviours in terms of specified executions, is a prerequisite for many applications, reaching from system validation through management of a system’s evolution to conformance checking of observed and expected behaviour. If two systems are not language-equivalent, the quantification of behavioural differences enables conclusions on the extent of deviation. Such quantifications are commonly done in a relative manner: A quotient is defined over some measure of two languages, which have potentially been derived via algebraic operations. However, there exists no systematic approach for defining quotients and it is unclear which measures enable meaningful comparisons of systems having infinite behaviours.

This paper introduces a framework for defining language quotients. We instantiate the framework with cardinality-based and entropy-based measures to handle finite and infinite behaviours, and prove important properties of the quotients. We demonstrate application of quotients in the field of process mining to capture precision and recall between a log of recorded system executions and a model of expected system executions. An experimental evaluation of the quotients using our open-source implementation demonstrates their feasibility and indicates that the quotients enable a monotonic assessment, unlike state-of-the-art measures in process mining.

1 INTRODUCTION

There are various use cases in software engineering in which the behaviour of systems has to be compared. Examples include the implementation of a specification (are all specified executions realised?) [1]; management of a system’s evolution (which functionality has been added in a revision?) [2], and conformance checking (are observed executions in line with a specification?) [3]. Such comparisons of systems are typically done in a relative manner, rather than in absolute terms. The behaviour of one system is put into perspective (or normalised), by defining a quotient of some property of its behaviour over the property of the other behaviour. For instance, the quotient of the implemented behaviour over the specified behaviour can be used to express a measure of the progress of the realisation of a specification.

The behaviour of software systems can commonly be described in terms of all possible executions, where an execution describes a sequence of actions of the system [4]. An action represents an atomic unit of functionality, which, depending on the type of system, may be as fine-grained as a single processor instruction or as coarse-grained as a Web service call. The behaviour of a system, therefore, can be represented as a language that defines a set of words over its actions, each word being one possible execution (also referred to as a run, trace, or process) of the system.

Behavioural comparison based on quotients of languages, however, faces two major challenges. First and foremost, in order to allow for a reasonable interpretation, quotients shall satisfy particular properties. One such property is monotonicity: When increasing the amount of behaviour in the numerator of a quotient while leaving the denominator unchanged, the quotient shall increase as well. Existing quotients as proposed, e.g., in the field of process mining [5] to compare modelled and recorded behaviour, however, do not satisfy such well-motivated properties [6].

A second challenge relates to the definition of quotients in the presence of systems that do not terminate, for which the behaviour corresponds to a language with infinitely many words. In that case, quotients defined over standard properties of languages, such as their cardinality, are not meaningful for behavioural comparison. In the past, this issue has been avoided by employing behavioural abstractions that capture a language by means of pair-wise relations over its actions [7]. Yet, such an abstraction does not capture the complete language semantics of a system [8] and, thus, introduces a bias in behavioural comparison.

Against this background, we address the problem of how to define meaningful quotients for behavioural comparison over finite and infinite languages? Concretely, this article contributes:

1. A framework for the definition of language quotients that guarantees several generic properties.

2. The definition of two quotients as instantiations of the framework, grounded in the cardinality of languages (for finite languages) and the entropy of automata to capture languages of systems (for finite and infinite languages).

3. The application of quotients for process mining to capture the precision and recall of a log of recorded executions and a model of presumed executions.

4. A publicly available implementation of the proposed precision and recall quotients.

5. An evaluation that demonstrates the feasibility of the proposed precision and recall quotients and compares them against state-of-the-art measures in process mining.
The remainder of this article is structured as follows. Section 2 describes the background of the research problem that we address. Section 3 introduces formal preliminaries in terms of languages and automata. The framework for the definition of language quotients is introduced in Section 4, which also includes a discussion of formal properties of such quotients and two instantiations. In Section 5, we present notions of precision and recall for process mining based on language quotients. These notions are evaluated in a series of experiments using real-world data in Section 6. Finally, Section 7 discusses our contributions in the light of related work, before we conclude in Section 8.

2 BACKGROUND ON BEHAVIOURAL COMPARISON

Once the behaviour of (existing or planned) software systems is captured by languages over their actions, insights into their differences and commonalities are obtained by comparing the respective languages. This is commonly done based on a measure that quantifies a property of a language, such as its cardinality, i.e., the number of words defined by the language. Based thereon, a ratio of these properties enables relative comparison of two languages and, thus, systems: The behaviour of one system is put into perspective (or normalised) relative to some base behaviour. We refer to such a ratio as a language quotient:

$$\text{language quotient} := \frac{\text{measure(language_1)}}{\text{measure(language_2)}}.$$ 

For illustration purposes, consider the scenario of a user logging into some application, which is based on the actions listed in Fig. 1a, such as creating a login session or conducting the actual authentication. Specific realisations of this scenario are given as finite automata in Fig. 1b - Fig. 1d. Albeit similar, the systems, $S_1$, $S_2$, and $S_3$, define a different language over the actions, denoted by $L(S_1)$, $L(S_2)$, and $L(S_3)$, respectively. In fact, the languages are in a subset relation, $L(S_3) \subset L(S_2) \subset L(S_1)$. Furthermore, Fig. 1e depicts three logs, $L_1$, $L_2$, and $L_3$, which represent recorded executions of actual login processes. Each log $L$ is a multiset of sequences over actions and, thus, also induces a language $L(L)$. The latter contains all words that occur at least once in the log.

The three automata may represent (i) different systems, (ii) different versions of the same system, or (iii) system specifications and their implementations. In any case, it useful to quantify to which extent the automata describe the same behaviour—this answers the question in how far (i) different systems provide the same functionality; (ii) the functionality of a system has changed over several versions; and (iii) a specification has been implemented already.

Considering also the logs, we note that similar questions emerge in the field of process mining [5], which targets the analysis of process-oriented information systems based on recorded executions of a business process. Given a model of a system and a log, process mining strives for quantifying the share of recorded behaviour that is in line with the model (fitness or recall of the log) or the share of modelled behaviour that is actually observed (precision of the model).

To address the above use cases, we may consider the question by how much one system extends the behaviour of another system? For systems $S_x$ and $S_y$, such that $L(S_y) \subseteq L(S_x)$, we may answer this question with a quotient defined using language cardinality as a measurement function:

$$\text{language extension}(S_x, S_y) := \frac{|L(S_x)|}{|L(S_y)|}.$$ 

A slightly different way to assess the relation between these systems, however, is the question of how much of the behaviour of one system is covered by another system? To this end, set-algebraic operations over languages may be incorporated in the definition of a quotient, as in the following definition:

$$\text{language coverage}(S_x, S_y) := \frac{|L(S_x) \cap L(S_y)|}{|L(S_x)|}.$$ 

The above quotients of language extension and coverage provide a straightforward means for behavioural comparison of systems, models of systems, and logs. Yet, they are useful only if the applied measurement function provides a meaningful mapping of a language into a numerical domain. For the cardinality function used above, we argue that this is the case solely for finite languages. For languages that define an infinite number of words, the numerator or denominator of a quotient may become infinity. Leaving aside the obvious definitional issues, any definition of a value for such a quotient would not only be arbitrary, but would also result in a single value for all infinite languages, regardless of their characteristics.

Considering our examples, we may compute the language extension using cardinality as a measure for the logs $L_1$, and $L_2$, capturing that $L(L_2) contains twice as many words as $L(L_1)$. However, language extension based on cardinality is not meaningful for any pair of languages of systems $S_1$, $S_2$, $S_3$.
and $S_2$, since $L(S_1)$ and $L(S_2)$ are infinite. In the same vein, computing the language coverage of a system model and a log, to assess the fitness of the log or the precision of the model, respectively, is not meaningful for the systems $S_1$ and $S_2$, and any of the logs.

Beyond the challenge to cope with infinite language, we note that quotients shall satisfy particular properties. As mentioned, the languages of the three automata are in a subset relation, $L(S_3) ⊂ L(S_2) ⊂ L(S_1)$. These subsumption relations, for instance, shall be reflected in the respective quotients of language extension: A quotient defined over the smallest language $L(S_1)$ and the largest language $L(S_1)$ should yield a smaller value than a quotient over $L(S_3)$ and the second largest language $L(S_2)$. Since language $L(S_1)$ contains $L(S_2)$ and is strictly larger, the additional behaviour shall lower the value of the respective ratio.

Desired properties of quotients such as those discussed above translate into requirements on the measurement functions. As we will discuss in more detail in the remainder, monotonicity of the measurement function and the existence of a supremum that bounds the measurement space are of particular relevance in this context. The former means that adding behaviour to a system strictly increases (or strictly decreases) the measure, whereas the latter implies that a specific value is defined to the empty behaviour.

Many measures for behavioural comparison proposed in the literature neglect such properties, raising debates on how to interpret the obtained results. In the domain of process mining, e.g., it was recently shown that none of the existing measures to assess the precision of a model against an event log satisfies a set of well-motivated essential properties [6].

Against this background, the fundamental challenge of using quotients for behavioural comparison is to come up with a framework for their meaningful definition. That is, the framework shall provide guarantees on the quotients to satisfy a collection of desirable properties.

3 Preliminaries

This section presents formal notions used to support the discussions in the subsequent sections.

3.1 Multisets, Sequences, and Languages

A multiset, or a bag, is a generalization of a set, i.e., a collection that can contain multiple instances of the same element. By $B(A)$, we denote the set of all finite multisets over some set $A$. For some multiset $B ∈ B(A)$, $B(a)$ denotes the multiplicity of element $a$ in $B$. For example, $B_1 := []$, $B_2 := [b, a, a]$, and $B_3 := [a^2, b]$ are multisets over the set $\{a, b\}$. Multiset $B_1$ is empty, i.e., it contains no elements, whereas $B_2(a) = 2 = B_2(a)$, $B_2(b) = 1 = B_3(b)$, and, hence, it holds that $B_2 = B_3$.

The standard set operations have been extended to deal with multisets as follows. If element $a$ is a member of multiset $B$, this is denoted by $a ∈ B$; otherwise, one writes $a ∉ B$. The union of two multisets $C$ and $D$, denoted by $C ∪ D$, is the multiset that contains all elements of $C$ and $D$ such that the multiplicity of an element in the resulting multiset is equal to the sum of multiplicities of this element in $C$ and $D$. For example, $[b] ∪ B_2 = [a^2, b^2]$. Also note that $\mathcal{L}_2$ in Fig. 1e is the union of $\mathcal{L}_1$ and the multiset of three sequences with two instances of sequence $(a, f, c)$; more info on sequences is provided below. The difference of two multisets $C$ and $D$, denoted by $C − D$, is the multiset that for each element $x ∈ C$ contains $\max(0, C(x) − D(x))$ occurrences of $x$. For example, it holds that $B_2 ∖ B_3 = B_1$, and $B_3 ∖ [b] = [a, a]$. Given a multiset $B ∈ B(A)$ over some set $A$, by $Set(B)$ we refer to the set that contains all and only elements in $B$, i.e., $Set(B) := \{b ∈ A | b ∈ B\}$.

A sequence is an ordered list of elements. By $σ := (a_1, a_2, \ldots, a_n) ∈ A^*$, we denote a sequence over some set $A$ of length $n ∈ N_0$, $a_i ∈ A$, $i ∈ [1..n]$, where $[j..k] := \{x ∈ N_0 | j ≤ x ≤ k\}$, $j, k ∈ N_0$. By $|σ| := n$, we denote the length of the sequence. By $σ[i]$, $i ∈ [1..n]$, we refer to the $i$-th element of $σ$, i.e., $σ[i] = a_i$. Given a sequence $σ$ and a set $K$, $σ\vert K$, we denote a sequence obtained from $σ$ by deleting all elements of $σ$ that are not members of $K$ without changing the order of the remaining elements. For example, it holds that $(a, b, c, a, c)\vert_{\{b, c\}} = (b, c)$. Given two sequences $σ$ and $σ'$, by $σ ◦ σ'$, we denote the concatenation of $σ$ and $σ'$, i.e., the sequence obtained by appending $σ'$ to the end of $σ$. For example, $(a, b, a) ◦ (b, a) = (a, b, a, b, a)$, where $(.)$ is the empty sequence. For two sets of sequences $X_1$ and $X_2$ over $A$, $X_1 ∩ X_2 := \{σ ∈ A^* | ∃σ ∈ X_1, ∃σ_2 ∈ X_2 : σ = σ_1 ∙ σ_2\}$. By $suffix(σ, i)$, $i ∈ N$, we denote the suffix of $σ$ starting from and including position $i$. For example, $L_1$ in Fig. 1e contains sequences $σ_1 := (a, b, c, e)$ and $σ_2 := (a, b, c, b, c, d, e)$. It holds that $suffix(σ_1, 3) = (d, e)$ and $suffix(σ_2, 6) = (d, e)$. If $σ := (a_1, a_2, \ldots, a_n) ∈ A^*$ is a sequence over $A$ and $f$ is a function over $A$, then $f(σ) := (f(a_1), f(a_2), \ldots, f(a_n))$. Similarly, if $A' ⊆ A$, then $f(A') := \{f(a) | a ∈ A'\}$.

An alphabet is any nonempty finite set. The elements of an alphabet are its labels, or symbols. By $Σ$, we denote a universe of symbols. For example, Fig. 1a specifies alphabet $Σ := \{a, b, c, d, e, f\}$. A word over an alphabet is a finite sequence of symbols from the alphabet. The word of length zero is called the empty word and is denoted by $ε$. A (formal) language over an alphabet $Σ$ is a set of words over $Σ$.

3.2 Finite Automata

We deal with a common notion of a finite automaton [9, 10].

Definition 3.1 (Nondeterministic finite automaton). A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, Λ, δ, q_0, A)$, where:
- $Q$ is a finite nonempty set of states,
- $Λ ⊆ Σ$ is a set of labels, such that $Q$ and $Σ$ are disjoint,
- $δ : Λ × (Q ∪ \{τ\}) → P(Q)$ is the transition function, where $τ ∈ Σ$ is a special label such that $τ ∉ Λ ∪ A$,
- $q_0 ∈ Q$ is the start state, and
- $A ⊆ Q$ is the set of accept states.\(^1\)

An NFA induces a set of computations.

Definition 3.2 (Computation). A computation of an NFA $(Q, Λ, δ, q_0, A)$ is either the empty word or a word $σ := (s_1, s_2, \ldots, s_n), n ∈ N$, where every $s_i$ is a member of $Λ ∪ \{τ\}$, $i ∈ [1..n]$, and there exists a sequence of states $q := (q_0, q_1, \ldots, q_n)$, where every $q_j$ is a member of $Q$.

\(^1\)By $N$ and $N_0$, we denote the set of all natural numbers excluding and including zero, respectively.

\(^2\)Given a set $A$, by $P(A)$, we denote the powerset of $A$. 
Section 1

For illustration of a quotient, a language quotient is defined based on $H$. We say that $s$ leads to $q_n$. By convention, the empty word always leads to the start state. An NFA $B := (Q, \Lambda, \delta, q_0, A)$ accepts a word $s$ iff $s$ is a composition of $B$ that leads to an accept state $q$ of $B$, i.e., $q \in A$.

Definition 3.3 (Language of an NFA). The language of an NFA $B := (Q, \Lambda, \delta, q_0, A)$, is denoted by $L(B)$, and is the set of words that $B$ accepts, i.e., $L(B) := \{s \in \Lambda^* \mid \exists r \in (\Lambda \cup \{\tau\})^* : (B \text{ accepts } r) \land (s = r|_\Lambda)\}$.

We say that $B$ recognizes $L(B)$.

In an NFA, the transition function takes a state and an input label to produce the set of possible next states. In a DFA, the transition function takes a state and an input label and produces the next state.

Definition 3.4 (Deterministic finite automaton). A deterministic finite automaton (DFA) is an NFA $(Q, \Lambda, \delta, q_0, A)$ with the property that for every state $q \in Q$ and for every label $s \in \Lambda$, $\delta(q, s)$ holds that $|\delta(q, s)| \leq 1$.

An NFA $(Q, \Lambda, \delta, q_0, A)$ is ergodic if its underlying graph is strongly irreducible, i.e., for all $(x, y) \in Q \times Q$ there exists a sequence of states $(q_1, \ldots, q_n) \in Q^n$, $n \in \mathbb{N}$, for which it holds that for every $k \in [1..n-1]$ there exists $\lambda \in \Lambda \cup \{\tau\}$ such that $\delta(q_k, \lambda) = q_{k+1}$, $q_1 = x$, and $q_n = y$.

A language $L \subseteq \Xi^*$ is regular iff it is the language of an NFA. A language $L \subseteq \Xi^*$ is irreducible if, given two words $w_1$ and $w_2$ in $L$, there exists a word $w \in \Xi^*$ such that the concatenation $w_1 \circ w \circ w_2$ is in $L$. A regular language $L$ is irreducible iff it is the language of an ergodic NFA, cf. [11].

An NFA $B := (Q, \Lambda, \delta, q_0, A)$ is $\tau$-free iff for all $q \in Q$ it holds that $\delta(q, \tau) = \emptyset$. Given an NFA $B$, one can always construct a $\tau$-free DFA $B'$ that recognizes the language of $B$.

Theorem 3.5 (Equivalence of NFAs and $\tau$-free DFAs). A language $L$ is the language of an NFA iff the language of some $\tau$-free DFA.

The proof of Theorem 3.5 is in [12], cf. Theorem 2.22. Hence, in what follows, we only consider $\tau$-free DFAs.

For illustration, consider the automaton in Fig. 1c. It is a $\tau$-free NFA $S_2 := (Q, \Lambda, \delta, q_0, A)$, defined by set of states $Q := \{A, B, C, D, E\}$, set of labels $\Lambda := \{a, b, c, d, e\}$, transition function $\delta := \{((a, a), \{b\}), ((B, b), \{C, D\}), ((C, c), \{B\}), ((D, d), \{E\}), ((E, e), \{A\})\}$, start state $q_0 := A$, and accept states $A := \{A\}$; note for the use of standard graphical notation for depicting automata. Fig. 2 shows a $\tau$-free DFA that is equivalent with the automaton in Fig. 1c, i.e., has the same language as the automaton in Fig. 1c.

4 A FRAMEWORK FOR LANGUAGE QUOTIENTS

This section introduces a framework for behavioural comparison of systems using language quotients. As detailed in Section 4.1, a language quotient is defined based on a measurement function over languages of systems. In Section 4.2, we demonstrate that the proposed quotients satisfy interesting properties for behavioural comparison of systems. Finally, in Section 4.3, we propose two measurement functions for instantiation of language quotients, one based on the cardinality of a language and one based on the topological entropy of automata that recognize a language.

4.1 Framework Definition

Behavioural comparison of systems is usually carried out based on properties of their languages. A property of a language can be captured by a measure $m$, which is a (set) function from the set of all languages over $\Xi$ to non-negative real numbers:

$$m : \wp(\Xi^*) \to \mathbb{R}_0^+.$$ 

A measure can be subject to these two constraints:

- A measure can be monotonic. A measure $m$ is (strictly monotonically) increasing iff for all $U \subseteq \Xi^*$ and $V \subseteq \Xi^*$ such that $U \subset V$, it holds that $m(U) < m(V)$.

- A measure can map the infimum of its domain to the infimum of its codomain. In this line, we define that a measure $m$ starts at zero iff $m(\emptyset) = 0$.

We say that a measure over languages is a language measure if it is increasing and starts at zero.

Using the notion of language measure, we define a language quotient as a means to set properties of languages into relation, as follows:

Definition 4.1 (Language quotient). Given two languages $L_1$ and $L_2$, and a language measure $m$, the language quotient of $L_1$ over $L_2$ induced by $m$ is the fraction of the measure of $L_1$ over the measure of $L_2$, i.e.,

$$\text{quotient}_m(L_1, L_2) := \frac{m(L_1)}{m(L_2)}.$$ 

Nomen est omen, a language quotient is defined over languages, not systems. The rationale behind this formalisation is that the framework of language quotients, once instantiated with a specific measure, may be applied for diverse algebraic operations defined over languages; examples include quotients that are defined over the intersection, union, or difference of languages. The reader can refer to the notion of language coverage in Section 1 for illustration of a quotient defined over intersection of languages. In Section 5.1, we provide further examples of quotients over the intersection of languages that are useful in the context of process mining.

The fact that language quotients are parameterized using language measures ensures their interesting properties, as detailed next.

4.2 Properties of Language Quotients

Language quotients enjoy some interesting properties that arise from the properties of language measures they are defined upon. First, if languages are in the strict subset relation, one can compare two quotients with the same numerator as follows.

$^3$By $\mathbb{R}_0^+$, we denote the set of all non-negative real numbers.
Fig. 3: Schematic representation of language quotient properties: Lemma 4.2 (top row) and Lemma 4.3 (bottom row).

**Lemma 4.2** (Fixed numerator quotients).
If $L_1, L_2, L_3 \subseteq \Sigma^*$ are languages such that $L_1$ is nonempty, $L_1 \subseteq L_2$, and $L_2 \subseteq L_3$, then it holds that $\text{quotient}_{m}(L_1, L_3) < \text{quotient}_{m}(L_1, L_2)$, where $m$ is a language measure.

**Proof.** Let us assume that $L_1 \neq \emptyset$, $L_1 \subseteq L_2$, and $L_2 \subseteq L_3$, but it holds that $\text{quotient}_{m}(L_1, L_3) \geq \text{quotient}_{m}(L_1, L_2)$. Because $m$ starts at zero and is increasing, it holds that $0 < m(L_2) < m(L_3)$. Because $m(L_1) > 0$, we reach a contradiction.

The statement of Lemma 4.2 is shown schematically in Fig. 3 (top row). For example, if $L_2$ and $L_3$ are languages of two systems that extend the behaviour of a system that recognizes language $L_1$, then, using the quotients, one can conclude that the system that recognizes $L_3$ extends the behaviour of the system that recognizes $L_1$ more than does the system that recognizes $L_2$. The difference between the extension behaviours is captured by $\text{quotient}_{m}(L_1, L_3) - \text{quotient}_{m}(L_1, L_2)$. The meaning of the difference depends on the meaning of language measure $m$ used to instantiate the quotients. For example, if $m$ is the cardinality of a language, then the difference stands for the fraction of the behaviour with which $L_3$ extends $L_1$ more than does $L_2$.

Second, if languages are, again, in the strict subset relation, one can compare two quotients with the same denominator as follows.

**Lemma 4.3** (Fixed denominator quotients).
If $L_1, L_2, L_3 \subseteq \Sigma^*$ are languages such that $L_1 \subseteq L_2$ and $L_2 \subseteq L_3$, then it holds that $\text{quotient}_{m}(L_1, L_3) < \text{quotient}_{m}(L_2, L_3)$, where $m$ is a language measure.

**Proof.** Let us assume that $L_1 \subseteq L_2$ and $L_2 \subseteq L_3$, but it holds that $\text{quotient}_{m}(L_2, L_3) \leq \text{quotient}_{m}(L_1, L_3)$. Because $m$ starts at zero and is increasing, it holds that $0 < m(L_1) < m(L_2)$. Because $m(L_3) > 0$, we reach a contradiction.

The statement of Lemma 4.3 is visualized schematically in Fig. 3 (bottom row). For example, if $L_3$ is a language of a specification of a system, and $L_1$ and $L_2$ are languages of its two implementations, then, based on the quotients, one can conclude that the implementation that recognizes $L_2$ is more complete than the implementation that recognizes $L_1$. In other words, the implementation that recognizes $L_2$ has a better coverage of the specification than the implementation that recognizes $L_1$. The extent to which the implementation that recognizes $L_2$ is more complete can be quantified by $\text{quotient}_{m}(L_2, L_3) - \text{quotient}_{m}(L_1, L_3)$. The meaning of the difference, again, depends on the meaning of language measure $m$ used to instantiate the quotients.

Finally, if one fixes the numerator, like in the case for comparing the amounts to which various systems extend a given behaviour, the quotients are bounded below.

**Corollary 4.4** (Fixed numerator quotients).
If $L_1, L_2 \subseteq \Sigma^*$ are languages such that $L_1 \subsetneq L_2$, then it holds that $\text{quotient}_{m}(L_1, L_2) < \text{quotient}_{m}(L_1, L_3)$, where $m$ is a language measure.

Corollary 4.4 follows immediately from Lemma 4.2, as it holds that $L_1 \subsetneq L_2$ and $L_2 \subseteq \Sigma^*$.

### 4.3 Framework Instantiations

This section proposes instantiations of the language quotient defined in Definition 4.1 by two specific measurement functions. The first one is based on the cardinality of a language, whereas the other one is grounded in the notion of topological entropy. When applied for the languages from the corresponding language classes, the instantiated quotients enjoy the properties proposed in 4.2.

**Cardinality quotient.** A language $L$ is a set of words. As such, its cardinality, denoted by $|L|$, i.e., the cardinality of set $L$, is a property that can serve as the basis for behavioural comparison. Clearly, cardinality is a language measure, i.e., it is increasing and starts at zero. By defining a language quotient based on this measure, we obtain the cardinality quotient:

**Definition 4.5** (Cardinality quotient).
Given two languages $L_1$ and $L_2$, the cardinality quotient of $L_1$ over $L_2$ is the fraction of the cardinality of $L_1$ over the cardinality of $L_2$, i.e.,

$$\text{quotient}_{\text{card}}(L_1, L_2) := \frac{|L_1|}{|L_2|}.$$  

Intuitively, the cardinality quotient captures the ratio of the sizes of two languages. However, it is well-defined only for $L_2 \neq \emptyset$. While this is merely a definitional issue that may be addressed explicitly (e.g., defining $\text{quotient}_{\text{card}}(L_1, L_2) := 0$ if $L_2 = \emptyset$), a more severe problem is the computation of the quotient for infinite languages.

Given an alphabet, finite by definition, a regular language may define a countably infinite set of words [13]. For example, the cardinality of an irreducible regular language is infinity. Again, one may address the resulting definitional issues explicitly, e.g., by adopting that a constant divided by infinity is equal to zero and that infinity divided by infinity is equal to one. However, any such convention is not useful for behavioural comparison in the context of regular languages. For instance, the language extension and language coverage, see Section 1, would be equal to one for any pair of ergodic automata, such as those given in Fig. 1b and Fig. 1c. We thus conclude that cardinality quotients provide a suitable means for behavioural comparison solely for finite languages.

**Eigenvalue quotient.** To obtain language quotients that are useful for comparing infinite languages, we instantiate them with a measure based on the topological entropy. Intuitively, the topological entropy of a language captures the increase in variability of the words of the language as their length goes to infinity.
Given a language $L$, let $C_n(L), n \in \mathbb{N}_0$ be the set of all the words in $L$ of length $n$, i.e., $C_n(L) := \{x \in L(B) \mid |x| = n\}$. Then, the topological entropy of $L$ is defined as follows, refer to [11, 14] for details:

$$ent(L) := \lim_{n \to \infty} \sup_{n \to \infty} \frac{\log |C_n(L)|}{n}$$

Topological entropy characterises complexity of a language and is closely related to properties of DFAs that recognize this language. In particular, the topological entropy of an automaton is equal to the topological entropy of the language that it recognises [11]. That is, for a DFA $B := (Q, \Lambda, \delta, q_0, A)$, with $C_n(B), n \in \mathbb{N}_0$, as the set of all the words in $L(B)$ of length $n$, i.e., $C_n(B) := \{x \in L(B) \mid |x| = n\}$, it holds that:

$$ent(L(B)) = ent(B) := \lim_{n \to \infty} \sup_{n \to \infty} \frac{\log |C_n(B)|}{n}$$

The topological entropy of a DFA, and thus of its language, is further related to the structure of the automaton. Below, we shall deal with square non-negative matrices $G := \{g_{ij}\}, i, j \in [1..n], n \in \mathbb{N}_0$, i.e., $g_{ij} \geq 0$ for all $i, j \in [1..n]$. The adjacency matrix of a DFA $(Q, \Lambda, \delta, q_0, A)$, with $Q := \{q_0, q_1, ..., q_n\}$, is a square matrix $G := \{g_{ij}\}, i, j \in [1..|Q|]$, such that $g_{ij} := \{|((q_i, \lambda), q_j)\in \delta \mid \lambda \in \Lambda\}$ for all $i, j \in [1..|Q|]$.

The topological entropy of a DFA $B$, i.e., $ent(B)$, is given by the logarithm of the Perron-Frobenius eigenvalue of its adjacency matrix, which is a unique largest real eigenvalue of the adjacency matrix of $B$ [11]. Note that an adjacency matrix of an ergodic DFA $B'$ has an eigenvalue $r$ such that $r$ is real, $r > 0$, and $r \geq |\lambda|$ for any eigenvalue $\lambda$ of the adjacency matrix of $B'$, refer to Theorem 1.5 in [15]. The relation between the entropy of a language and the entropy of an ergodic DFA recognizing this language, as outlined above, is important for computational reasons. That is, it provides us with a straightforward approach to compute the entropy of a language, via the Perron-Frobenius theory.

Importantly, topological entropy is a monotonic measure over languages. Let $x, y \in \Sigma^*$ be two words. If there exist $u, v \in \Sigma^*$ such that $x = u \circ y \circ v$, then $y$ is a sub-word of $x$, denoted by $y \subset x$. Let $L \subseteq \Sigma^*$ be a language and let $K$ be a nonempty set of words (or sub-words) of $L$. By $L^K$, we denote the set $\{x \in L \mid \forall y \in K : (y \subset x)\}$, i.e., the language obtained from $L$ by forbidding all the words in $K$.

**Theorem 4.6 (Monotonicity of entropy, cf. Theorem 1 in [11]).**

If $L \subseteq \Sigma^*$ is an irreducible regular language and $K$ is a nonempty set of sub-words of some words in $L$, then it holds that $ent(L^K) < ent(L)$.

Because of Theorem 4.6, topological entropy over irreducible regular languages is an increasing measure. However, it does not start at zero. Indeed, $ent(\emptyset)$ is undefined, because the eigenvalue of a zero matrix is equal to zero; here, we assume that the adjacency matrix of an ergodic DFA that induces the empty language is the zero square matrix of order one.

By $\text{eig}(L)$, where $L$ is an irreducible regular language, we denote the Perron-Frobenius eigenvalue of the adjacency matrix of an ergodic DFA that recognizes $L$, or the eigenvalue measure of $L$. We also say that $\text{eig}(L)$ is the eigenvalue of $L$.

---

Fig. 5: DFA $S_5$.

**Corollary 4.7 (Eigenvalue measure).**

The eigenvalue measure over irreducible regular languages is a language measure, i.e., it is increasing and starts at zero.

Corollary 4.7 stems from Theorem 4.6 and the facts that (i) the logarithm is strictly increasing and (ii) the eigenvalue of a zero matrix is equal to zero. Because of Corollary 4.7, one can use it to define a language quotient as follows:

**Definition 4.8 (Eigenvalue quotient).**

Given two irreducible regular languages $L_1$ and $L_2$, the eigenvalue quotient of $L_1$ over $L_2$ is the fraction of the eigenvalue of $L_1$ over the eigenvalue of $L_2$, i.e.,

$$\text{quotient}_{\text{eig}}(L_1, L_2) := \frac{\text{eig}(L_1)}{\text{eig}(L_2)}.$$ 

Eigenvalue quotients are defined over irreducible regular languages. In the next section, we show how one can use a language measure over irreducible regular languages to induce a language measure and, thus, language quotients over arbitrary regular languages.

As an example, consider automata $S_1$, $S_4$, and $S_5$ in Fig. 1b, Fig. 2, and Fig. 4. These three automata are ergodic and, thus, languages $L(S_1)$, $L(S_4)$, and $L(S_5)$ are irreducible. Moreover, it holds that $L(S_5) \subset L(S_4)$ and $L(S_4) \subset L(S_1)$, respectively.

Let $\text{quotient}_{\text{eig}}(L(S_5), L(S_4)) = 0.539$, $\text{quotient}_{\text{eig}}(L(S_4), L(S_1)) = 0.513$, $\text{quotient}_{\text{eig}}(L(S_5), L(S_1)) = 0.952$, and $\text{quotient}_{\text{eig}}(L(S_5), L(X^*)) = 0.242$, where $X$ is the set of symbols $\{a, b, c, d, e\}$. Indeed, it holds that: (i) $\text{quotient}_{\text{eig}}(L(S_5), L(S_5)) > \text{quotient}_{\text{eig}}(L(S_5), L(S_4))$ (refer to Lemma 4.2), (ii) $\text{quotient}_{\text{eig}}(L(S_5), L(S_1)) < \text{quotient}_{\text{eig}}(L(S_5), L(S_1))$ (refer to Lemma 4.3), and (iii) $\text{quotient}_{\text{eig}}(L(S_5), X^*) < \text{quotient}_{\text{eig}}(L(S_5), L(S_4))$ (refer to Corollary 4.4).

---

Fig. 5: (a) Adjacency matrices of DFAs in (a) Fig. 2 and (b) Fig. 4.

To demonstrate that $\text{quotient}_{\text{eig}}(L(S_5), L(S_4))$ equals to 0.952, Fig. 5a and Fig. 5b show adjacency matrices of $S_5$ and $S_5$, respectively. Note that the Perron-Frobenius eigenvalue of the matrix in Fig. 5a is 1.2702, while the Perron-Frobenius eigenvalue of the matrix in Fig. 5b is 1.21061.

---

5 PreciseR and Recall in Process Mining

Language quotients, as introduced in the previous section, provide a general means for behavioural comparison. To
demonstrate the use of the quotients, this section proposes and discusses their application in process mining [5]. One of the problems studied in process mining is the problem of process model discovery. Given an event log, which is a record of observed executions of a system, a discovery technique constructs a model of the system that “represents” the behaviour captured in the event log. As a system may execute same sequences of actions multiple times, its event log is a multiset of words that encode the executions.

**Definition 5.1** (Event log).

An *(event)* log is a finite multiset over a language.

An element of an event log is a *trace* of the log, whereas an element of a trace is an *event* of the trace. Given an event log \( L \), \( L(\mathcal{L}) := \text{Set}(\mathcal{L}) \) is the language of \( \mathcal{L} \). For example, event logs \( L_1, L_2, \) and \( L_3 \), listed in Fig. 1e contain two, five, and three traces, respectively. Note that \( L_2 \) contains trace \((a, f, e)\) twice, which denotes that this sequence of actions was executed and recorded in the log twice.

The quality of the generated process model is typically evaluated using precision, fitness (a specific type of recall), simplicity, and generalization [16]. Since both the latter relate to the usage context of the process model, we restrict the following discussion to precision and recall. More specifically, we use the framework of behavioural quotients to define a precision and recall of models w.r.t. their event logs (Section 5.1) and instantiate it based on the short-circuit language measure (Section 5.2). Finally, we demonstrate that our precision and recall quotients satisfy important requirements for precision and recall measures [5] (Section 5.3).

### 5.1 Definition of Precision and Recall

This section proposes two quotients for comparing behaviours captured in a log and a DFA, namely precision and recall of a DFA w.r.t. a log. These quotients are inspired by the precision and recall measures that have proved to be useful in information retrieval, binary classification, and pattern recognition. The precision and recall measures proposed here can be used to measure precision and fitness, respectively, of models discovered from event logs.

In information retrieval, given a set of relevant documents and a set of retrieved documents, precision is the fraction of relevant retrieved documents over the retrieved documents. Given an event log and a DFA, we propose to measure how precisely a DFA (model) describes an event log as the fraction of executions observed and recorded in the log and also modelled in the DFA over all the executions (of which there can be infinitely many) modelled in the DFA.

**Definition 5.2** (Precision of DFA w.r.t. event log).

Given an event log \( \mathcal{L} \) and a DFA \( B \), the precision of \( B \) w.r.t. \( \mathcal{L} \) induced by a language measure \( m \) is denoted by \( \text{precision}_m(B, \mathcal{L}) \) and is the language quotient induced by \( m \) of the intersection of the languages of \( B \) and \( \mathcal{L} \) over the language of \( B \), i.e.,

\[
\text{precision}_m(B, \mathcal{L}) := \text{quotient}_m(L(B) \cap L(\mathcal{L}), L(B)).
\]

Precision is thus the ratio of the measure of traces of the event log that are also computations of the DFA (modelled and observed behaviour) to the measure of all the computations of the DFA (modelled behaviour).

For example, the precision of automaton \( S_3 \) in Fig. 1d w.r.t. event log \( L_2 \) in Fig. 1e induced by the cardinality of a language is computed as follows:

\[
\text{precision}_{\text{card}}(S_3, L_2) = \frac{|L(S_3) \cap L(L_2)|}{|L(S_3)|} = \frac{1}{2}.
\]

Indeed, the languages of \( S_3 \) and \( L_2 \) share word \( \sigma := (a, b, c, d, e) \) only, while the language of \( S_3 \) consists of two words: \( \tau \) and \( (a, b, c, d, e) \).

In information retrieval, given a set of relevant documents and a set of retrieved documents, recall is the fraction of relevant retrieved documents over the relevant documents. Given an event log and a DFA, we propose to measure how good the DFA (model) captures the behaviour of the log as the fraction of executions observed and recorded in the log and also modelled in the DFA over all the behaviour recorded in the event log.

**Definition 5.3** (Recall of DFA w.r.t. event log).

Given an event log \( \mathcal{L} \) and a DFA \( B \), the recall of \( B \) w.r.t. \( \mathcal{L} \) induced by a language measure \( m \) is denoted by \( \text{recall}_m(B, \mathcal{L}) \) and is the language quotient induced by \( m \) of the intersection of the languages of \( B \) and \( \mathcal{L} \) over the language of \( \mathcal{L} \), i.e.,

\[
\text{recall}_m(B, \mathcal{L}) := \text{quotient}_m(L(B) \cap L(\mathcal{L}), L(\mathcal{L})).
\]

Recall is therefore the ratio of the measure of traces of the event log that are also computations of the DFA (modelled and observed behaviour) to the measure of the traces of the event log (observed behaviour).

For example, the recall of automaton \( S_3 \) in Fig. 1d w.r.t. event log \( L_2 \) in Fig. 1e induced by the cardinality of a language is computed as follows:

\[
\text{recall}_{\text{card}}(S_3, L_2) = \frac{|L(S_3) \cap L(L_2)|}{|L(L_2)|} = \frac{1}{4}.
\]

This result is easy to verify by checking that the language of \( L_2 \) consists of four words.

### 5.2 Short-circuit Language Measure

The notions of fitness and recall of a DFA w.r.t. a log, refer to Section 5.1 for details, take a language measure as a parameter. The language of an event log is finite. If the language of the DFA is also finite, one can instantiate the precision and recall with the cardinality of a language, similar as proposed in Definition 4.5 and exemplified in Section 5.1. To account for irreducible regular languages, one can instantiate the quotients with the eigenvalue measure, refer to Section 4.3 for details. Unfortunately, the language of an event log is not irreducible. Moreover, the language of a DFA is not guaranteed to be irreducible. To overcome these limitations, in this section, we propose a short-circuit measure over languages.

**Definition 5.4** (Short-circuit measure over languages).

A short-circuit measure over languages over alphabet \( \Psi \subseteq \Xi \) induced by a short-circuit measure over alphabet \( \Psi \) is the (set) function \( m^* : \Psi(\Xi^*) \to \mathbb{R}_0^+ \) defined by \( m^*(L) := m((L \circ \{\chi\})^* \circ L) \), where \( L \) is a language over \( \Psi \), i.e., \( L \subseteq \Psi^* \), and \( \chi \in \Xi \setminus \Psi \) is a short-circuit symbol.
Below, we demonstrate that a short-circuit measure over regular languages induced by a language measure over irreducible languages is a language measure, refer to Proposition 5.10. Hence, the quotients induced by such short-circuit measures enjoy all the properties proposed in Section 4.2.

If a short-circuit measure is induced by a measure that starts at zero, then it also starts at zero.

**Lemma 5.5** (Short-circuit measure starts at zero).
If \( m \) is a measure over languages over alphabet \( \Phi \subseteq \Xi \) that starts at zero, then a short-circuit measure \( m^* \) over languages over alphabet \( \Psi \subseteq \Phi \) starts at zero.

**Proof.** Assume that \( m^*(L) := m((L \circ \{\chi\})^* \circ L) \), where \( L \) is a language over \( \Psi \) and \( \chi \in \Xi \setminus \Psi \). Let \( L' \) be the empty language, i.e., \( L' = \emptyset \). Then, it holds that \( (L' \circ \{\chi\})^* \circ L' = \emptyset \). Consequently, it holds that \( m^*(\emptyset) = m(\emptyset) \). Finally, because \( m \) starts at zero, it holds that \( m^* \) also starts at zero.

Moreover, if a short-circuit measure is induced by an increasing measure, then it is also increasing.

**Lemma 5.6** (Short-circuit measure is increasing).
If \( m \) is an increasing measure over languages over alphabet \( \Phi \subseteq \Xi \), then a short-circuit measure \( m^* \) over languages over alphabet \( \Psi \subseteq \Phi \) is increasing.

**Proof.** Assume that \( m^*(L) := m((L \circ \{\chi\})^* \circ L) \), where \( L \) is a language over \( \Psi \) and \( \chi \in \Xi \setminus \Psi \). Let \( W \subseteq \Psi^* \) and \( V \subseteq \Psi \) be two languages such that \( U \subseteq V \). Let \( W \subseteq \Psi^* \) be a language such that \( U = V \setminus W \) and \( W \subseteq V \). Then, it holds that \( W \neq \emptyset \).

For all \( u \in (U \circ \{\chi\})^* \circ U \), it holds that \( u \in (V \circ \{\chi\})^* \circ V \); note that \( u \in V \). Let \( W \subseteq W \) be a word. Then, it holds that \( w \circ w \in (V \circ \{\chi\})^* \circ V \); note that \( w \in V \). In addition, it holds that \( w \circ w \notin (U \circ \{\chi\})^* \circ U \); note that \( w \notin U \).

Hence, \( (U \circ \{\chi\})^* \circ U \subseteq (V \circ \{\chi\})^* \circ V \). Then, it holds that \( m((U \circ \{\chi\})^* \circ U) < m((V \circ \{\chi\})^* \circ V) \), and, consequently, \( m^*(U) < m^*(V) \).

Thus, if \( m \) is a language measure, then \( m^* \) is a language measure as well.

**Corollary 5.7** (Short-circuit measure).
If \( m \) is a language measure over languages over alphabet \( \Phi \subseteq \Xi \), then a short-circuit measure \( m^* \) over languages over alphabet \( \Psi \subseteq \Phi \) is a language measure.

The proof of Corollary 5.7 follows immediately from the definition of a language measure, refer to Section 4.1, and Lemmata 5.5 and 5.6. Next, we establish a relation between short-circuit measures and the family of language measures over irreducible regular languages. To this end, we recall some basic properties of regular languages.

The result of the concatenation of two regular languages is a regular language.

**Theorem 5.8** (Concatenation closure).
The class of regular languages is closed under the concatenation operation.

For the proof of Theorem 5.8 refer to Theorem 1.47 in [13].

If \( L \) is a regular language, then \( L^* \) is regular.

**Theorem 5.9** (Star closure).
The class of regular languages is closed under the Kleene star operation.

For the proof of Theorem 5.9 refer to Theorem 1.49 in [13].

Finally, we show that a language measure over irreducible regular languages can be used to induce a language measure over regular languages.

**Proposition 5.10** (Short-circuit measure over languages).
If \( m \) is a language measure over irreducible regular languages over alphabet \( \Phi \subseteq \Xi \), then \( m^* \) is a language measure over regular languages over alphabet \( \Psi \subseteq \Phi \).

**Proof.** Assume that \( m^*(L) := m((L \circ \{\chi\})^* \circ L) \), where \( L \) is a regular language over \( \Psi \) and \( \chi \in \Xi \setminus \Psi \). Let \( w_1 \) and \( w_2 \) be two words in \( (L \circ \{\chi\})^* \circ L \). Then, by construction, it holds that \( w_1 \circ \chi \circ w_2 \in L \circ \{\chi\} \); note that \( w_1 \circ \chi \circ w_2 \subseteq L \). Hence, \( (L \circ \{\chi\})^* \circ L \) is irreducible. Moreover, \( (L \circ \{\chi\})^* \circ L \) is a regular language, refer to Theorem 5.8 and Theorem 5.9; note that \( \{\chi\} \) is a regular language. Finally, because of Corollary 5.7, \( m^* \) is a language measure.

For example, \( \text{eig}^* \) is a language measure over regular languages, where \( \text{eig} \) is the eigenvalue measure; note Corollary 4.7 and Proposition 5.10. Therefore, we recommend using \( \text{eig}^* \) measure to induce various language quotients over arbitrary regular languages, e.g., the precision and recall quotients proposed in Section 5.1.

Consider automaton \( S_1 \) in Fig. 1b and event log \( L_3 \) in Fig. 1e. Fig. 7a and Fig. 7b show automata with languages \( (L(q_1) \circ \{\chi\}) \circ L(q_1) \) and \( (L(q_3) \circ \{\chi\}) \circ L(q_3) \), respectively, whereas Fig. 7c shows an automaton with language \( (L(q_1) \cap L(q_3)) \circ \{\chi\} \circ (L(q_1) \cap L(q_3)) \). It is easy to see that given a \( \tau \)-free DFA \( B := (Q, \Delta, \delta, \epsilon, A) \), it holds that \( (L(B') = (L(B) \circ \{\chi\}) \circ L(B) \); where \( B' := (Q, \Delta \cup \{\chi\} \cup \delta \cup A \setminus \{\epsilon\}, \epsilon, \{q_0\}, A) \). Note that the automaton in Fig. 7a was obtained from the automaton in Fig. 1b using this simple transformation and subsequent minimization [17]. Such minimization is possible because any automaton with the language of interest, in this case \( (L(q_1) \circ \{\chi\}) \circ L(q_1) \), suffices. Fig. 6 shows the adjacency matrix of the automaton in Fig. 7a. The Perron-Frobenius eigenvalue of this matrix is 2.521. The Perron-Frobenius eigenvalues of the adjacency matrices of automata in Fig. 7b and Fig. 7c are 1.226 and 1.128, respectively. Thus, it holds that \( \text{precision}_{\text{eig}}(S_1, L_3) = 0.447 \) and \( \text{recall}_{\text{eig}}(S_1, L_3) = 0.92 \).

<table>
<thead>
<tr>
<th>Table 1: Precision and recall values.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automaton</strong></td>
</tr>
<tr>
<td>S_1</td>
</tr>
<tr>
<td>S_1</td>
</tr>
<tr>
<td>S_1</td>
</tr>
<tr>
<td>S_2</td>
</tr>
<tr>
<td>S_2</td>
</tr>
<tr>
<td>S_2</td>
</tr>
<tr>
<td>S_3</td>
</tr>
<tr>
<td>S_3</td>
</tr>
<tr>
<td>S_3</td>
</tr>
</tbody>
</table>
All the precision and recall values induced by $\text{eig}^*$ for each of the three DFAs in Figs. 1b–1d w.r.t. every event log in Fig. 1e are listed in Table 1. Note that these values obey all the properties discussed in Section 4.2.

5.3 Properties of Precision and Recall

Precision and recall, as defined in Section 5.1, are language quotients and, thus, possess all the properties of language quotients discussed in Section 4.2. This section proposes some further properties specific for the precision and recall of a DFA w.r.t. an event log.

Firstly, precision and recall take values from the interval that contains zero and one as well as all the numbers between zero and one.

**Proposition 5.11 (Precision interval).**
Given an event log $L$, a DFA $B$, such that $L(B) \neq \emptyset$, and a language measure $m$ over regular languages, it holds that $0 \leq \text{precision}_m(B, L) \leq 1$.

Proposition 5.11 follows immediately from Definition 5.2 and the fact that $m$ is a language measure over regular languages. Indeed, it holds that $L(B) \cap L(L) \subseteq L(B)$ and, thus, $m(L(B) \cap L(L)) \leq m(L(B))$; note that $m$ is increasing.

**Proposition 5.12 (Recall interval).**
Given an event log $L$, such that $L(L) \neq \emptyset$, a DFA $B$, and a language measure $m$ over regular languages, it holds that $0 \leq \text{recall}_m(B, L) \leq 1$.

Proposition 5.12 holds because of Definition 5.3, and the facts that $L(B) \cap L(L) \subseteq L(L)$ and $m$ is increasing.

Secondly, precision and recall are maximal, i.e., are equal to one, when the languages of the DFA and log are in containment relations.

**Proposition 5.13 (Maximal precision).**
Given an event log $L$, a DFA $B$, such that $L(B) \neq \emptyset$, and a language measure $m$ over regular languages, $L(B) \subseteq L(L)$ iff $\text{precision}_m(B, L) = 1$.

If $L(B) \subseteq L(L)$, then it holds that $\text{precision}_m(B, L) = m(L(B))/m(L(L)) = 1$. Conversely, if $\text{precision}_m(B, L) = 1$, then $m(L(B) \cap L(L)) = m(L(B))$. Then, it holds that $L(B) \subseteq L(L)$.

**Proposition 5.14 (Maximal recall).**
Given an event log $L$, such that $L(L) \neq \emptyset$, a DFA $B$, and a language measure $m$ over regular languages, $L(L) \subseteq L(B)$ iff $\text{recall}_m(B, L) = 1$.

The proof of Proposition 5.14 follows the structure of the proof of Proposition 5.13 but swaps the roles of the languages of $B$ and $L$.

Thirdly, precision and recall are both maximal, i.e., are equal to one, iff the languages of the DFA and log are identical.

**Corollary 5.15 (Maximal precision and recall).**
Given an event log $L$, such that $L(L) \neq \emptyset$, a DFA $B$, such that $L(B) \neq \emptyset$, and a language measure $m$ over regular languages, $L(B) = L(L)$ iff $\text{precision}_m(B, L) = 1$ and $\text{recall}_m(B, L) = 1$.

Corollary 5.15 follows immediately from Proposition 5.13 and Proposition 5.14.

Finally, precision and recall are minimal, i.e., are equal to zero, when the languages of the DFA and log do not overlap.

**Proposition 5.16 (Minimal precision).**
Given an event log $L$, a DFA $B$, such that $L(B) \neq \emptyset$, and a language measure $m$ over regular languages, $L(B) \cap L(L) = \emptyset$ iff $\text{precision}_m(B, L) = 0$.

If $L(B) \cap L(L) = \emptyset$, then $\text{precision}_m = m(\emptyset)/m(L)$ because $m(\emptyset) = 0$; note that $m$ starts at zero. Conversely, if $\text{precision}_m(B, L) = 0$, then $m(L(B) \cap L(L)) = 0$. Then, $L(B) \cap L(L) = \emptyset$ because $m$ starts at zero and is increasing.

**Proposition 5.17 (Minimal recall).**
Given an event log $L$, such that $L(L) \neq \emptyset$, a DFA $B$, and a language measure $m$ over regular languages, $L(B) \cap L(L) = \emptyset$ iff $\text{recall}_m(B, L) = 0$.

The proof of Proposition 5.17 follows the structure of the proof of Proposition 5.14 but swaps the roles of the languages of $B$ and $L$.

6 Experimental Evaluation

In the following, we briefly describe the implementation of the precision and recall induced by $\text{eig}^*$. Then, we empirically show the theoretical advantages of the proposed eigenvalue-based measures by comparing them with existing approaches for measuring precision and recall in process mining. Tables 2 and 3 list the approaches considered for our comparative evaluation, detailed later on in Section 7.

6.1 Implementation

By $\text{det}(B)$, we denote the deterministic version of $B$, i.e., the DFA with the language $\text{lang}(B)$. Given $B$, $\text{det}(B)$ always exists, refer to Theorem 3.5, and can be constructed using the Rabin-Scott powerset construction method [9], which has the worst-case time complexity of $O(2^n)$, where $n$ is the number of states in the NFA [25]. For each regular language, there exists a unique (up to isomorphism) DFA with a minimum number of states [26] that recognizes the language. Let $B'$ be a DFA. By $\text{min}(B')$, we denote the minimal version of $B'$, i.e., the DFA with a minimum number of states
that recognizes $\text{lang}(B')$. There exist several algorithms that given a DFA $B'$ construct $\text{min}(B')$. For example, the worst-case time complexity of the Hopcroft algorithm [17] is $O(nm \log(m))$, where $n$ is the number of states and $m$ is the size of the alphabet.

In a nutshell, we accept as input any nondeterministic automaton $B$ and compute its deterministic version $B'$, which we subsequently minimize to get $\text{min}(B')$. Then, we compute the largest eigenvalue of $\text{min}(B')$. To ensure that the eigenvalue is computable, we transform the DFA $\text{min}(B')$ by short-circuiting, as described in Section 5.2. Short-circuiting is a simple $O(n)$ operation, where $n$ is the number of the (sink-)nodes in $\text{min}(B')$. After short-circuiting, we create the adjacency matrix of the resulting automaton. The adjacency matrix serves as input to existing numerical Fortran-based methods for determining the largest eigenvalue of a matrix [27]. Note that typically, we would expect that the matrix of a language is rather sparse. Thus, we are able to handle very large graphs on personal computers and compute their eigenvalues with the help of memory-friendly sparse data structures for matrices.

Compressed column storage is a typical format for sparse matrices. We use the Java library Matrix Toolkit Java (MTJ) that relies on the low level libraries in ARPACK [27] to run the eigenvalue computation. However, MTJ only exposes the eigenvalue computation of symmetric matrices in ARPACK. Note that the adjacency matrices of automata are usually not symmetric. To this end, we adapted MTJ to be able to use the ARPACK routines to compute eigenvalues of general matrices. Our extended implementation for computing the largest eigenvalue for non-symmetric matrices is made publicly available.

So far we know how to compute the eigenvalue of a language. The next step is to determine the quotient of two languages. To compute precision and recall, we need to compute their intersection. The intersection of languages is a well-known operation from automata theory, and its complexity is $O(nm)$, where $n$ and $m$ are the numbers of states in the two automata [26].

Finally, the implementation contains the computation of recall and precision measures for process mining. The only remaining step is to translate input models into their corresponding automata. An event log can be trivially encoded as an automaton that accepts the set of words contained in the event log, e.g., by capturing each trace as a sequence of transitions starting at the start state. Thus, the problem of computing precision and recall is reduced to computing the automata of the model and the log, their intersection, and the respective eigenvalues. These three eigenvalues are the basis for computing the two quotients of recall and precision. With the eigenvalue of the intersection automaton in the numerator we can use the eigenvalue of the model as denominator to compute precision, or swap it with the eigenvalue of the log to obtain recall.

### 6.2 Monotonicity Experiments

In the experiments, we want to highlight the advantages of the monotonicity property of the eigenvalue-based measure as compared to existing measures.

Without restriction of generality, we demonstrate the monotonicity properties on both sides of the domain of process mining with different models and different event logs. By varying the amount of behaviour of the model with respect to a fixed event log, we expect the precision to decrease by adding additional behaviour to the model. The precision should increase by removing excess behaviour from the model with respect to a fitting event log.

That is, in all our experiments, we fix all variables, and vary a specific variable to see the effect. We expect strictly monotone behaviour in the resulting quotients, when increasing or decreasing one language either in the numerator or denominator of the language quotient. Note that we expect the occasional numerical instability of our method, as the numerical methods to compute an eigenvalue of a general matrix provide no formal guarantees of convergence in a fixed amount of time or iterations. Therefore, we ran the experiments with a set maximum number of iterations (300k) for practical reasons.

We use the Comprehensive Benchmark Framework for the evaluation of the precision and recall measures [30].

#### 6.2.1 Monotonicity of Precision Measures

We compare the proposed precision and recall measures induced by $\text{eig}^*$ (cf. Section 5.3) with the precision measure candidates proposed in literature (cf. Section 7). In the following, we use regular expressions to describe the model languages. In the first experiment, we add behaviour to the model starting with a perfectly fitting model to the log that has up to two $a$’s before $b$. More precisely:

$L$ is the log with the language $a\{0,2\}ob$;

$M_x$ are the models with language $a\{0,x\}ob$;

$M_*$ is the model with language $a^*ob$.

Notation $a\{\min, \max\}$ is short-hand for enumerating the minimal and maximal number of repetitions of $a$. In this case, we get the log $[b, a,b, a,a,b]$. 

---

**TABLE 2: Precision measures used in this evaluation.**

<table>
<thead>
<tr>
<th>Short label</th>
<th>Full name and reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>advBehAppropriateness</td>
<td>Advanced behavioural appropriateness [18]</td>
</tr>
<tr>
<td>alignmentPrecision</td>
<td>Alignment-based precision [19]</td>
</tr>
<tr>
<td>antiAlignPrecision</td>
<td>Anti-alignments precision [20]</td>
</tr>
<tr>
<td>bestAlignPrecision</td>
<td>Best optimal-alignments precision [21]</td>
</tr>
<tr>
<td>negativeEventPrecision</td>
<td>AGNEs specificity [22]</td>
</tr>
<tr>
<td>oneAlignPrecision</td>
<td>One optimal-alignments precision [21]</td>
</tr>
<tr>
<td>precisionEig</td>
<td>This paper</td>
</tr>
<tr>
<td>precisionETC</td>
<td>ETC precision [23]</td>
</tr>
<tr>
<td>projectedPrecision</td>
<td>PCC precision [24]</td>
</tr>
<tr>
<td>simpleBehAppropriateness</td>
<td>Simple behavioural appropriateness [18]</td>
</tr>
</tbody>
</table>

**TABLE 3: Fitness measures used in this evaluation.**

<table>
<thead>
<tr>
<th>Short label</th>
<th>Full name and reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignmentFitness</td>
<td>Alignment-based fitness [28]</td>
</tr>
<tr>
<td>negativeEventRecall</td>
<td>AGNEs recall [22]</td>
</tr>
<tr>
<td>tokenBasedFitness</td>
<td>Token-based fitness [18]</td>
</tr>
<tr>
<td>parsingMeasure</td>
<td>Continued parsing measure [29]</td>
</tr>
<tr>
<td>projectedRecall</td>
<td>PCC recall [24]</td>
</tr>
<tr>
<td>properCompletion</td>
<td>Proper completion [18]</td>
</tr>
<tr>
<td>recallEig</td>
<td>This paper</td>
</tr>
</tbody>
</table>

---

5Sources can be found at https://github.com/andreas-solti/matrix-toolkits-java and at the Maven Central repository.
Fig. 8: Increasing number of optional a’s before b. Starting with up to two a’s before b, stepwise allow more a’s up to the closure that allows an arbitrary number of a’s before b.

Fig. 8 shows the results of various precision measures on the y-axis, plotted for the increasing model language starting from 0–2 possible repetitions of a before b up to 0–20 possible repetitions, refer to the x-axis. The last measurement on the end of the x-axis denotes the precision with respect to the model that allows an arbitrary number (i.e., 0–∞) of a’s before the b, that is a^*ob.

The simple behavioural appropriateness measure [18] shows a trend contrary to the other measures, as the precision increases with a more permissive model. Advanced behavioural appropriateness [18] fails to recognize the growth of the model’s language that makes it less “precise”. The anti-alignment measure [20] has shown the correct trend, but fails with the computation of the a^*ob model. Projected precision is strictly monotone in the region between up to 2 and up to 20 a’s before b, but violates monotonicity at the closure. The remaining measures show a similar behaviour starting at 1.00 for the perfectly fitting model and decreasing but stabilizing quickly. These measures, however, do not distinguish between the models a{0, y}ob, where y ∈ [3 . . . 20]. Our eigenvalue-based precision measure shows a steady stabilizing decline the more possible repetitions of a are added to the model and distinguishes all the models by their precision values.

Besides iteration, parallelism, captured as possible inter-leaving of symbols is another dimension that we investigate. We compare the varying interleaving recordings of a fixed alphabet of size 5. This experiment corresponds to drawing five out of five symbols from an alphabet without replacement, where the order matters. Hence, there are 5! = 120 distinct combinations of symbols, i.e., 120 distinct words. A process model that allows parallel execution of five activities also permits exactly 120 different executions. We would expect a model that enumerates all 120 combinations to be equally precise, as another model using the corresponding parallel building block that says that the same five activities can be done in any order.

Next, we use the following log and models.

\[ L_{5|} \] is the log with the language \{abcde, abced, abdec, abdce, abced\}.

\[ M_{5|} \] are the models with the language of \[ L_{5|} \] and further permutations, such that 5 ≤ x ≤ 120 and \(|L(M_{x|})| = x\) (i.e., the number of allowed traces is x).

\[ M_{||} \] is the model with language of all 120 combinations of the five symbols a, b, c, d, e.

Most existing precision measures (cf. listed in Table 2 and discussed in Section 7) are monotonically decreasing for the fixed log \[ L_{5|} \] and an increasingly permissive model, refer to Fig. 9. However, for the given log, the model of 120 explicit permutations can have different precision than the model with five activities in parallel, although these two models have the same language. Note that only three measures reported the same precision values for these models: advanced behavioural appropriateness, projected precision, and our eigenvalue-based precision measure.

The monotonicity of the experiment is violated by the ETC precision [31], by the one-align, and best-align, and anti-alignment based precisions [3, 20], and also the simple behavioural appropriateness [18]. The anti-alignment based precision [20] fails to compute a value for the fully parallel model in the given time. Simple behavioural appropriateness can only be computed up to 110 permutations, and the value drops almost in half when looking at the parallel model.

To check monotonicity in the denominator we investigate the real life event log of the BPI Challenge 2012 [32]. We mine a model \( M \) that is able to replay the entire log with the inductive miner infrequent and noise threshold parameter setting 0.0. Then, we select the first 5 percent of the log as the sublog \( L_{5|} \) and compute the precision of the model and the sub-log. Because we know that the model is able to replay the entire log, we know that \( \ell(L_{5|}) \leq L(M) \), that is the language of the log is contained in the language of the model.

Fig. 9: The behaviour of different precision measures for the log \( L \) with respect to models that allow different permutations of the same five symbols, including the precision of all the explicit 5! = 120 permutations, and the precision of the language equivalent parallel model with 120 implicitly allowed permutations.

Fig. 10: Expected increased precisions with more and more of the log behaviour of the main process (A) in [32].
Fig. 11: Each dot represents the relative increase or decrease in Fig. 10 at each subsequent measurement step as the size of the log increases. A relative decrease in precision is marked with a red triangle.

We repeat this process for increasing sub-logs in 5 percent steps, such that $L_{5}\% \subset L_{10}\% \subset \ldots \subset L_{100}\%$. The resulting precision values are reported in Fig. 10.

It is in the nature of the experiment that even if we increase the number of traces in the log in a step, new behaviour is not necessarily added. At each step, we can potentially end up adding only traces that the previous log already contained.

To make the results easily accessible, we created a difference plot shown in Fig. 11, which plots the deltas to the previous value (e.g., if the value increased by 0.1, when increasing the log by 5 percent, we add a mark at 0.1). Hence, for a monotonically increasing measure one should observe only non-negative values.

Only three precision measures are consistent in this setting, that is their graphs are monotonically increasing. These measures are advanced behavioural appropriateness, largest eigenvalue-based precision, and projected precision. Some negative values are due to the non-deterministic nature of some of the precision measures, as discussed in [6]. However, there is also a systematic error in the anti-alignment based precision measure [20] that reports an unexpected downward trend in precision, despite the fact that the model is fixed and the number of considered traces (and with them the behaviour) increases in this experiment.

6.2.2 Monotonicity of Recall Measures
Recall of a language model with respect to an event log is defined as the fraction of the shared behaviour by the behaviour in the event log. In this case, both measures capture finite behaviour, which makes this problem less challenging than measuring precision.

Fig. 12 shows the results of the following experiment. Given a sequential model of 10 activities and a fitting event log with no noise, we start increasing the amount of noisy traces in the event log. Here, noise is defined as removing, adding, or swapping events in the event log, and the percentage shown on the x-axis reflects the relative number of traces affected by noise. The recall values are plotted for different techniques. Very basic measures are the parsing measure and the notion of proper completion. These measures simply count the fraction of traces that are entirely fitting. In contrast alignment based fitness notions, together with the negative event recall and the token based fitness, look at the misalignments in a finer granularity. That is, they penalize minor deviations only slightly. In contrast, the parsing measure is based on a binary decision for each trace. This means that small deviations are counted as much as completely unrelated traces.

The proposed in this paper measure for recall depends on the size of the languages of the log and model. The aforementioned example shows that the behaviour in the log, by inserting some random noise, increases early on with the noise level. This leads to a rapid drop in recall, as indeed the model fails to capture the random behaviour, but only captures the deterministic sequential part of it. The effect of the increased number of noisy traces does not increase the behaviour of the log at higher noise levels that much, as the probability that a new noisy trace has already been seen increases with the number of noisy traces. In contrast, the other measures show a linear trend, as they do not take into account the size of the behaviour, but count the number of fitting traces with respect to the size of the log. As a consequence, traditional approaches treat the two cases listed in Table 4 equivalently, while our measure judges the recall of situation a) lower than that of b), as the variance in the log is lower in b, even though it has the same amount of deviating cases.

While the amount of noisy traces increases linearly in this experiment, we are interested in the behaviour that is in both model and log versus the behaviour in the event log only. The novel eigenvalue-based measure captures this non-linearity in the behaviour of the event log. Thus, we conclude that if one is interested in the measure of how much behaviour of a log, a model is able to capture, our measure is more suitable, while if we are interested only in the fitting part, and do not need to distinguish potentially different errors, the traditional fitness/recall measures are preferable. Latter linearly capture a decreasing number of fitting traces with respect to a given model.

6.3 Scalability Evaluation
Practical language measures and quotients must be able to handle large languages. Hence, we measured wall-clock time of the eigenvalue-based precision and recall for 12 large real-
life logs and models. The logs are publicly available\(^7\) and are of different complexities. The log with the least variation in traces (BPI Challenge 2013, open cases) can be translated into a finite acyclic automaton with only 116 states, while the BPI’17 log showing the most variation in traces produces an automaton with 105,387 states.

For all these logs, we mined a process model with the inductive miner \cite{simpson_towards_2015} with the default noise threshold 0.2. The models are considerably smaller in their automaton representation, as the mined models do not allow duplicate transitions. For all these log and model pairs, we applied our method by first constructing the respective finite automata and computing the eigenvalues of their short-circuited representations. The observed wall-clock times of the computations of the largest eigenvalues for the log \(L\), the model \(M\), and their intersection automaton \(L \cap M\) are shown in Table 5. As an indicator for the complexity, the number of states of the respective automata are listed in the table. An adjacency matrix of an automaton has size that is quadratic in the size of the automaton, which can pose practical difficulties when storing it on a computer. However, adjacency matrices are usually sparse, which allowed us to use their memory efficient representations.\(^8\)

In all but two of the real-life cases, our method computes the automata and their largest eigenvalues within 10 minutes to yield the precision and recall values. The variance in measured wall-clock times is remarkable, however. The longest time to compute the precision and recall was taken for the BPI’17 log. In this particular case, the numeric determination of the largest eigenvalue failed due to non-convergence within the set 300,000 iterations. The underlying technique called “implicitly restarted Arnoldi iterations” \cite{parlettimplicitly_1982} apparently has issues to handle this specific automaton matrix. In fact, the author of the software package states that “The question of determining a shift strategy that leads to a provable rapid rate of convergence is a difficult problem that continues to be researched.” \cite{dongarracomparing_2013} to mention but a few.

Section 7.1 firstly outline noticeable behavioural equivalence concepts in the literature and behavioural comparison including inheritance and similarity. Then, Section 7.2 describes the evolution of the precision and recall behavioural comparison metrics in the field of process mining, along with highlights on commonalities and differences to our approach.

### 7 Related Work

The comparison of behaviours has played a major role in the verification of software and hardware artefacts across several areas of computer science and software engineering, including the theory of concurrent systems \cite{milner1989concurrent,hoare1978communicating}, reactive systems \cite{alur1994model,alur2013reactive}, and agent programming \cite{castelfranchi1999agreement,castelfranchi2001right} to mention but a few.

Section 7.1 firstly outline noticeable behavioural equivalence concepts in the literature and behavioural comparison including inheritance and similarity. Then, Section 7.2 describes the evolution of the precision and recall behavioural comparison metrics in the field of process mining, along with highlights on commonalities and differences to our approach.

#### 7.1 Behavioural equivalence

In the context of dynamic systems, there are several notions of behavioural equivalence, which are broadly classified into two categories: equivalences that are based on the interleaving semantics and those based on the true concurrency semantics \cite{hirschkoff2003transfinite}. We remark that the models under analysis in this paper fall under the class of finite-state, assume the presence of final/accepting states, and operate with interleaving semantics. Probably the most important behavioural equivalence between two models of computation in this context is the one that guarantees that any step performed in one model can be mimicked by the other one, and vice versa \cite{hirschkoff2003transfinite}. This idea is the basis for the notion of bisimulation \cite{milner1980calculus,robinson1994bisimulation}. On rooted labelled transition systems (a super-class of the models we analyse), bisimulation imposes that from the initial state onward, possible actions must coincide between the models and inductively lead to states that are bisimilar as well. Weak bisimulation \cite{robinson1994bisimulation} relaxes bisimulation in that it considers only observable actions, i.e., it is permitted that models guarantee bisimulation on non-\(\tau\) transitions only, as \(\tau\) transitions can be added as prefix- or suffix-moves to that extent. Branching bisimulation enforces

---

\(^7\)Event logs are available at: https://data.4tu.nl/repository/collection:event_logs_real

\(^8\)The experiments and the code used in this paper are available at https://github.com/andreas-solti/eigen-measure
weak bisimulation by requiring that the same set of choices is offered before and after each unobservable action [44]. Bisimulation exerts less strict conditions than graph isomorphism, which is a bijection between all states preserving transitions. However, it is also more specific than trace equivalence, solely ascertaining that observable actions match, thus being insensitive to non-determinism, internal actions, choices, and deadlocks [35]. Completed trace equivalence adds the condition that, if models have sink states from which no further action is possible, they must be reachable in models by replaying the same traces. Our research benefits from the multiple notions of behavioural equivalence and investigations conducted on the matter so far, yet it abstracts from the decision problem on the matching of behaviours and rather aims at assessing how much the behaviour of a first model is extended by a second one.

In [45, Ch. 7], Kunze and Weske declare not only behavioural equivalence, but also behavioural similarity and inheritance, as main challenges pertaining to behavioural comparison. In particular, the authors introduce a property for the latter, namely trace inheritance, which is enjoyed only if the language of a system model is included in the language of another system model at the same level of abstraction. In the light of that definition, our research thus focuses on behavioural equivalence [46], and specifically trace inheritance, between behavioural models. However, we aim at providing a measure assessing in how far languages extend one another, rather than checking whether the property holds true or not. This quantitative aspect typically pertains more to behavioural similarity. To measure it, applying naïve approaches based on set-similarity measures such as the Jaccard coefficient [47] to the set of models’ traces proves infeasible: loops lead to trace sets of infinite cardinality. To overcome that problem, approaches to behaviour similarity were introduced that restricted the analysis to local relations between traces’ events [48]. Noticeable examples include the $n$-gram similarity [49], comparing models by the shared allowed $n$-long sub-sequences in models’ respective traces. Despite the efficiency of the solution, the treat to validity is that even if $n$-grams coincides, not necessarily do the traces as well. Nevertheless, the best results are reportedly achieved with the least strict parameter, namely $n = 2$. Later on, behavioural profiles similarity was introduced in [48]. The main idea is to compare “footprints” of models, obtained by matrices connecting pairs of event labels with mutually exclusive relations. Those relations are exclusiveness, strict order, and interleaving order, i.e., the fundamental relations of behavioural profiles as of [7]. Despite being semantically reach, it is shown in [8] that the expressive power of behavioural profiles is strictly less than regular languages, thus entailing that they cannot be used to decide trace equivalence of finite state automata. Our approach abstracts from the local perspective on traces or relations between events in that it resorts on the topological entropy to compare the variability of languages. We reflect the comparison of behavioural models into precision and recall.

7.2 Precision and recall in process mining

In our research, we put a stronger emphasis on the area of business process management [50, 51] and process mining in particular [5]. Process mining is the field of science that aims at extracting knowledge about business processes from the digital data stored by organisations’ IT systems. Process mining is adopted to discover new facts, including process models themselves that were not documented before, compare the expected process behaviour with reported reality and detect deviations between the former and the latter [52]. It shows thus the inherent aim of finding and assessing the match between the behaviour of dynamic systems, in terms of to-be process models versus as-is process data. Therefore, the identification of quotients that allow for a comparative measurement of behaviours naturally suits the matter. In particular, Buijs et al. [53] identified as (replay) fitness, precision, generalization, and simplicity the four main quality dimensions for assessing the quality of process mining results [54]. Those metrics were the result of a strive for a comparative evaluation framework in the field begun years before by Rozinat et al. [55].

A first precision measures called “behavioural appropriateness” is introduced in [18] as the degree of how much behaviour is permitted by the model although not observed in the event log. The simple behavioural appropriateness builds on the observation that an increase of alternatives or parallelism entails a higher number of enabled transitions during log replay, while the advanced behavioural appropriateness uses long-distance precedence dependencies between pairs of activities [56]. In this way, it is higher when sometimes-forward and sometimes-backward relation pairs shared between model and log approximate the total amount of the model. Conversely, it is lower if the model allows for more variability. The assumption of total fitness of the log entails that the log cannot show more variability than the model. Our approach also compares the availability of actions at given states, but abstracts from the exact replay of traces by considering the entropy of the languages. Therefore, no pairwise-comparison of behavioural relations between actions is needed.

The ETConformance approach avoids the complete exploration of the model behaviour by traversal of the model to solely reflect the traces recorded in the log [57]. To that extent, a finite (acyclic) rooted deterministic labelled transition system named prefix automaton is generated by folding traces based on prefix trace-equivalence of the generated states. The assumption of total fitness entails that the set of available transitions contains the ones permitted by the prefix automaton. The approach is further extended in [23]. The locality of the approach allows for a higher efficiency of the computation, with the downside that only behaviour close to the event log is considered. Similarly, our approach assesses precision by quantifying the behavioural differences among states of a finite-state rooted labelled transitions systems. However, it abstracts from the recorded runs of the involved models.

In [28], the authors propose an approach combining the concept of prefix automaton with the one of alignments [19] to deal with non-entirely fitting logs. The proposed alignment-based precision is the arithmetic mean over all events in the log of the ratio between the activities that were allowed by the model and the ones that were actually executed as per the prefix automaton, given the replay history. In [21] different precision measures are proposed based on the nature of
the alignments to be considered. The underlying structure remains a prefix automaton, as in [23,57], augmented by associating weights to states. As in the approaches of [28] and [21], the precision measure proposed here does not take into account diverging behaviours. To that extent, the log repair given by alignments could be beneficial to a pre-processing phase. Because our solution resorts on the entropy of models’ languages, it abstracts from the replay and counting of events.

More recently, [24] introduce precision and recall metrics to compare the behaviour of models or logs, requiring a finite state automaton as the underlying structure for a state-to-state comparison like [21,23,57]. To cope with the high computational effort required by the intersection operations, a projection of both models is pre-computed for every subset of $k$ actions in the joint alphabet. Resulting automata contained silent transitions and presented non-determinism. The resulting Projected Conformance Checking (PCC) precision and a corresponding recall measure build then on $k$-subsets projections. As in [24], we benefit from minimisation of the underlying structure and provide dual definitions for precision and recall. However, the computation of measures based on eigenvalues does not require the approximation via $k$-projections.

The anti-alignment based precision is defined in [20] using the concept of anti-alignment first proposed in [58]. An anti-alignment is a finite trace of a given length which is accepted by the process model, yet not in the log and sufficiently distant from any trace therein, e.g., determined using edit distance [59]. For computing precision, every distinct trace is removed from the log and an anti-alignment of equal length is generated with maximum distance. These are averaged. Likewise, we reason on language properties of analysed models, thus abstracting from the number of occurrences of a trace. However, our approach does not require the iterative scan and comparison of models excluding parts of the behaviour, thus saving on algorithmic computation time.

The Artificially Generated Negative Events technique (AGNEs) discovers process models out of event logs enriched with artificially injected negative events [22]. The assumption is that the log includes the complete set of behavioural patterns, which means that events can only be missing in a log because they are not permitted by the process. The notion of recall can then be defined as the rate of true positives over all events classified as positive, and specificity accordingly. Before the computation, a preliminary reduction of matching event sequences to single traces is conducted such that traces do not add up to the overall amount. Our definitions of precision and recall are also dual and do not depend on the number of occurrences of the same trace. However, no artificial injection of noise is required in our approach, thus reducing the bias that the alteration of the input behaviour with negative information may cause.

To evaluate their discovery algorithm, namely the Heuristic Miner, Weijters et al. [29] introduce the so-called Parsing Measure (PM), which is based on the fraction of correctly parsed traces over all traces in the input event log, similarly to the completeness measure introduced in [60]. As a derivative, the Continued Parsing Measure (CPM) provides a more fine-granular analysis, at the price of being bound to the model of the underlying Heuristic Miner. Our notion of recall for a model is also based on the measuring of the part of language not covering another behaviour. However, the measure we propose is less dependent on the recorded traces and is not based on the count of events.

The fitness measure proposed by Rozinat and van der Aalst [18] counts the number of tokens consumed and produced during the replay of traces over the Petri net model, and puts them into relation with missing tokens and tokens remaining after completion. It extends a simpler measure computed as the ratio of traces causing missing or remaining tokens defined in the same paper and named proper completion [54]. Another token-based fitness metric uses genetic process mining [61], later refined to account for trace frequency [62]. In contrast to [18,61,62], we aim at defining measures that are not tailored to specific behaviour modelling language, thus we do not rely on Petri net semantics to define recall.

The concept of alignment-based fitness introduced in [28] relies on a cost function to be specified by the user, indicating the penalty for non-synchronous moves in the replay of traces on the model. Fitness is then computed for every trace as the total cost of the optimal alignment, divided by a worst-case alignment, indicated as the one consisting of moves in the trace for every event, followed by moves in the model from the start to the end of a shortest run. Log fitness is then calculated by averaging the trace fitness values over all traces. Alignments are a valuable means to make the approach independent on the modelling language, as in the rationale of our investigation. Our technique does not require the indication of costs and it resorts on the computation of optimal or worst-case runs on the input models.

To conclude, Tax et al. [6] recently defined five requirements (there named axioms) that a precision metric should guarantee, in a strive for the general definition of fundamental properties that should be enjoyed by process mining quality measures. The authors show that neither of aforementioned simple behavioural appropriateness [18], advanced behavioural appropriateness [18], ETC precision [23,57], AGNEs specificity [22], or PCC precision [24] comply with their requirements for precision. By design, our approach fulfils all those requirements instead, as shown in Section 6.

8 Conclusion

This article proposed behavioural quotients as a means to relate the behaviours of dynamic systems, e.g., software systems. A quotient takes a language measure as a parameter, which is responsible for mapping the system’s behaviour onto the numerical domain for further comparisons with the other behaviours. Three example language measures are put forward in the article: one over finite, one over irreducible regular, and one over regular languages. The language measure over regular languages is based on the notion of topological entropy and is used to instantiate behavioural quotients into precision and recall measures for process mining. The extensive evaluation demonstrates that the proposed precision and recall can be computed in a reasonable time and qualitatively outperform all the measures for precision and fitness used in process mining.
Future work on behavioural quotients should aim at extending and improving them in several ways. First of all, behavioural quotients can be extended to behavioural representations of dynamic systems other than their languages, e.g., behavioural profiles [7, 8], declarative models [63, 64], and hybrid representations [65, 66].

Second, one can propose new language measures for instantiating behavioural quotients, and study interpretations and computational complexities of these measures.

Third, language quotients can be improved to account for multiplicities of words. This should allow addressing phenomena like repetitive occurrences of a trace in an event log. Note that the quotients proposed in this article abstract from multiplicities of words.

Fourth, language quotients can be improved to account for similar words. Note that the quotients proposed in this article are based on the exact comparisons of words. Hence, two words that differ in a single character are considered to be completely different.

Fifth, one can explore other variants of a short-circuit measure over languages, refer to Definition 5.4, e.g., $n^m(L) = (L \cup \{y\})^+$, where $L$ is a language, to study their impacts on properties of the precision and recall measures. For example one can study the sensitivity of changes of precision and recall values with respect to changes in the languages of the input automata and event logs.

Finally, one can design new quality measures that relate arbitrary numbers of behaviours, e.g., to establish a basis for comparing results of various process querying methods [67] and different behavioural representations [68, 69].

Acknowledgments
Artem Polyvyanyy and Matthias Weidlich are grateful for the support by the Universities Australia (UA) and the German Academic Exchange Service (DAAD) as part of the Joint Research Co-operation Scheme. Artem Polyvyanyy was partly supported by the Australian Research Council (RISE) and different behavioural representations of dynamic systems other than their languages, e.g., behavioural profiles [7, 8], declarative models [63, 64], and hybrid representations [65, 66].

References
Artem Polyvyanyy and Matthias Weidlich are grateful for the support by the Universities Australia (UA) and the German Academic Exchange Service (DAAD) as part of the Joint Research Co-operation Scheme. Artem Polyvyanyy was partly supported by the Australian Research Council (RISE) and different behavioural representations of dynamic systems other than their languages, e.g., behavioural profiles [7, 8], declarative models [63, 64], and hybrid representations [65, 66].

Acknowledgments
Artem Polyvyanyy and Matthias Weidlich are grateful for the support by the Universities Australia (UA) and the German Academic Exchange Service (DAAD) as part of the Joint Research Co-operation Scheme. Artem Polyvyanyy was partly supported by the Australian Research Council (RISE) and different behavioural representations of dynamic systems other than their languages, e.g., behavioural profiles [7, 8], declarative models [63, 64], and hybrid representations [65, 66].

References


[28] W. M. van der Aalst, A. Adriansyah, and B. F. van Dongen, “Replaying history on process models for conformance checking
Artem Polyvyanyy** Dr. Artem Polyvyanyy is a senior lecturer at the School of Computing and Information Systems, Melbourne School of Engineering, at the University of Melbourne, Australia. He has a strong background in Computer Science, Software Engineering, and Business Process Management from the National University of Kyiv-Mohyla Academy, Kyiv, Ukraine, and the Hasso Plattner Institute, Potsdam, Germany. He received a PhD degree (Dr. rer. nat.) in the scientific discipline of Practical Computer Science from the University of Potsdam, Germany. His personal research interests include Distributed and Parallel Systems, Concurrency Theory, Petri Nets, Formal Methods, Information Systems, Software Engineering, Workflow Management, and Business Process Management. More recently, he has conducted research on process analysis, behavior abstraction in concurrent systems, process mining, and process querying.

Andreas Solti Dr. Andreas Solti is a post doctoral researcher with the Institute for Information Business at the Vienna University of Economics and Business (Austria). His research interests are in the area process mining, process analytics, optimization, and sensor data analytics. He is also interested in adaptive learning technologies that can be supported with machine learning.

Matthias Weidlich Prof. Matthias Weidlich is a junior professor at the Department of Computer Science at Humboldt-Universität zu Berlin (HU). He leads the Process-Driven Architectures group, which is funded by the German Research Foundation (DFG) through the Emmy-Noether Programme. Before joining HU, he held positions at Imperial College London and the Technion – Israel Institute of Technology. He holds a PhD from the Hasso Plattner Institute, University of Potsdam. His research focuses on process-oriented information systems and event-driven infrastructures. He is a Junior-Fellow of the German Informatics Society (GI) and the recipient of the Berlin Young Researcher Award 2016.

Claudio Di Ciccio Dr. Claudio Di Ciccio is an Assistant Professor with the Institute for Information Business at the Vienna University of Economics and Business (Austria). His research interests include process mining, declarative process modelling, and service-oriented computing. He has published more than 50 research papers and articles, among others in ACM Transactions on Software Engineering and Methodology, IEEE Internet Computing, Information Systems, and Decision Support Systems. He obtained his Ph.D. at La Sapienza University of Rome, with a thesis on the automated discovery of flexible workflows from semi-structured text data sources. He is member of the IEEE Task Force on Process Mining.

Jan Mendling Prof. Jan Mendling is a Professor in the Department of Information Systems and Operations, Wirtschaftsuniversität Wien, Austria. He received diploma degrees from the University of Trier, a PhD degree from Wirtschaftsuniversität Wien, and held positions with Queensland University of Technology and Humboldt-Universität zu Berlin. He is also a visiting professor at University of Ljubljana and University of Liechtenstein. He has published research on business process management and information systems in various journals and is co-author of the textbooks *Fundamentals of Business Process Management* (Springer) and *Wirtschaftsinformatik* (de Gruyter, in German).
Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Polyvyanyy, A; Solti, A; Weidlich, M; Di Ciccio, C; Mendling, J

Title:
Behavioural Quotients for Precision and Recall in Process Mining

Date:
2018

Citation:

Persistent Link:
http://hdl.handle.net/11343/208876

File Description:
Submitted version