Monotone Precision and Recall Measures for Comparing Executions and Specifications of Dynamic Systems

ARTEM POLYVYANYY, The University of Melbourne, Australia
ANDREAS SOLTI, Vienna University of Economics and Business, Austria
MATTHIAS WEIDLICH, Humboldt University of Berlin, Germany
CLAUDIO DI CICCIO, Sapienza University of Rome, Italy
JAN MENDLING, Vienna University of Economics and Business, Austria

The behavioural comparison of systems is an important concern of software engineering research. For example, the areas of *specification discovery* and *specification mining* are concerned with measuring the consistency between a collection of execution traces and a program specification. This problem is also tackled in *process mining* with the help of measures that describe the quality of a process specification automatically discovered from execution logs. Though various measures have been proposed, it was recently demonstrated that they neither fulfil essential properties, such as *monotonicity*, nor can they handle infinite behaviour. In this paper, we address this research problem by introducing a new framework for the definition of behavioural quotients. We proof that corresponding quotients guarantee desired properties that existing measures have failed to support. We demonstrate the application of the quotients for capturing precision and recall measures between a collection of recorded executions and a system specification. We use a prototypical implementation of these measures to contrast their monotonic assessment with measures that have been defined in prior research.

CCS Concepts: • **Theory of computation** → *Formal languages and automata theory*; *Regular languages*; • **Software and its engineering** → **Software functional properties**; *Software verification and validation*; *Software post-development issues*; • **Mathematics of computing** → **Information theory**;

Additional Key Words and Phrases: System comparison, behavioural comparison, behavioural analysis, entropy, process mining, conformance checking, precision, recall, fitness, coverage.

Reference:
Artem Polyvyanyy, Andreas Solti, Matthias Weidlich, Claudio Di Ciccio, and Jan Mendling. 2020. Monotone Precision and Recall Measures for Comparing Executions and Specifications of Dynamic Systems. (March 2020). http://dx.doi.org/10.1145/3387909

1 INTRODUCTION

The analysis of dynamic systems is a focus of software engineering research [44, 107], and other related areas, for example business process management [37, 117], information systems [8, 16], social science [1, 26], and management science [76]. Software engineering research is primarily concerned with the analysis of behaviours captured in software systems, program specifications, and execution traces. This analysis often takes the form of behaviour comparison, with use cases ranging from specification discovery [24, 64, 69, 86] and specification mining [5, 63, 82, 88], through conformance checking between requirements and specifications [4], software evolution [27], software test coverage [11, 96], and black-box software testing [111, 118], to measurements of accuracy of the reverse-engineered specifications [62, 109]. For example, specification discovery and specification

Authors' addresses: Artem Polyvyanyy, artem.polyvyanyy@unimelb.edu.au, The University of Melbourne, Level 8, Doug McDonell Building, Parkville, VIC, 3010, Australia; Andreas Solti, solti@ai.wu.ac.at, Vienna University of Economics and Business, Austria; Matthias Weidlich, matthias.weidlich@hu-berlin.de, Humboldt University of Berlin, Germany; Claudio Di Ciccio, diciccio@di.uniroma1.it, Sapienza University of Rome, Italy; Jan Mendling, mendling@ai.wu.ac.at, Vienna University of Economics and Business, Austria.

mining study ways to infer software specifications from program executions. The quality of such inference techniques is often defined in terms of measurements of discrepancies between the execution traces used as input and the resulting program specifications. *Process mining* [98] integrates these perspectives by comparing the behaviour of a system as specified with the behaviour recorded during execution and has applications in computationally-intensive theory development [9].

A key challenge in the analysis of dynamic systems is the definition of meaningful measures that express the degree to which *different system behaviours are in line with each other*. Technically, such comparisons are formulated in a *relative* manner, defining a *quotient* of some aspect of one behaviour over the same aspect of another behaviour. For instance, the quotients of the behaviours of a system at different points in time reveal how the system has changed. In process mining, in turn, the quotient of the behaviour of a system as recorded in a log over the behaviour as specified can be used to analyse the trustworthiness of the latter. Yet, defining such quotients is challenging: A recent commentary on measures in process mining identifies a set of intuitive properties and shows that none of the available measures fulfils them [95].

We approach the above problem based on the notion of a *formal language*. This is a suitable starting point because the sequential (state-based) behaviour of a dynamic system, e.g., a software system or information system, can be modelled as a state machine or an automaton [14, 20]. An action represents an atomic unit of work, which, depending on the type of system, may be a program instruction, a Web service call, or a manual activity executed by a human agent. The behaviour of a system, therefore, can be represented by a *language* that defines a set of words over its actions. Then, each word is one possible execution (also known as a run, trace, sequence, or process) of the system. Alternatively, the comparison of the behaviours of dynamic systems was tackled in the literature using model structure [109] or abstract representations of the behaviours [115].

Behavioural comparison based on quotients of languages faces two major challenges. First and foremost, quotients have to satisfy essential properties in order to facilitate a reasonable interpretation. One such property is *monotonicity*: When increasing the amount of behaviour in the numerator of a quotient while leaving the amount of behaviour in the denominator unchanged, the quotient shall increase as well. Existing quotients as proposed, e.g., in the field of process mining [98] to compare recorded and specified behaviour, do not satisfy this well-motivated property [95, 99]. The second challenge relates to the definition of quotients in the presence of systems that describe infinite behaviours, i.e., the behaviours that consist of infinitely many words. In that case, quotients defined over standard aspects of languages, such as their cardinality, are not meaningful for behavioural comparison. In process mining, this issue has been avoided by using behavioural abstractions that capture a language by means of pairwise relations over its actions [115]. Yet, such an abstraction does not capture the complete language semantics of a system [77] and, thus, introduces a bias into the behavioural comparison. In software engineering, this issue is avoided by substituting the behaviour of a program specification with a finite collection of its simulated execution traces [62, 109]. Still, these approaches suffer from the problem of sampling the suitable finite portion of a possibly infinite behaviour [110].

In this paper, we address the problem of *how to define meaningful quotients for behavioural comparison of finite and infinite languages.* To solve this problem, we define measures that quantify the relation between the specified and recorded behaviours. Concretely, this article contributes:

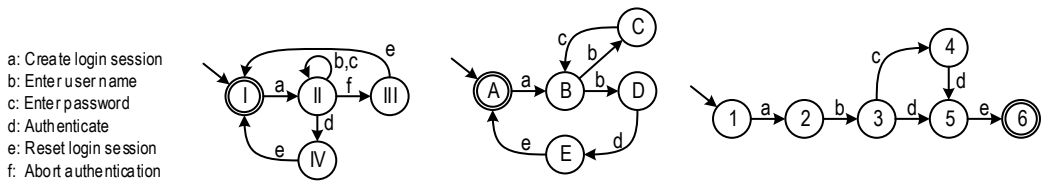
(i) A framework for the definition of behavioural quotients that guarantee desired properties.
(ii) The definition of two quotients as instantiations of the framework that are grounded in the cardinality of a language (for finite languages) and the entropy of an automaton (for finite and infinite languages).

(iii) Application of the proposed quotients to define monotone precision and recall measures between the behaviour as recorded in an execution log of a system and the behaviour captured in a specification of the system.
(iv) A publicly available implementation of the proposed precision and recall quotients.
(v) An evaluation using execution logs of real IT systems that contrasts the monotonicity of our precision and recall quotients with the 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 we address. Section 3 introduces formal preliminaries in terms of languages and automata. The framework for the definition of quotients is introduced in Section 4. This section also includes two instantiations of the framework and a discussion of formal properties of the quotients. In Section 5, we present quotients of precision and recall for comparisons of a collection of recorded system executions with a system specification. Section 6 discusses our open source implementation of quotients for comparing specifications and executions of systems. The precision and recall quotients are compared to other measures in a series of experiments using real-world data in Section 7. Section 8 discusses our contributions in the light of related work. Section 9 discusses threats to the validity of the reported conclusions, lessons we learned in the course of this work, and issues related to the adoption of the presented methods in software engineering practice. Finally, Section 10 concludes the paper.

2 BACKGROUND ON BEHAVIOURAL COMPARISON

The behaviour of dynamic systems can be captured by the help of languages over their actions. This comes with the benefit that their behavioural differences and commonalities can be analyzed by comparing the respective languages. Behavioural comparisons can be summarized using measures that quantify an aspect of a language, such as its cardinality, i.e., the number of words defined by the language. A ratio of such aspects facilitates a relative comparison of two languages by putting one behaviour is into perspective of some base behaviour. We refer to such a ratio as a *language quotient*, i.e., $(language)\ quotient := \frac{measure(language_1)}{measure(language_2)}$.



(a) Actions (b) System $\mathcal{S}_1$ (c) System $\mathcal{S}_2$ (d) System $\mathcal{S}_3$

$\mathcal{L}_1 := [\langle a,b,d,e\rangle, \langle a,b,c,b,c,d,e\rangle]$ $\mathcal{L}_2 := \mathcal{L}_1 \uplus [\langle a,b,c,c,d,e\rangle, \langle a,f,e\rangle, \langle a,f,e\rangle]$ $\mathcal{L}_3 := [\langle a,b,c,b,c,d,e\rangle, \langle a,b,b,f\rangle, \langle a,f,e\rangle]$

(e) Logs

Fig. 1. Exemplary systems and logs capturing a login process.

Example 2.1. For illustration purposes, consider the scenario of a user logging into some application. Fig. 1(a) lists the corresponding actions, such as *creating a login session* or conducting the actual *authentication*. Specific realisations of this scenario are given as finite automata in Figs. 1(b)–1(d). Albeit similar, the systems $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ define different languages over the actions, denoted by $L(\mathcal{S}_1)$, $L(\mathcal{S}_2)$, and $L(\mathcal{S}_3)$, respectively. Note that the languages of $\mathcal{S}_2$ and $\mathcal{S}_1$ are in a subset relation, i.e., it holds that $L(\mathcal{S}_2) \subset L(\mathcal{S}_1)$. Furthermore, Fig. 1(e) depicts three logs, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$, each representing recorded executions of actual login processes. Each log $\mathcal{L}$ is a multiset of sequences

over actions and, thus, also induces a language $L(\mathcal{L})$. The latter contains all words that occur at least once in the log. ⌟

The three automata from our example may represent (i) different systems, (ii) different versions of the same system, or (iii) system specifications and their implementations. In any case, it is 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 correctly and completely.

We note that similar questions emerge in the field of process mining [98], which targets the analysis of information systems based on recorded executions of a process. Given a specification and a log, process mining strives for quantifying the share of recorded behaviour that is in line with the specification (*fitness* or *recall* of the log) or the share of specified behaviour that is actually recorded (*precision* of the specification).

To address the above use cases, we essentially ask *how much one system extends the behaviour of another system.* For systems $\mathcal{S}_x$ and $\mathcal{S}_y$, such that $L(\mathcal{S}_y) \subseteq L(\mathcal{S}_x)$, we may answer this question with a quotient defined using language cardinality as a measurement function:
(*language*) *extension*$(\mathcal{S}_x, \mathcal{S}_y) := \frac{|L(\mathcal{S}_x)|}{|L(\mathcal{S}_y)|}$.

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: (*language*) *coverage*$(\mathcal{S}_x, \mathcal{S}_y) := \frac{|L(\mathcal{S}_x) \cap L(\mathcal{S}_y)|}{|L(\mathcal{S}_x)|}$.

The above quotients of language extension and coverage provide a straight-forward means for behavioural comparison of systems, specifications 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.

Example 2.2. Taking up Example 2.1, we may compute the *language extension* using cardinality as a measure for the logs $\mathcal{L}_1$ and $\mathcal{L}_2$, capturing that $L(\mathcal{L}_2)$ contains twice as many words as $L(\mathcal{L}_1)$. However, *language extension* based on cardinality is not meaningful for any pair of languages of systems $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, since $L(\mathcal{S}_1)$ and $L(\mathcal{S}_2)$ are infinite. In the same vein, computing the *language coverage* of a specification and a log, to assess the fitness of the log or the precision of the specification, is not meaningful for the systems $\mathcal{S}_1$ and $\mathcal{S}_2$, and any of the logs. ⌟

Beyond the challenge posed by infinite languages, we note that quotients have to satisfy specific properties. We illustrate these properties using the examples introduced above.

Example 2.3. The languages of automata $\mathcal{S}_2$ and $\mathcal{S}_1$ are in the subset relation, which should be reflected in the respective quotients of language extension. For example, given any log $\mathcal{L}$ such that $L(\mathcal{L}) \subseteq L(\mathcal{S}_2)$, it should hold that a quotient of $L(\mathcal{L})$ to $L(\mathcal{S}_1)$ should yield a smaller value than a quotient of $L(\mathcal{L})$ to $L(\mathcal{S}_2)$. Since language $L(\mathcal{S}_1)$ contains $L(\mathcal{S}_2)$ 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 that capture a particular aspect of languages. As we will discuss 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 as 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 specification against a log satisfies a set of well-motivated properties [95, 99].

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 should 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 $\mathcal{B}(A)$, we denote the set of all finite multisets over some set A. For some multiset $B \in \mathcal{B}(A)$, $B(a)$ 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 $B_2(a) = 2 = B_3(a)$, $B_2(b) = 1 = B_3(b)$, 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 $a \in B$; otherwise, one writes $a \notin B$. The union of two multisets C and D, denoted by $C \uplus D$, 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, $[b] \uplus B_2 = [a^2, b^2]$. Also note that $\mathcal{L}_2$ in Fig. 1(e) is the union of $\mathcal{L}_1$ and the multiset of three sequences with two instances of sequence $\langle a, f, e\rangle$; more info on sequences is provided below. The difference of two multisets C and D, denoted by $C \setminus D$, is the multiset that for each element $x \in C$ contains $max(0, C(x) - D(x))$ occurrences of x. For example, it holds that $B_3 \setminus B_2 = B_1$, and $B_3 \setminus [b] = [a, a]$. Given a multiset $B \in \mathcal{B}(A)$ over set A, by $Set(B)$ we refer to the set that contains all and only elements in B, i.e., $Set(B) := \{b \in A \mid b \in B\}$.

A *sequence* is an ordered collection of elements. By $\sigma := \langle a_1, a_2, \ldots, a_n\rangle \in A^*$, we denote a sequence over some set A of length $n \in \mathbb{N}_0$, $a_i \in A$, $i \in [1..n]$, where $[j..k] := \{x \in \mathbb{N}_0 \mid j \leq x \leq k\}$, $j, k \in \mathbb{N}_0$.[1] By $|\sigma| := n$, we denote the length of the sequence. By $\sigma_{[i]}$, $i \in [1..n]$, we refer to the i-th element of σ, i.e., $\sigma_{[i]} = a_i$. Given a sequence σ and a set K, by $\sigma|_K$, 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 $\langle \mathsf{a}, \mathsf{b}, \mathsf{d}, \mathsf{c}, \mathsf{a}\rangle|_{\{\mathsf{b},\mathsf{c}\}} = \langle \mathsf{b}, \mathsf{c}\rangle$. Given two sequences σ and σ', by $\sigma \circ \sigma'$, we denote the *concatenation* of σ and σ', i.e., the sequence obtained by appending σ' to the end of σ. For example, $\langle \mathsf{a}, \mathsf{b}, \mathsf{a}\rangle \circ \langle\rangle \circ \langle \mathsf{b}, \mathsf{a}\rangle = \langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{a}\rangle$, where $\langle\rangle$ is the empty sequence. For two sets of sequences X_1 and X_2 over A, $X_1 \circ X_2 := \{\sigma \in A^* \mid \exists\, \sigma_1 \in X_1 \exists\, \sigma_2 \in X_2 : \sigma = \sigma_1 \circ \sigma_2\}$. By $suffix(\sigma, i)$, $i \in \mathbb{N}$, we denote the suffix of σ starting from and including position i. For example, $\mathcal{L}_1$ in Fig. 1(e) contains sequences $\sigma_1 := \langle \mathsf{a}, \mathsf{b}, \mathsf{d}, \mathsf{e}\rangle$ and $\sigma_2 := \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\rangle$. It holds that $suffix(\sigma_1, 3) = \langle \mathsf{d}, \mathsf{e}\rangle$ and $suffix(\sigma_2, 6) = \langle \mathsf{d}, \mathsf{e}\rangle$.

If $\sigma := \langle a_1, a_2, \ldots, a_n\rangle \in A^*$ is a sequence over A and f is a function over A, then $f(\sigma) := \langle f(a_1), f(a_2), \ldots, f(a_n)\rangle$. Similarly, if $A' \subseteq A$, then $f(A') := \{f(a) \mid a \in A'\}$.

An *alphabet* is any nonempty finite set. The elements of an alphabet are its *labels*, or *symbols*. By Ξ, we denote a universe of symbols. For example, Fig. 1(a) specifies alphabet $\Sigma := \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}\}$. A

[1] By $\mathbb{N}$ and $\mathbb{N}_0$, we denote the set of all natural numbers excluding and including zero, respectively.

word over an alphabet is a finite sequence of its symbols. 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 [46]. Let Ξ be a universe of labels and let $\tau \in \Xi$ be a special *silent* label.

Definition 3.1 (Nondeterministic finite automaton).
A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Lambda, \delta, q_0, A)$, where Q is a finite nonempty set of *states*, $\Lambda \subset \Xi$ is a set of *labels*, such that Q and Ξ are disjoint, $\delta : Q \times (\Lambda \cup \{\tau\}) \to \wp(Q)$ is the *transition function*, where $\tau \notin Q \cup \Lambda$, $q_0 \in Q$ is the *start state*, and $A \subseteq Q$ is the *set of accept states*.[2] ⌟

An NFA induces a set of computations.

Definition 3.2 (Computation).
A *computation* of an NFA $(Q, \Lambda, \delta, q_0, A)$ is either the empty word or a word $s := \langle s_1, s_2, \ldots, s_n \rangle$, $n \in \mathbb{N}$, where every s_i is a member of $\Lambda \cup \{\tau\}$, $i \in [1\,..\,n]$, and there exists a sequence of states $q := \langle q_0, q_1, \ldots, q_n \rangle$, where every q_j is a member of the set of states Q, $j \in [1\,..\,n]$, such that for every $k \in [1\,..\,n]$ it holds that $q_k \in \delta(q_{k-1}, s_k)$. ⌟

We say that s *leads to* q_n. By convention, the empty word leads to the start state. An NFA $B := (Q, \Lambda, \delta, q_0, A)$ *accepts* a word s iff s is a computation of B that leads to an accept state q of B.

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., $L(B) := \{s \in \Lambda^* \mid \exists\, r \in (\Lambda \cup \{\tau\})^* : ((B \text{ accepts } r) \wedge (s = r|_\Lambda))\}$. ⌟

We say that B *recognises* $L(B)$. In an NFA, the transition function takes a state and label to produce the set of possible next states, while in a deterministic finite automaton the transition function takes a state and label and produces the next state.

Definition 3.4 (Deterministic finite automaton).
A *deterministic finite automaton* (DFA) is an NFA $(Q, \Lambda, \delta, q_0, A)$ such that for every state $q \in Q$ it holds that $\delta(q, \tau) = \varnothing$ and for every state $q \in Q$ and for every label $s \in \Lambda$ it 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 $\langle q_1, \ldots, q_n \rangle \in Q^*$, $n \in \mathbb{N}$, for which it holds that for every $k \in [1\,..\,n-1]$ there exists $\lambda \in \Lambda \cup \{\tau\}$ such that $q_{k+1} \in \delta(q_k, \lambda)$, $q_1 = x$, 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 $w_1, w_2 \in L$, 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 [18].

An NFA $B := (Q, \Lambda, \delta, q_0, A)$ is *τ-free* iff for all $q \in Q$ it holds that $\delta(q, \tau) = \varnothing$. By definition, every DFA is *τ-free*. Given an NFA B, one can always construct a DFA B' that recognises the language of B [46].

Example 3.1. We illustrate the above notions using the automaton in Fig. 1(c), which is defined according to our model as $\mathcal{S}_2 := (Q, \Lambda, \delta, q_0, A)$, with states $Q := \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \mathsf{E}\}$, labels $\Lambda := \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$, transition function $\delta := \{((\mathsf{A}, \mathsf{a}), \{\mathsf{B}\}), ((\mathsf{B}, \mathsf{b}), \{\mathsf{C}, \mathsf{D}\}), ((\mathsf{C}, \mathsf{c}), \{\mathsf{B}\}), ((\mathsf{D}, \mathsf{d}), \{\mathsf{E}\}), ((\mathsf{E}, \mathsf{e}), \{\mathsf{A}\})\}$, start state $q_0 := \mathsf{A}$, and accept states $A := \{\mathsf{A}\}$. This automaton is a τ-free NFA. However, a DFA that recognises the language of $\mathcal{S}_2$ may be constructed, as illustrated by Fig. 2. ⌟

[2] Given a set A, by $\wp(A)$, we denote the powerset of A.

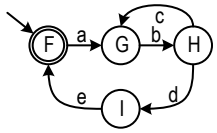


Fig. 2. DFA $\mathcal{S}_4$ that recognises the language of $\mathcal{S}_2$ in Fig. 1(c).

The discussions in Section 4 and Section 5 rely on the use of DFAs. However, software systems and their executions may induce NFA with silent transitions, as observed in the dataset used in the evaluation reported in Section 7.3. The transformation of an NFA into an equivalent DFA is an inherent step of the approach which impacts its performance. Hence, we introduce NFAs here and refer to the transformation from NFAs into DFA explicitly in Section 6.

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 the languages of systems. In Section 4.2, we demonstrate that the proposed quotients satisfy desirable properties for behavioural comparison of systems. Finally, in Section 4.3, we propose two measurement functions for instantiating language quotients, one based on the cardinality of a language and one based on its topological entropy.

4.1 Framework Definition

A behavioural comparison of systems is usually carried out based on aspects of their languages. An aspect of a language can be captured by a measure $m : \wp(\Xi^*) \to \mathbb{R}_0^+$, which is a (set) function from the set of all languages over Ξ to non-negative real numbers.[3] Two desirable properties of a measure are:

- A measure can be monotonic. A measure *m is (strictly monotonically) increasing* iff for all $U \subset \Xi^*$ 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 $m(\varnothing) = 0$.

We say that a measure over languages is a *language measure* iff it is increasing and starts at zero.[4]

A language quotient sets aspects 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:

$$quotient_m(L_1, L_2) := \frac{m(L_1)}{m(L_2)}.$$

⌟

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; examples include quotients that are defined over the intersection, union, or difference of languages, (see the notion of *language coverage*

[3]By $\mathbb{R}_0^+$, we denote the set of all non-negative real numbers.
[4]Thus, a language measure satisfies the properties of *non-negativity* and defines the empty set to be a *null set* (see [94] for details). However, it is not required to be *countable* or *finite additive*, as these properties are not exploited in the subsequent analysis of this article. Note that if a language measure m is *countably additive*, $(\Xi^*, \wp(\Xi^*), m)$ defines a *measure space*, as it is studied in mathematical analysis.

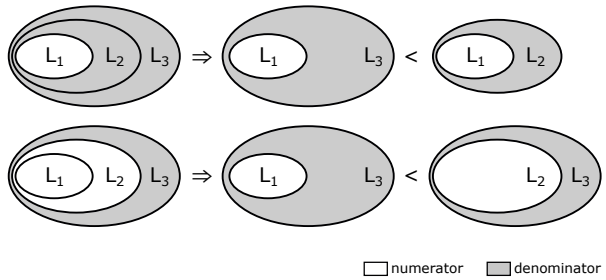


Fig. 3. Schematic representation of: Lemma 4.2 (top row) and Lemma 4.3 (bottom row).

in Section 1 for illustration). In Section 5.1, we provide further examples of quotients over the intersection of languages that are useful in the context of process mining.

4.2 Properties of Language Quotients

Language quotients enjoy useful properties that rest on the properties of a language measure. One can compare quotients with the same numerators as follows.

Lemma 4.2 (Fixed numerator quotients).
If $L_1, L_2, L_3 \subseteq \Xi^*$ are languages such that L_1 is nonempty, $L_1 \subset L_2$, and $L_2 \subset L_3$, then it holds that $quotient_m(L_1, L_3) < quotient_m(L_1, L_2)$, where m is a language measure. ⌟

Proof. Let us assume that $L_1 \neq \emptyset$, $L_1 \subset L_2$, and $L_2 \subset L_3$, but it holds that $quotient_m(L_1, L_2) \leq quotient_m(L_1, L_3)$. 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 the lemma is shown schematically in Fig. 3 (top row). If L_2 and L_3 are languages of two systems that extend the behaviour of a third system that recognises language L_1, then, using the quotients, one can conclude that the system that recognises L_3 *extends* the behaviour of the system that recognises L_1 more than does the system that recognizes L_2. The difference between the extension behaviours is captured by $quotient_m(L_1, L_3) - quotient_m(L_1, L_2)$. The meaning of the difference depends on the meaning of language measure m used to instantiate the quotients. If m measures 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.

Moreover, language quotients with the same denominators can be compared as below.

Lemma 4.3 (Fixed denominator quotients).
If $L_1, L_2, L_3 \subseteq \Xi^*$ are languages such that $L_1 \subset L_2$ and $L_2 \subset L_3$, then it holds that $quotient_m(L_1, L_3) < quotient_m(L_2, L_3)$, where m is a language measure. ⌟

Proof. Assume that $L_1 \subset L_2$ and $L_2 \subset L_3$ but it holds that $quotient_m(L_2, L_3) \leq quotient_m(L_1, L_3)$. Because m starts at zero and is increasing, it holds that $0 \leq m(L_1) < m(L_2)$. Because $m(L_3) > 0$, we reach a contradiction. ∎

The statement of the lemma 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 recognises L_2 is more complete than the implementation that recognizes L_1. In other words, L_2 has better *coverage* of the specification than L_1. The extent to which the implementation that recognises L_2 is more complete can be quantified by $quotient_m(L_2, L_3) - quotient_m(L_1, L_3)$. The meaning of the difference, again, depends on the meaning of language measure m used to instantiate the quotients.

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

Corollary 4.4 (Fixed numerator quotients). *If $L_1, L_2 \subset \Xi^*$ are languages such that $L_1 \subset L_2$, then it holds that $quotient_m(L_1, \Xi^*) < quotient_m(L_1, L_2)$, where m is a language measure.* ⌟

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

4.3 Framework Instantiations

This section proposes two language quotients, as instantiations of Definition 4.1 using specific measurement functions. Thus, these quotients have all the properties proposed in Section 4.2. The first quotient is based on the cardinality of a language, whereas the other one is grounded in the notion of topological entropy.

Cardinality quotient. As language L is a set of words, its *cardinality*, denoted by $|L|$, 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). The *cardinality quotient* of language L_1 over language L_2 is the fraction of the cardinality of L_1 over the cardinality of L_2, i.e., $quotient_{car}(L_1, L_2) := \frac{|L_1|}{|L_2|}$. ⌟

The cardinality quotient captures the ratio of the sizes of two languages. It is well-defined only for $L_2 \neq \emptyset$. Note that this is a definitional issue that may be addressed explicitly (e.g., defining $quotient_{car}(L_1, L_2) := 0$ 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 [91]. 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 2, would be equal to one for any pair of ergodic automata, such as those in Fig. 1(b) and Fig. 1(c). 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., $C_n(L) := \{x \in L \mid |x| = n\}$. Then, the *topological entropy* of L is defined as follows (see [18, 75] for details)[5]:

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

Topological entropy characterises the complexity of a language and is closely related to the properties of the DFAs that recognise this language. For a DFA $B := (Q, \Lambda, \delta, q_0, A)$, with $C_n(B)$, $n \in \mathbb{N}_0$, we denote the set of all the words in $L(B)$ of length n, i.e., $C_n(B) := \{x \in L(B) \mid |x| = n\}$. Then, the topological entropy of B is defined as the topological entropy of the language that it recognises [18]:

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

[5]Given a sequence (x_n), $\limsup_{n \to \infty} x_n$ is the *limit superior* of (x_n) and is defined by $\inf\{\sup\{x_m \mid m \geq n\} \mid n \geq 0\}$.

(a) B_1 (b) B_2 (c) $\hat{B}_1$ (d) $\hat{B}_2$

Fig. 4. Four DFAs.

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 $G := \{g_{ij}\}$, $i, j \in [1\,..\,n]$, $n \in \mathbb{N}$, i.e., $g_{ij} \geq 0$ for all $i, j \in [1\,..\,n]$. The adjacency matrix of a DFA $(Q, \Lambda, \delta, q_0, A)$, where $Q := \{q_0, q_1, ..., q_n\}$, $n \in \mathbb{N}_0$, is a square matrix $G := \{g_{ij}\}$, $i, j \in [1\,..\,|Q|]$, such that $g_{ij} := |\{q_j \in \delta(q_i, \lambda) \mid \lambda \in \Lambda\}|$, for all $i, j \in [1\,..\,|Q|]$.[6] The topological entropy of an ergodic 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 [18]. 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 λ of the adjacency matrix of B [89, Theorem 1.5]. The relation between the entropy of a language and the entropy of an ergodic DFA recognising this language, as outlined above, is important for computational reasons, as it provides us with a straight-forward approach to compute the entropy of a language, via the Perron-Frobenius theory.

Topological entropy is *not* an increasing measure over regular languages. It is neither an increasing measure over irreducible regular languages. Indeed, for two ergodic automata B_1 and B_2 shown in Fig. 4(a) and Fig. 4(b), respectively, it holds that $L(B_1) \subset L(B_2)$ and $ent(L(B_1)) = 1.0 = ent(L(B_2))$; note that logarithm base two was used to compute the entropy.

Let $U \subset V$ be two regular languages over alphabet $\Psi \subset \Xi$ such that $U \subset V$. Let $\hat{U}$ and $\hat{V}$ denote the languages $(U \circ \{\langle\chi\rangle\})^* \circ U$ and $(V \circ \{\langle\chi\rangle\})^* \circ V$, $\chi \in \Xi \setminus \Psi$, respectively. We say that $\hat{U}$ *and* $\hat{V}$ *are the results of short-circuiting* U *and* V *with* χ. Note that given an automaton B it is straight-forward to construct an automaton that recognizes the short-circuited version of $L(B)$. This can be achieved by inserting fresh transitions in B, each labelled with χ, from each accept state of B to its start state to obtain automaton $\hat{B}$. For example, automata $\hat{B}_1$ and $\hat{B}_2$ from Fig. 4(c) and Fig. 4(d), respectively, recognise the short-circuited versions of languages $L(B_1)$ and $L(B_2)$, where B_1 and B_2 are shown in Fig. 4(a) and Fig. 4(b). Note that any augmented in this way automaton is guaranteed to be ergodic and, thus, the language such an automaton recognises is irreducible.

Let $(u_n)_{n=1}^{\infty}$ and $(v_n)_{n=1}^{\infty}$ be two sequences such that:

$$u_n := \frac{\log|C_n(\hat{U})|}{n} \qquad \text{and} \qquad v_n := \frac{\log|C_n(\hat{V})|}{n}.$$

For every $n \in \mathbb{N}_0$ it holds that $C_n(\hat{U}) \subseteq C_n(\hat{V})$ because $\hat{U} \subset \hat{V}$. Let $w_u \in U$ and $w_v \in V \setminus U$ be two words. Let $(\alpha_i)_{i=1}^{\infty}$ and $(\beta_i)_{i=1}^{\infty}$ be two sequences such that $\alpha_1 := |w_v|$, $\beta_1 := |w_u|$,

$$\alpha_j := \alpha_{j-1} + \frac{LCM(|w_u|+1, |w_v|+1)}{|w_u|+1}, \text{ and } \beta_j := \beta_{j-1} + \frac{LCM(|w_u|+1, |w_v|+1)}{|w_v|+1},\ j > 1.^{7}$$

Then, for any $k \in \mathbb{N}$ it holds by induction that $|(w_u \circ \langle\chi\rangle)^{\alpha_k} \circ w_u| = |(w_v \circ \langle\chi\rangle)^{\beta_k} \circ w_v|$.[8] Note that $w_u \in V$ and $w_v \notin U$. Hence, for any $n \in \mathbb{N}$, $\sup\{u_m \mid m \geq n\} < \sup\{v_m \mid m \geq n\}$ and, consequently:

$$\inf\{\sup\{u_m \mid m \geq n\} \mid n \geq 0\} \ < \ \inf\{\sup\{v_m \mid m \geq n\} \mid n \geq 0\}.$$

[6] Recall from Section 3.2 that every DFA is τ-free.

[7] By $LCM(a, b)$ we denote the the *least common multiple* of two integers a and b.

[8] Given a word w, by w^k, $k \in \mathbb{N}$, we denote concatenation of k instances of w, e.g., $(\mathsf{ab})^3 = \mathsf{ababab}$.

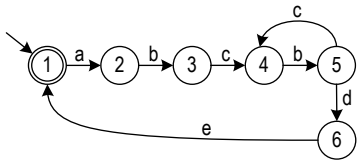


Fig. 5. DFA $\mathcal{S}_5$.

That is, it holds that $ent(\hat{U}) < ent(\hat{V})$. As a consequence of this fact, it holds that $ent(\hat{B}_1) = 1.2925$ and $ent(\hat{B}_2) = 1.585$, i.e., $ent(\hat{B}_1) < ent(\hat{B}_2)$, where $\hat{B}_1$ and $\hat{B}_2$ are automata shown in Fig. 4(c) and Fig. 4(d), respectively; again, logarithm base two was used to compute the entropy.

Given a measure over languages, one can obtain a corresponding short-circuit measure as follows.

Definition 4.6 (Short-circuit measure).
A *short-circuit measure* over languages over alphabet $\Psi \subset \Xi$ induced by a measure over languages $m : \wp(\Xi^*) \to \mathbb{R}_0^+$ is the (set) function $m^\bullet : \wp(\Psi^*) \to \mathbb{R}_0^+$ defined by $m^\bullet(L) := m((L \circ \{\langle\chi\rangle\})^* \circ L)$, where L is a language over Ψ, i.e., $L \subseteq \Psi^*$, and $\chi \in \Xi \setminus \Psi$ is a short-circuit symbol. ⌟

By $eig(L)$, where L is a language, we denote the Perron-Frobenius eigenvalue of the adjacency matrix of a DFA that recognises L, and call it the *eigenvalue measure* of L. We also say that $eig(L)$ is the *eigenvalue* of L. We accept that the adjacency matrix of a DFA that induces the empty language is the zero square matrix of order one. Hence, the above definitions and observations lead to the next conclusion.

Lemma 4.7 (Language measures).
The short-circuit topological entropy ($ent^\bullet$) and the short-circuit eigenvalue measure ($eig^\bullet$) are language measures, i.e., they are increasing and start at zero. ⌟

The fact that the short-circuit eigenvalue measure is increasing follows immediately from the facts that (i) the short-circuit topological entropy is a language measure and (ii) the logarithm is a strictly increasing function. Finally, to avoid decisions of which logarithm base to use when computing the entropy, in what follows, we define and use language quotients induced by the eigenvalue measure.

Definition 4.8 (Eigenvalue quotient).
Given two regular languages L_1 and L_2, the *eigenvalue quotient* of L_1 over L_2 is the fraction of the short-circuit eigenvalue measure of L_1 over the short-circuit eigenvalue measure of L_2, i.e.,

$$quotient_{eig^\bullet}(L_1, L_2) := \frac{eig^\bullet(L_1)}{eig^\bullet(L_2)}.$$

⌟

Example 4.1. Consider automata $\mathcal{S}_1$, $\mathcal{S}_4$, and $\mathcal{S}_5$ in Fig. 1(b), Fig. 2, and Fig. 5. These three automata are ergodic and it holds that $L(\mathcal{S}_5) \subset L(\mathcal{S}_4)$ and $L(\mathcal{S}_4) \subset L(\mathcal{S}_1)$. It holds that $quotient_{eig^\bullet}(L(\mathcal{S}_4), L(\mathcal{S}_1)) = 0.6$, $quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_1)) = 0.5525$, $quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_4)) = 0.9208$, and $quotient_{eig^\bullet}(L(\mathcal{S}_5), \Phi^*) = 0.2321$, where Φ is the set $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$. Indeed, it holds that:

(i) $quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_1)) < quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_4))$ — see Lemma 4.2;
(ii) $quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_1)) < quotient_{eig^\bullet}(L(\mathcal{S}_4), L(\mathcal{S}_1))$ — see Lemma 4.3; and
(iii) $quotient_{eig^\bullet}(L(\mathcal{S}_5), \Phi^*) < quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_4))$ — see Corollary 4.4.

To show that $quotient_{eig^\bullet}(L(\mathcal{S}_5), L(\mathcal{S}_4))$ indeed equals to 0.9208, Fig. 6(a) and Fig. 6(b) show adjacency matrices of $\mathcal{S}_4$ and $\mathcal{S}_5$, respectively. Note that the Perron-Frobenius eigenvalue of the matrix in Fig. 6(a) is 1.5129, while the Perron-Frobenius eigenvalue of the matrix in Fig. 6(b) is 1.3931. ⌟

	F	G	H	I
F	1	1	0	0
G	0	0	1	0
H	0	1	0	1
I	1	0	0	0

(a)

	1	2	3	4	5	6
1	1	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	1	0	1
6	1	0	0	0	0	0

(b)

Fig. 6. Adjacency matrices of the short-circuit, i.e., after insertions of the χ transitions, versions of DFAs from (a) Fig. 2 and (b) Fig. 5.

5 PRECISION AND RECALL

Language quotients provide a general means for behavioural comparison. To demonstrate the use of the quotients, this section proposes and discusses their application in process mining [98]. One of the problems studied in process mining is the problem of *process discovery*. Given a log of recorded executions of a system, a discovery technique constructs a specification of the system that "represents" the behaviour captured in the log. As a system may execute same sequences of actions multiple times, its log is a multiset of words that encode the executions.

Definition 5.1 (Log).
A *log* is a finite multiset over a language. ⌟

An element of a log is a *trace*, whereas an element of a trace is an *event* of the trace. Given a log $\mathcal{L}$, $L(\mathcal{L}) := Set(\mathcal{L})$ is the *language* of $\mathcal{L}$.

Example 5.1. For example, logs $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$, listed in Fig. 1(e) contain two, five, and three traces, respectively. Note that $\mathcal{L}_2$ contains trace $\langle a, f, e\rangle$ twice, which denotes that this sequence of actions was recorded in the log two times. ⌟

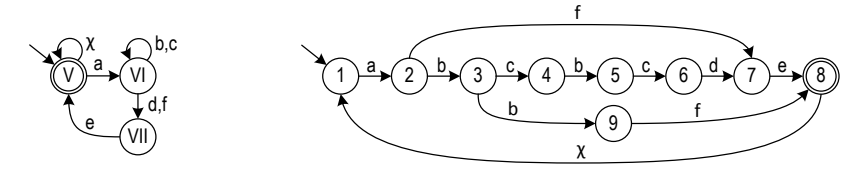
The quality of a generated process specification is typically evaluated using *precision*, *fitness* (a specific type of recall), *simplicity*, and *generalization* [98]. We use the framework of behavioural quotients to define precision and recall of specifications w.r.t. logs (Section 5.1). We demonstrate that our precision and recall quotients satisfy important requirements for precision and recall measures (Section 5.2).

5.1 Definition of Precision and Recall

This section proposes two quotients for comparing behaviours captured in a given log and DFA, namely *precision* and *recall* of the DFA w.r.t. the 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 specifications discovered from 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 a log and a DFA, we propose to measure how precisely a DFA (specification) describes a log as the fraction of executions recorded in the log and specified in the DFA over all the executions (of which there can be infinitely many) specified in the DFA.

Definition 5.2 (Precision of DFA w.r.t. log).
Given a log $\mathcal{L}$ and a DFA B, the *precision* of B w.r.t. $\mathcal{L}$ induced by a language measure m is denoted



	V	VI	VII
V	1	1	0
VI	0	2	2
VII	1	0	0

(c)

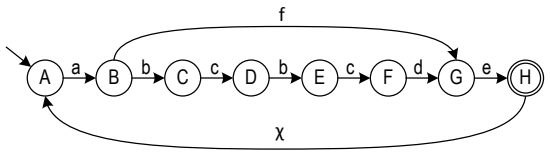


Fig. 7. Three DFAs and the adjacency matrix of the automaton in (a).

by $precision_m(B, \mathcal{L})$ 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., $precision_m(B, \mathcal{L}) := quotient_m(L(B) \cap L(\mathcal{L}), L(B))$. ⌟

Precision is the ratio of the measure of traces of the log that are also computations of the DFA (specified and recorded behaviour) to the measure of all the computations of the DFA (specified behaviour).

Example 5.2. The precision of automaton $\mathcal{S}_3$ in Fig. 1(d) w.r.t. log $\mathcal{L}_2$ in Fig. 1(e) induced by the cardinality of a language is computed as follows: $precision_{car}(\mathcal{S}_3, \mathcal{L}_2) = \frac{|L(\mathcal{S}_3) \cap L(\mathcal{L}_2)|}{|L(\mathcal{S}_3)|} = \frac{1}{2}$; the languages of $\mathcal{S}_3$ and $\mathcal{L}_2$ share one word, $\sigma := \langle \mathsf{a}, \mathsf{b}, \mathsf{d}, \mathsf{e} \rangle$, while the language of $\mathcal{S}_3$ has two words: σ and $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e} \rangle$. ⌟

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 a log and a DFA, we measure how well the DFA captures the behaviour of the log as the fraction of executions recorded in the log and specified in the DFA over all the behaviour recorded in the log.

Definition 5.3 (Recall of DFA w.r.t. log).
Given a 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 $recall_m(B, \mathcal{L})$ 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., $recall_m(B, \mathcal{L}) := quotient_m(L(B) \cap L(\mathcal{L}), L(\mathcal{L}))$. ⌟

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

Example 5.3. For example, the recall of automaton $\mathcal{S}_3$ in Fig. 1(d) w.r.t. log $\mathcal{L}_2$ in Fig. 1(e) induced by the cardinality of a language is computed as follows: $recall_{car}(\mathcal{S}_3, \mathcal{L}_2) = \frac{|L(\mathcal{S}_3) \cap L(\mathcal{L}_2)|}{|L(\mathcal{L}_2)|} = \frac{1}{4}$. This result is easy to verify by checking that the language of $\mathcal{L}_2$ consists of four words. ⌟

The notions of fitness and recall of a DFA w.r.t. a log take a language measure as a parameter. The language of a 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, as proposed above. The language of a DFA is, however, often infinite. To overcome this limitation, we suggest instantiating the precision and recall with the short-circuit eigenvalue measure, as illustrated below; see Section 4.3 for details on the short-circuit eigenvalue measure.

Example 5.4. Consider automaton $\mathcal{S}_1$ in Fig. 1(b) and log $\mathcal{L}_3$ in Fig. 1(e). Fig. 7(a) and Fig. 7(b) show automata with languages $(L(\mathcal{S}_1) \circ \{\langle\chi\rangle\})^* \circ L(\mathcal{S}_1)$ and $(L(\mathcal{L}_3) \circ \{\langle\chi\rangle\})^* \circ L(\mathcal{L}_3)$, respectively, whereas Fig. 7(d) shows an automaton with language $((L(\mathcal{S}_1) \cap L(\mathcal{L}_3)) \circ \{\langle\chi\rangle\})^* \circ (L(\mathcal{S}_1) \cap L(\mathcal{L}_3))$. It is easy to see that given a DFA $B := (Q, \Lambda, \delta, q_0, A)$, it holds that $L(B') = (L(B) \circ \{\langle\chi\rangle\})^* \circ L(B)$, where $B' := (Q, \Lambda \cup \{\chi\}, \delta \cup (A \times \{q_0\}), q_0, A)$. Note that the automaton in Fig. 7(a) was obtained from the automaton in Fig. 1(b) using this simple transformation and subsequent minimization [45]. Such minimization is possible because any automaton with the language of interest, in this case $(L(\mathcal{S}_1) \circ \{\langle\chi\rangle\})^* \circ L(\mathcal{S}_1)$, suffices. Fig. 7(c) shows the adjacency matrix of the automaton in Fig. 7(a). The Perron-Frobenius eigenvalue of this matrix is 2.521. The Perron-Frobenius eigenvalues of the adjacency matrices of automata in Fig. 7(b) and Fig. 7(d) are 1.226 and 1.128, respectively. Thus, it holds that $precision_{eig^\bullet}(\mathcal{S}_1, \mathcal{L}_3) = 0.447$ and $recall_{eig^\bullet}(\mathcal{S}_1, \mathcal{L}_3) = 0.92$.

Table 1 lists all the precision and recall values induced by $eig^\bullet$ for each of the three DFAs in Figs. 1(b)–1(d) w.r.t. every log in Fig. 1(e). The presented values obey all the properties discussed in Section 4.2 and in the subsequent section.

Table 1. Precision and recall values.

Automaton	Log	Precision	Recall
$\mathcal{S}_1$	$\mathcal{L}_1$	0.442	1.0
$\mathcal{S}_1$	$\mathcal{L}_2$	0.506	1.0
$\mathcal{S}_1$	$\mathcal{L}_3$	0.447	0.92
$\mathcal{S}_2$	$\mathcal{L}_1$	0.661	0.897
$\mathcal{S}_2$	$\mathcal{L}_2$	0.661	0.784
$\mathcal{S}_2$	$\mathcal{L}_3$	0	0
$\mathcal{S}_3$	$\mathcal{L}_1$	0.881	0.897
$\mathcal{S}_3$	$\mathcal{L}_2$	0.881	0.784
$\mathcal{S}_3$	$\mathcal{L}_3$	0	0

5.2 Properties of Precision and Recall

This section discusses important properties of precision and recall quotients, as defined in Section 5.1, instantiated with some language measure, e.g., the short-circuit eigenvalue measure ($eig^\bullet$) from Section 4.3. As the precision and recall quotients are language quotients, they inherit all the properties discussed in Section 4.2. Next, we present these properties for the precision quotient; note that one can derive corresponding properties for the recall quotient analogously.

Proposition 5.4 (Precision monotonicity over designs). *Let $\mathcal{L}$ be an event log and let B_1 and B_2 be two DFAs such that $L(\mathcal{L}) \subset L(B_1)$ and $L(B_1) \subset L(B_2)$. Let m be a language measure over regular languages. Then, it holds that $precision_m(B_2, \mathcal{L}) < precision_m(B_1, \mathcal{L})$.*

Proposition 5.4 follows from Lemma 4.2 by accepting that $L_1 = L(\mathcal{L})$, $L_2 = L(B_1)$, and $L_3 = L(B_2)$.

Proposition 5.5 (Precision monotonicity over executions). *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two event logs and let B be a DFA such that $L(\mathcal{L}_1) \subset L(\mathcal{L}_2)$ and $L(\mathcal{L}_2) \subset L(B)$. Let m be a language measure over regular languages. Then, it holds that $precision_m(B, \mathcal{L}_1) < precision_m(B, \mathcal{L}_2)$.*

Proposition 5.5 follows from Lemma 4.3 by accepting that $L_1 = L(\mathcal{L}_1)$, $L_2 = L(\mathcal{L}_2)$, and $L_3 = L(B)$.

Proposition 5.6 (Precision minimality). *Let $\mathcal{L}$ be an event log and let B be a DFA such that $L(\mathcal{L}) \subset L(B)$ and $L(B) \subset \Phi^*$, $\Phi^* \subset \Xi^*$. Let m be a language measure over regular languages. Then, it holds that $precision_m(\Phi^*, \mathcal{L}) < precision_m(B, \mathcal{L})$.*

Proposition 5.6 follows from Corollary 4.4 by accepting that $L_1 = L(\mathcal{L})$ and $L_2 = L(B)$.

Next, we present further properties specific for the precision and recall of a DFA w.r.t. a log. First, precision and recall take values from the interval that contains zero and one.

Proposition 5.7 (Precision interval). *Given a log $\mathcal{L}$, a DFA B, such that $L(B) \neq \emptyset$, and a language measure m over regular languages, it holds that $0 \leq precision_m(B, \mathcal{L}) \leq 1$.*

Proposition 5.7 follows from Definition 5.2 and the fact that m is a language measure over regular languages. Indeed, it holds that $L(B) \cap L(\mathcal{L}) \subseteq L(B)$ and, thus, $m(L(B) \cap L(\mathcal{L})) \leq m(L(B))$; note that m is an increasing measure.

Proposition 5.8 (Recall interval).
Given a log $\mathcal{L}$, such that $L(\mathcal{L}) \neq \emptyset$, a DFA B, and a language measure m over regular languages, it holds that $0 \leq recall_m(B, \mathcal{L}) \leq 1$.

Proposition 5.8 holds because of Definition 5.3, the facts that $L(B) \cap L(\mathcal{L}) \subseteq L(\mathcal{L})$, and, again, because m is an increasing measure.

Second, precision and recall equal to one when the languages of a given DFA and log are in the containment relation.

Proposition 5.9 (Maximal precision).
Given a log $\mathcal{L}$, a DFA B, such that $L(B) \neq \emptyset$, and a language measure m over regular languages, $L(B) \subseteq L(\mathcal{L})$ iff $precision_m(B, \mathcal{L}) = 1$.

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

Proposition 5.10 (Maximal recall).
Given a log $\mathcal{L}$, such that $L(\mathcal{L}) \neq \emptyset$, a DFA B, and a language measure m over regular languages, $L(\mathcal{L}) \subseteq L(B)$ iff $recall_m(B, \mathcal{L}) = 1$.

The proof of Proposition 5.10 follows the structure of the proof of Proposition 5.9 but swaps the roles of the languages of B and $\mathcal{L}$.

Third, precision and recall both equal to one iff the languages of the DFA and log are identical.

Corollary 5.11 (Maximal precision and recall).
Given a log $\mathcal{L}$, $L(\mathcal{L}) \neq \emptyset$, a DFA B, $L(B) \neq \emptyset$, and a language measure m over regular languages, $L(B) = L(\mathcal{L})$ iff $precision_m(B, \mathcal{L}) = 1$ and $recall_m(B, \mathcal{L}) = 1$.

Corollary 5.11 follows immediately from Proposition 5.9 and Proposition 5.10.

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

Proposition 5.12 (Minimal precision).
Given a log $\mathcal{L}$, a DFA B, such that $L(B) \neq \emptyset$, and a language measure m over regular languages, $L(B) \cap L(\mathcal{L}) = \emptyset$ iff $precision_m(B, \mathcal{L}) = 0$.

If $L(B) \cap L(\mathcal{L}) = \emptyset$, then $precision_m = m(\emptyset)/m(B) = 0$, as m starts at zero. Conversely, if $precision_m(B, \mathcal{L}) = 0$, then $m(L(B) \cap L(\mathcal{L})) = 0$. Then, $L(B) \cap L(\mathcal{L}) = \emptyset$ because m starts at zero and is increasing.

Proposition 5.13 (Minimal recall).
Given a log $\mathcal{L}$, such that $L(\mathcal{L}) \neq \emptyset$, a DFA B, and a language measure m over regular languages, $L(B) \cap L(\mathcal{L}) = \emptyset$ iff $recall_m(B, \mathcal{L}) = 0$.

The proof of Proposition 5.13 follows the structure of the proof of Proposition 5.10 but swaps the roles of the languages of B and $\mathcal{L}$.

6 IMPLEMENTATION

The eigenvalue-based recall and precision measures for comparing specification and executions have been implemented and are publicly available.[9] Algorithm 1 summarizes the steps for computing the measures in pseudocode. As input, the algorithm takes two specifications. One specification describes a collection of "retrieved" executions, while the other captures "relevant" executions. For example, in the context of process mining, one can see executions encoded in an event log as relevant, i.e., those that encode valuable information, while executions captured by the model discovered from the event log as retrieved, i.e., constructed from the event log.

Algorithm 1: PrecisionAndRecallForSpecificationsOfDynamicSystems

Input: Two NFAs *ret* and *rel* describing retrieved and relevant executions, respectively.
Output: A pair $(prec, rec)$, where *prec* and *rec* are, respectively, precision and recall for *ret* and *rel*.

```
1  // Construct deterministic versions of ret and rel
2  if ¬IsDeterministic(ret) then dRet ← Determinize(ret) else dRet ← ret;
3  if ¬IsDeterministic(rel) then dRel ← Determinize(rel) else dRel ← rel;
4  mRet ← Minimize(dRet);                                  /* Minimize automaton dRet */
5  mRel ← Minimize(dRel);                                  /* Minimize automaton dRel */
6  scRet ← ShortCircuit(mRet);                          /* Short-circuit automaton mRet */
7  scRel ← ShortCircuit(mRel);                          /* Short-circuit automaton mRel */
8  intersection ← Intersection(mRet, mRel);      /* Construct intersection of mRet and mRel */
9  scIntersection ← ShortCircuit(intersection);        /* Short-circuit automaton intersection */
10 // Compute Perron-Frobenius eigenvalues of the adjacency matrices of automata
11 eigRet ← PerronFrobenius(scRet);
12 eigRel ← PerronFrobenius(scRel);
13 eigIntersection ← PerronFrobenius(scIntersection);
14 return (eigInersection/eigRet, eigInersection/eigRel);
```

Lines 2 and 3 of the algorithm ensure that *dRet* and *dRel* are *deterministic versions* of *ret* and *rel*, respectively; that is *dRet* and *dRel* are DFAs that recognise the languages $L(ret)$ and $L(rel)$, respectively. Given a τ-free NFA, a DFA that recognises the language of the NFA always exists [46] and can be constructed using the Rabin-Scott powerset construction method [85], which has the worst-case time complexity of $O(2^n)$ where n is the number of states in the NFA [71]. However, in practice, the DFA constructed from the NFA has about as many states as the NFA, but often more transitions [46]. If an NFA is not τ-free, one still can always construct a DFA that recognises the language of the NFA [46]. The construction extends the powerset construction to account for silent transitions. Hence, at lines 2 and 3 of Algorithm 1, both functions IsDeterministic and Determinize take an NFA as input. Function IsDeterministic returns *true* if the input NFA is a DFA, and otherwise, returns *false*, while function Determinize constructs and returns a deterministic version of the input NFA. In our tools, function Determinize implements the extended version of the Rabin-Scott powerset construction from [46] that constructs τ-free automata.

Lines 4 and 5 of the algorithm construct the minimal versions of the DFAs *dRet* and *dRel*, respectively. For every DFA A, there exists a unique (up to isomorphism) DFA with a minimum

[9]The source code for computing the measures and for performing the experiments reported in Section 7 is available at https://github.com/andreas-solti/eigen-measure. In addition, as part of the jBPT library [80] (refer to https://github.com/jbpt/codebase, `jbpt-pm` module), we develop and maintain a tool with a command-line interface to compute measures that compare specifications of dynamic systems. As of today, the tool supports the computation of measures presented in this work and the measures based on the partial matching of executions presented in [78].

number of states that recognises the language of A, called the *minimal version* of A [46]. There exist several algorithms that, given a DFA, construct its minimal version. For example, the worst-case time complexity of the algorithm by Hopcroft [45] is $O(nm \log(m))$, where n is the number of states and m is the size of the alphabet. Function Minimize used at lines 4 and 5 takes a DFA as input and returns its minimal version. In our tools, function Minimize implements the algorithm by Hopcroft. Note that the minimization step can be skipped as the computation of the topological entropy does not require a DFA to be minimal (see Section 4.3). While performing the scalability evaluation reported in Section 7.3, we noticed that it is faster to minimize a DFA and then compute the Perron-Frobenius eigenvalue of its adjacency matrix than to compute the eigenvalue of the original DFA. A detailed study of this phenomenon, despite important, is out of the scope of this paper. The computation times reported in Table 5 include the minimization times.

Next, lines 6–9 of Algorithm 1 construct *short-circuit versions* of automata *mRet* (line 6), *mRel* (line 7), and the intersection *intersection* of *mRet* and *mRel* (line 9) constructed at line 8. Given a DFA, its short-circuit version is obtained by adding, for each accept state a, a short-circuit transition from a to the start state. All the short-circuit transitions have a dedicated short-circuit label (χ), which is not in the set of labels of the original automaton. It is easy to see that the short-circuit version of a DFA is deterministic. Function ShortCircuit at lines 6, 7, and 9 of the algorithm implements the above construction. The intersection of regular languages is a well-known operation in automata theory and is implemented in function Intersection at line 8 of the algorithm. Its time complexity is $O(nm)$, where n and m are the numbers of states in the intersected automata [46].

Lines 11–13 of Algorithm 1 compute Perron-Frobenius eigenvalues [18] of the adjacency matrices of *scRet* (line 11), *scRel* (line 12), and *scIntersection* (line 13). Function PerronFrobenius used at lines 11–13 of the algorithm takes a DFA as input, constructs its adjacency matrix, and returns a largest eigenvalue of the matrix. Note that, typically, the adjacency matrix of a DFA is rather sparse. Thus, we can handle very large DFAs on personal computers and compute eigenvalues of their adjacency matrices with the help of memory-friendly sparse data structures.

We use the Java library `Matrix Toolkit Java (MTJ)` that relies on the low level numerical Fortran-based libraries in `ARPACK` [59] to compute eigenvalues. The available version of the `MTJ` library was capable of handling symmetric matrices in `ARPACK`. Note that the adjacency matrices of automata are usually not symmetric. Thus, we extended `MTJ` to expose `dnaupd` and `dneupd` routines of `ARPACK` to compute eigenvalues of general matrices. Our extension for computing a largest eigenvalue of a non-symmetric matrix is publicly available.[10] The underlying technique for computing a largest eigenvalue of a matrix is called "implicitly restarted Arnoldi iterations" [59] and has a low polynomial time complexity. Note that the numerical methods for computing an eigenvalue of a general matrix converge but provide no guarantees of convergence in a fixed number of iterations. Thus, we set the threshold of 300 000 as the maximum number of allowed iterations for practical reasons. 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" [58]. In the rare cases of non-convergence of the computation, we use the estimated value of the eigenvalue obtained at the end of the computation. The proposed method is not tied to the ARPACK implementation for computing a largest eigenvalue of a matrix. Thus, the use of more recent results in eigenvalue computation [36] and parallel algorithms for eigenvalue computation [48] may improve the overall performance of our tools.

Finally, line 14 of Algorithm 1 returns a pair with the precision and recall for the input automata as the first and second element, respectively, computed as quotients of the corresponding eigenvalues. To compute precision and recall of a system with respect to a given event log, i.e., the quotients

[10] The source code is available at https://github.com/andreas-solti/matrix-toolkits-java and the Maven Central Repository.

presented in Section 5, one can invoke Algorithm 1 with NFA *ret* encoding the system and NFA *rel* encoding the event log. Indeed, one can also invoke Algorithm 1 with two NFAs that describe two systems to estimate the language coverage.

The worst time complexity of Algorithm 1 is dominated by the exponential worst time complexity of the NFA determinization at lines 2 and 3. Note, however, that in practice the automata generated from software code or business process models are readily deterministic. Besides, it is easy to construct a DFA that encodes an event log, for example, as a prefix tree.

7 EXPERIMENTAL EVALUATION

The goal of the evaluation reported in this section is to demonstrate that the proposed eigenvalue-based measures for comparing specifications and collections of executions of systems advance the state-of-the-art and can be readily applied in practice. This is achieved by answering the below research questions; Tables 2 and 3 list the approaches considered in our comparative evaluation, they are discussed in detail in Section 8.

Short label	Full name and reference
advBehAppropriateness	Advanced behavioural appropriateness [87]
alignmentPrecision	Alignment-based precision [3]
antiAlignPrecision	Anti-alignments precision [102]
bestAlignPrecision	Best optimal-alignments precision [2]
negativeEventPrecision	AGNEs specificity [40]
oneAlignPrecision	One optimal-alignment precision [2]
precisionEig	Eigenvalue-based precision (this paper)
precisionETC	ETC precision [72]
projectedPrecision	PCC precision [57]
simpleBehAppropriateness	Simple behavioural appropriateness [87]

Table 2 Precision measures.

Short label	Full name and reference
alignmentFitness	Alignment-based fitness [97]
negativeEventRecall	AGNEs recall [40]
tokenBasedFitness	Token-based fitness [87]
parsingMeasure	Continued parsing measure [116]
projectedRecall	PCC recall [57]
properCompletion	Proper completion [87]
recallEig	Eigenvalue-based recall (this paper)

Table 3 Fitness (recall) measures.

RQ1: Are state-of-the-art precision and recall measures for process mining monotone?
RQ2: Are the eigenvalue-based quotients applicable for comparing the behaviours of two systems?
RQ3: Is the computation of eigenvalue-based quotients feasible for practical applications?

To answer RQ1, we studied which state-of-the-art precision and recall measures in process mining fulfill Lemmata 4.2 and 4.3; we assume for this evaluation that specifications used in comparisons describe bounded systems, i.e., systems that induce finite collections of reachable states. In Section 7.1, we give a negative answer to RQ1 for the state-of-the-art precision measures; for each evaluated measure, we present at least one example that violates the property of monotonicity.

To answer RQ2, we compare specifications of software systems studied in [39] with specifications discovered from their sample executions by computing language coverage values. We argue that such comparisons can be used to reveal insights on the quality of automated algorithms for discovering system specifications, see Section 7.2, within reasonable time bounds, see Section 7.3.

Finally, to answer RQ3, we measured the wall-clock time of computing eigenvalue-based quotients for logs and specifications from real-world and synthetic datasets. First, we computed the eigenvalue-based precision and recall for fifteen real-world event logs and specifications automatically discovered from these logs. The logs are publicly available[11] and encode executions of real-world IT systems executing business processes with real customers. We observed that for most input pairs, each composed of a specification and a log, computations of both precision and recall measures are accomplished within ten minutes, and often much faster. Second, we report the values and computation times of the eigenvalue-based quotients for inputs taken from a collection

[11]Event logs are published at: https://data.4tu.nl/repository/collection:event_logs_real

of real-world specifications, corresponding collections of sample executions of controlled sizes, and specifications automatically discovered from the sample executions.[12]

To perform the experiments, we used our implementation of the eigenvalue-based quotients, described in Section 6, and relied on the Comprehensive Benchmark Framework (CoBeFra) [105] to compute other precision and recall measures.

7.1 Comparing Executions and Specifications: Monotonicity of Precision and Recall

For a given log, a monotonic precision measure should *always* decrease when fresh behaviour is added to the specification. Conversely, a monotonic precision measure should *always* increase when the excess behaviour is removed from the specification. We use three experimental setups that address these two phenomena and show that all the precision measures listed in Table 2, except the eigenvalue-based measure, fail to demonstrate monotonicity for at least one of the setups. We also compare and discuss precision and recall measurements computed for several real-world and synthetic datasets using the techniques listed in Tables 2 and 3.

7.1.1 Monotonicity of Precision Measures. Next, we discuss the results of the three experimental setups that aim to study the monotonicity of the existing precision measures. In the first setup, given the log that contains traces with up to two events a before event b and a perfectly fitting specification, we gradually add behaviour to the specification. We use regular expressions[13] to describe the languages of the log and specifications:

L is the *log* with the language $\texttt{a\{0,2\}} \circ \texttt{b}$;
M_x are the *specifications* with language $\texttt{a\{0,x\}} \circ \texttt{b}$, $\texttt{x} \in [2\,..\,20]$;
$M_\star$ is the *specification* with language $\texttt{a}^* \circ \texttt{b}$.

Fig. 8 shows the values of various precision measures on the y-axis, plotted for the different specification languages reported on the x-axis: from 0–2 possible repetitions of a before b up to 0–20 repetitions. The last measurement on the end of the x-axis denotes the precision w.r.t. the

[12]The collection of the specifications used in the experiment (which also includes specifications that due to space considerations were not discussed in this article), as well as tools to generate sample executions and discovered models, and scripts to reproduce the experiment is available at: https://github.com/andreas-solti/monotone-precision.

[13]Notation a{<min>,<max>} is a short-hand for enumerating the minimal and maximal number of repetitions of symbol a. For example, a{0,2}∘b specifies the language {<b>,<a,b>,<a,a,b>}.

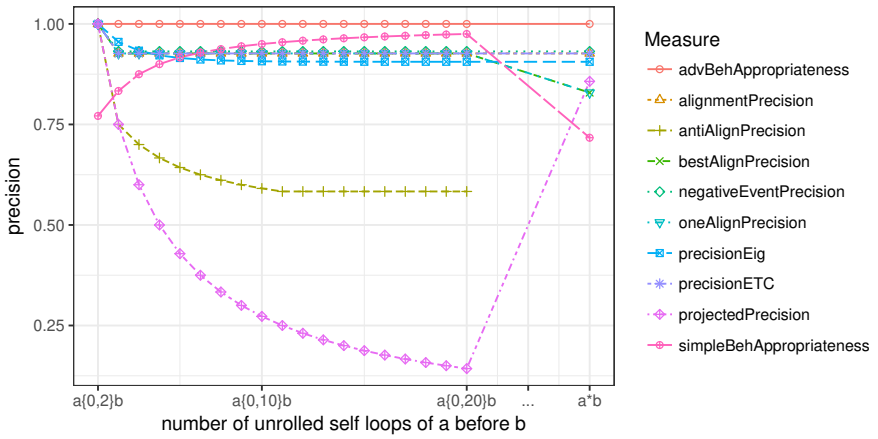


Fig. 8. Increasing number of optional a's before b. Starting with up to two a's before b, stepwise allow more a's up to the closure that allows an arbitrary number of a's before b.

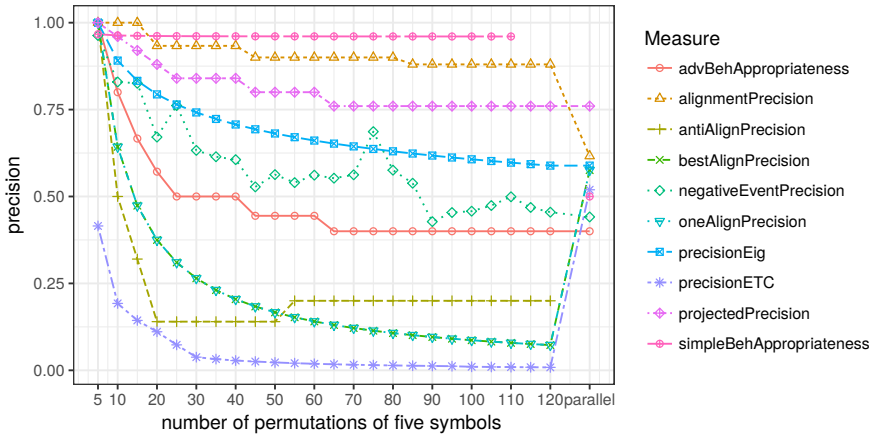


Fig. 9. The trends of precision measures for the log L w.r.t. specifications that allow permutations of the same five symbols, including the precision of all the explicit 5! = 120 permutations, and the precision of the language equivalent parallel specification with 120 implicitly allowed permutations.

specification that accepts an arbitrary number (i.e., 0–∞) of a's before b, that is $\mathtt{a}^* \circ \mathtt{b}$. The values were recorded only if they were computed under the threshold of ten minutes.

The simple behavioural appropriateness measure [87] shows a trend opposite to the other measures, as the precision values increase for more permissive specifications. Advanced behavioural appropriateness [87] fails to recognise the "growth" of the specifications' languages. The anti-alignments precision [102] demonstrates the correct trend, but has been unable to compute the precision for the $\mathtt{a}^* \circ \mathtt{b}$ specification within the threshold time. PCC precision [57] is strictly monotone in the region between up to 2 and up to 20 a's before b, but violates the monotonicity in the step from the $\mathtt{a\{0,20\}} \circ \mathtt{b}$ to $\mathtt{a}^* \circ \mathtt{b}$ specification. The other measures show a similar trend starting at 1.00 for the perfectly fitting specification and decreasing but stabilizing quickly. These measures, however, do not distinguish between the specifications $\mathtt{a\{0,y\}} \circ \mathtt{b}$, where $y \in [3 \,..\, 20]$. Our eigenvalue-based precision measure shows a steady stabilizing decline, i.e., the more possible repetitions of a before b are allowed by the specification the smaller the precision value is.

Besides iteration, parallelism, captured via interleavings of actions, is another dimension that we investigate. We vary the number of permutations over a fixed alphabet of size 5. Each word is constructed by drawing five out of five available symbols without replacement, where the order matters. Hence, there are 5! = 120 distinct permutations of symbols, i.e., 120 distinct words. A process specification that allows parallel execution of five activities also permits exactly 120 different executions. We expect a specification that enumerates all 120 permutations to be equally precise as another specification that uses a parallel building block that says that the same five activities can be done in any order. Thus, the second experimental setup uses the following log and specifications.

- $L_{5||}$ is the *log* with language {abcde, abced, abdec, abdce, abecd};
- $M_{x||}$ is the collection of *specifications* such that each specification describes all the five traces in $L_{5||}$, and further permutations of symbols a, b, c, d, and e, such that specification $M_{x||}$, $5 \leq x \leq 120$, describes x distinct permutations, and for all $5 \leq x < y \leq 120$ it holds that $M_{y||}$ describes all the permutations described by $M_{x||}$;
- $M_{||}$ is the *specification* with the language of all 120 permutations of symbols a, b, c, d, and e implemented as a parallel block of five simultaneously enabled activities.

Most existing precision measures listed in Table 2 show decreasing trends for log $L_{5||}$ and the collection of specifications $M_{x||}$, $5 \leq x \leq 120$, as it can be noticed in Fig. 9. However, the

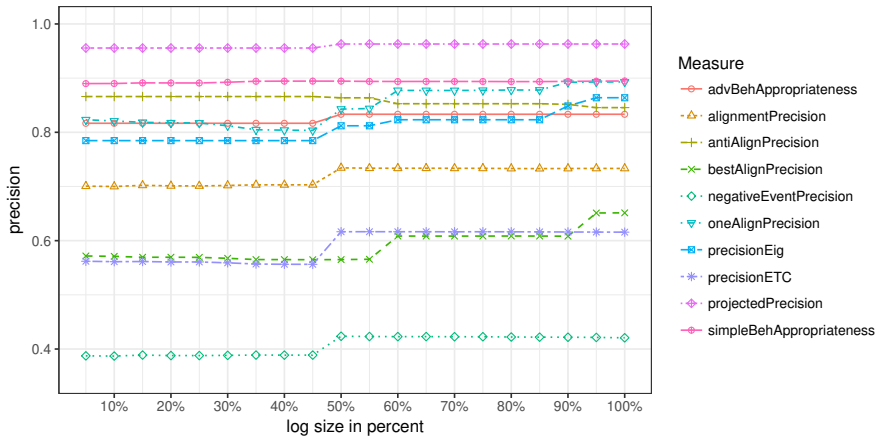


Fig. 10. Expected increase in precision with more traces in the log of the main process (A) in [101].

specification that explicitly encodes all the 120 permutations often has a different precision value than the specification with five activities in parallel, although these two specifications describe the same language. Note that only three measures reported the same precision values for both these specifications, namely advanced behavioural appropriateness [87], PCC precision [57], and our eigenvalue-based measure. The monotonicity in the second experimental setup is violated by the ETC precision [72], one optimal-alignment precision [2], best optimal-alignments precision [2], anti-alignments precision [102], and AGNEs specificity [40]. We were unable to compute anti-alignments precision [102] for the fully parallel specification within ten minutes. Also, we were unable, using available tools and the ten minutes time threshold, to compute simple behavioural appropriateness [87] for the specifications that explicitly capture more than 110 permutations. Note that the value of simple behavioural appropriateness drops significantly for the parallel specification.

In the third experimental setup, we use the real-world log of the BPI Challenge 2012 [101]. We discover a specification M that can replay all the traces in the log using Inductive Miner [57] with the infrequent and noise threshold parameters set to 0. Then, we select five percent of random traces from the log to obtain sub-log $L_{5\%}$ and compute the precision of the specification w.r.t. the sub-log. We repeat this process for other sub-logs, each obtained by adding an additional five percent of random traces from the original log, such that $L_{5\%} \subset L_{10\%} \subset \ldots \subset L_{100\%}$; $L_{x\%}$ is a sub-log that contains x percent of traces from the BPI Challenge 2012 log. Because the specification fits the original log perfectly, it holds that M describes all the traces in all the studied sub-logs. The measured precision values are reported in Fig. 10. Note that when we increase the number of traces in the sub-log, new behaviour is not necessarily added. At each step, we can potentially end up adding only traces that the previous sub-log already contains. To make the results accessible, we also created the plot shown in Fig. 11. It depicts the differences between two consecutive precision values, e.g., if the precision value increased by 0.1 when adding five percent of traces, we add a mark at 0.1. For a monotonically increasing measure, one should observe only non-negative differences; negative differences are emphasized with red triangles in the figure.

Only three precision measures demonstrate monotonicity for the third experimental setup. These measures are advanced behavioural appropriateness [87], PCC precision [57], and our eigenvalue-based precision. Some negative values are due to the non-deterministic nature of corresponding precision measures, as discussed in [95]. Also, there is a systematic error in the anti-alignments precision values [102] that show an unexpected downward trend, despite the fact that the

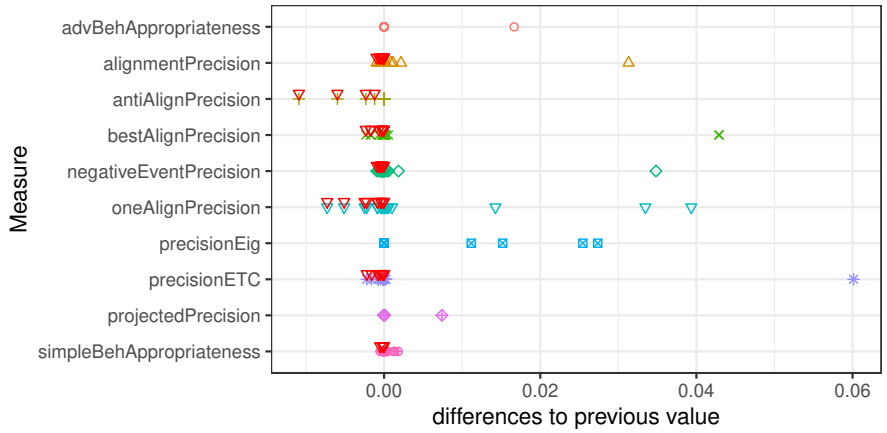


Fig. 11. Each dot represents the relative increase or decrease in Fig. 10 at each subsequent measurement step as the size of the log increases; red triangles encode relative decrease.

specification is fixed and the number of traces in the sub-logs increases in this experimental setup. Note that the eigenvalue-based precision measure, as guaranteed by its properties, successfully recognises all the four changes in the behaviours encoded in the sub-logs.

To conclude, for each of the precision measures listed in Table 2 except the eigenvalue-based precision, we were able to construct at least one example that violates the monotonicity property captured in Lemmata 4.2 and 4.3.

7.1.2 Monotonicity of Recall Measures. The recall of a specification w.r.t. a log is defined as the fraction of a measurement of the shared behaviour by a measurement of the behaviour in the log. In this case, both measurements capture finite behaviour, which makes the problem of computing the fraction less challenging than in the case of measuring precision.

Fig. 12 plots recall values for the measures listed in Table 3. The values were obtained in the following experimental setup. Given a sequential specification of ten activities and a fitting log with no noise, we start increasing the amount of noisy traces in the log. Here, noise is defined as removing, adding, or swapping events in the log, and the percentage shown on the x-axis reflects the relative number of traces affected by noise. Continued parsing measure [116] and proper completion [87] simply count the fraction of traces that are entirely fitting. For example, continued parsing measure is based on a binary decision for each trace. Hence, small deviations between

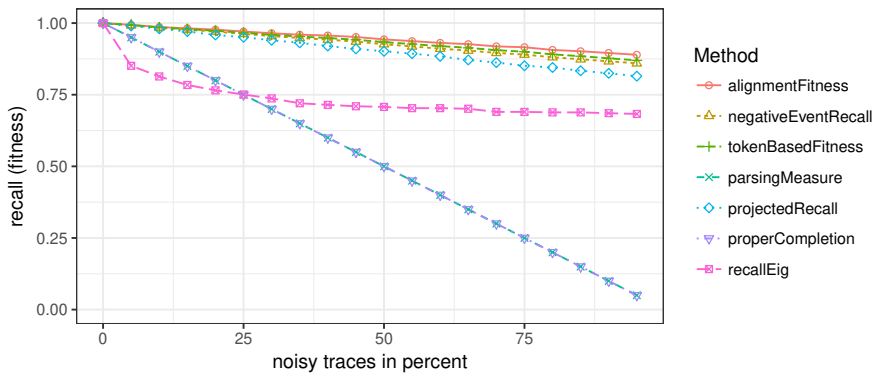


Fig. 12. Recall measures for a sequential specification and increasing amount of noise in a fitting log.

a trace and execution have the same impact on the measured value as significant differences. In contrast, alignment-based fitness [97], AGNEs recall [40], and token-based fitness [87] are sensitive to small discrepancies in traces in the log and executions of the specification, i.e., they penalize minor deviations between traces and executions only slightly.

The eigenvalue-based measures depend on the "sizes" of the languages of the compared log and specification. The above experimental setup shows that the behaviour in the log changes significantly with the insertion of first noisy traces, when the noise level is low. This leads to a rapid drop in recall as, indeed, the specification fails to capture the fresh behaviour, but only captures its deterministic sequential part. The increase in the number of noisy traces does not change 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* w.r.t. 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 for the situation described in the first row lower than that for the situation described in the second row, as the variance in the log is lower even though it has the same number of deviating traces.

Table 4. Precision and recall for specification that describes one execution $\langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$ and two logs $L_{abc(d|e)?}$ and $L_{abc(d)?}$. Log $L_{abc(d|e)?}$ consists of three traces: $\langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$, $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle$, and $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{e} \rangle$. Log $L_{abc(d)?}$ consists of five traces: three occurrences of trace $\langle \mathsf{a}, \mathsf{b}, \mathsf{c} \rangle$ and two occurrences of trace $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \rangle$.

Specification	Log	Precision	Recall
S_{abc}	$L_{abc(d\|e)?}$	1.0	0.789
S_{abc}	$L_{abc(d)?}$	1.0	0.856

While the amount of noisy traces increases linearly in this experiment, we are interested in the behaviour that is in both specification and log versus the behaviour in the log only. Our eigenvalue-based recall captures this non-linearity in the behaviour of the log. Thus, we conclude that if one is interested in the measure of how much behaviour of a log is captured in a specification, our measure is more suitable. However, if one is interested only in the fitting part of the log and does not need to distinguish between different deviations, the traditional fitness/recall measures are preferable. Latter linearly capture a decreasing number of fitting traces w.r.t. a given specification.

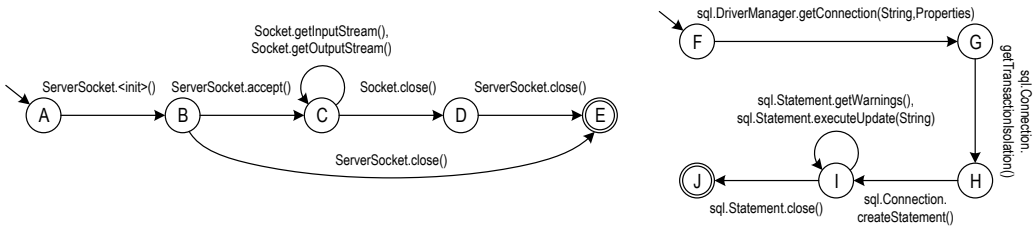
7.2 Comparing Specifications: Coverage

The language coverage measure for two systems $\mathcal{S}_x$ and $\mathcal{S}_y$ was introduced in Section 2 using the language cardinality measure. To overcome the problem of measuring infinite languages of systems, according to the framework presented in Section 4.1, we instantiate the language coverage quotient with the short-circuit measure induced by the eigenvalue measure, i.e., $eig^\bullet$, as follows:

$$coverage_{eig^\bullet}\left(\mathcal{S}_x, \mathcal{S}_y\right) := \frac{eig^\bullet\left(L(\mathcal{S}_x) \cap L(\mathcal{S}_y)\right)}{eig^\bullet\left(L(\mathcal{S}_x)\right)}.$$

We demonstrate the use of $coverage_{eig^\bullet}$ for measuring how well the behaviour of one software system *covers* the behaviour of some other software system using the following experiment. Given a specification of a software system $\mathcal{S}$, we simulate a collection of its executions, i.e., a log, $\mathcal{L}$. Next, we discover a specification $\mathcal{D}$ from $\mathcal{L}$. Finally, we compute $coverage_{eig^\bullet}(\mathcal{S}, \mathcal{D})$ and $coverage_{eig^\bullet}(\mathcal{D}, \mathcal{S})$.

Fig. 14(a) plots language coverage values between the specification in Fig. 13(a) and the discovered specifications from various collections of its simulated traces. Similarly, Fig. 14(b) plots language coverage values between the specification in Fig. 13(b) and the corresponding discovered

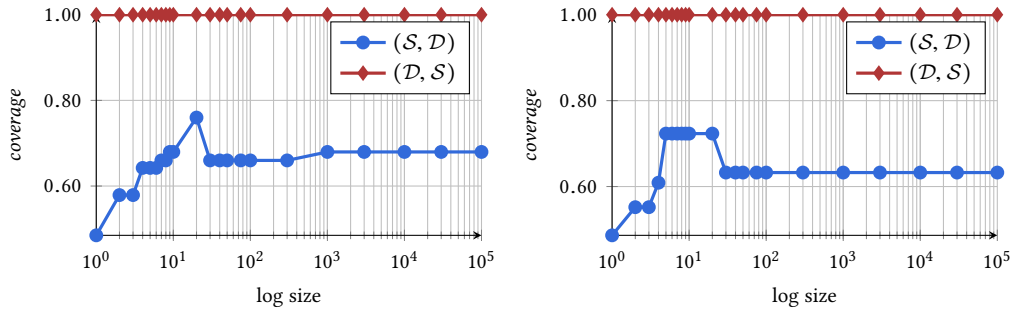


(a) Socket API [5] (b) Hibernate [39]

Fig. 13. Two specifications of software systems captured as DFAs.

specifications. Both specifications in Fig. 13 were used in [39] in the context of evaluating an algorithm for automatically discovering specifications from executions of software systems. In particular, Fig. 13(a) captures the Socket API reproduced from [5], while Fig. 13(b) describes a part of Hibernate functionality (see [39] for details). The specifications were discovered using Inductive Miner [57] with the infrequent and noise threshold parameters set to 0.2.

In both plots in Fig. 14, each blue circle encodes $coverage_{eig\bullet}(\mathcal{S},\mathcal{D})$ for the corresponding specification $\mathcal{S}$ from Fig. 13 and some specification $\mathcal{D}$ automatically discovered from a log whose size (measured as the number of, not necessarily distinct, traces) is reflected on the x-axis; note that the x-axis uses a logarithmic scale. Similarly, red diamonds encode the corresponding $coverage_{eig\bullet}(\mathcal{D},\mathcal{S})$ values. Note that the simulated logs are in the subset relation, i.e., each log $\mathcal{L}'$ is strictly contained in every log $\mathcal{L}''$ that has more traces than $\mathcal{L}'$.



(a) Socket API (b) Hibernate

Fig. 14. Language coverage between designed and discovered specifications of software systems.

The fact that all the measured $coverage_{eig\bullet}(\mathcal{D},\mathcal{S})$ values are equal to 1.0 (see the red diamonds in Fig. 14), tells that the behaviours of all the discovered specifications are sub-behaviours of the corresponding specifications from Fig. 13. In general, one should expect that specifications discovered from more traces of a system should cover the behaviour of the system better, i.e., the values denoted by blue circles in Fig. 14 should increase with the increase in the size of the log. Any other trend suggests that the discovery algorithm ignores, or even loses, knowledge about the behaviour of the system with more observed traces available.

7.3 Scalability

Next, we study the scalability of the eigenvalue-based precision and recall measures using real-world and synthetic datasets.

7.3.1 Scalability Evaluation on Real-World Data. Practical language measures and quotients must be able to handle large languages. Hence, we measured the wall-clock time of the eigenvalue-based precision and recall computation for fifteen real-world logs and corresponding discovered specifications. The logs encode executions of real-world IT systems executing genuine business processes with real customers. The logs are publicly available[14] and of different complexities. The log with the least variation in traces is the BPI Challenge (BPIC) 2013 (open cases) log. It can be encoded in a finite acyclic automaton with only 116 states. The BPIC 2017 log, on the other hand, translates to an automaton with 105 387 states.

Table 5. Measurements on Ryzen 5 2600X with 64GB of RAM.

	Automaton size (# states)			Largest eigenvalue			Recall	Precision	Wallclock-time (minutes)			
Log name	L	M	L ∩ M	L	M	L ∩ M			L	M	L ∩ M	total
BPIC'12	27 943	4	27 943	1.40	22.00	1.40	1.000	0.063	5.69	0.00	5.69	11.38
BPIC'13-closed	280	9	57	2.09	2.70	1.85	0.837	0.685	0.00	0.00	0.00	0.00
BPIC'13-incidents	4 426	4	67	2.20	3.19	1.78	0.731	0.558	0.23	0.00	0.02	0.25
BPIC'13-open	116	8	5	2.71	2.08	1.75	0.559	0.840	0.00	0.00	0.00	0.00
BPIC'15-1	33 090	25	12 815	1.62	390.24	1.45	0.771	0.004	8.65	0.00	1.08	9.72
BPIC'15-2	32 060	26	25 863	1.64	389.67	1.63	0.983	0.004	4.45	0.00	3.66	8.11
BPIC'15-3	33 353	16	33 200	1.68	374.26	1.68	1.000	0.004	0.86	0.00	0.87	1.73
BPIC'15-4	27 566	28	27 466	1.71	322.20	1.71	1.000	0.005	1.30	0.01	1.56	2.87
BPIC'15-5	36 221	14	26 609	1.37	369.17	1.36	0.996	0.004	3.00	0.00	0.85	3.85
BPIC'17	105 387	6	105 387	1.39	22.45	1.39	1.000	0.062	80.71	0.00	80.71	161.43
WABO-1	23 416	17	10 585	1.63	367.98	1.51	0.844	0.004	0.85	0.00	1.00	1.85
WABO-2	23 930	14	23 312	1.49	369.26	1.39	0.820	0.004	0.87	0.00	0.65	1.52
WABO-3	23 519	35	23 519	1.61	361.57	1.57	0.954	0.004	0.60	0.00	0.42	1.03
WABO-4	19 984	46	19 984	1.54	297.59	1.54	1.000	0.005	2.97	0.00	2.97	5.93
WABO-5	25 060	4	24 981	1.37	346.03	1.37	1.000	0.004	0.43	0.00	1.43	1.86

For each log, we discovered a specification using Inductive Miner [57] configured with the default noise threshold of 0.2. For each log and corresponding discovered specification, 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 specification M, and their intersection automaton $L \cap M$ are shown in Table 5. As an indicator of the complexity, the number of states of the respective automata are listed in the table. Note that the specification automata are considerably smaller than the corresponding log automata. Presumably, this is because the employed discovery algorithm constructs specifications that do not contain duplicate actions. The adjacency matrix of an automaton has size that is quadratic in the number of states in 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.

The variance in measured wall-clock times is notable. The longest time to compute the eigenvalue-based precision and recall was taken for the BPIC 2017 log, whereas for most of the experimented logs, both precision and recall values were computed under ten minutes and often much faster. The technique used for computing largest eigenvalues is called "implicitly restarted Arnoldi iterations" [59]. Note that this numerical method for computing a largest eigenvalue of a general matrix always converges, but provides no guarantees of convergence in a fixed number of iterations. Thus, in our implementation, for practical reasons, we use the threshold of 300 000 iterations for the maximum number of iterations. For all the experimented logs, this threshold was sufficient to ensure the convergence of the computations.

[14]Logs are available at: https://data.4tu.nl/repository/collection:event_logs_real

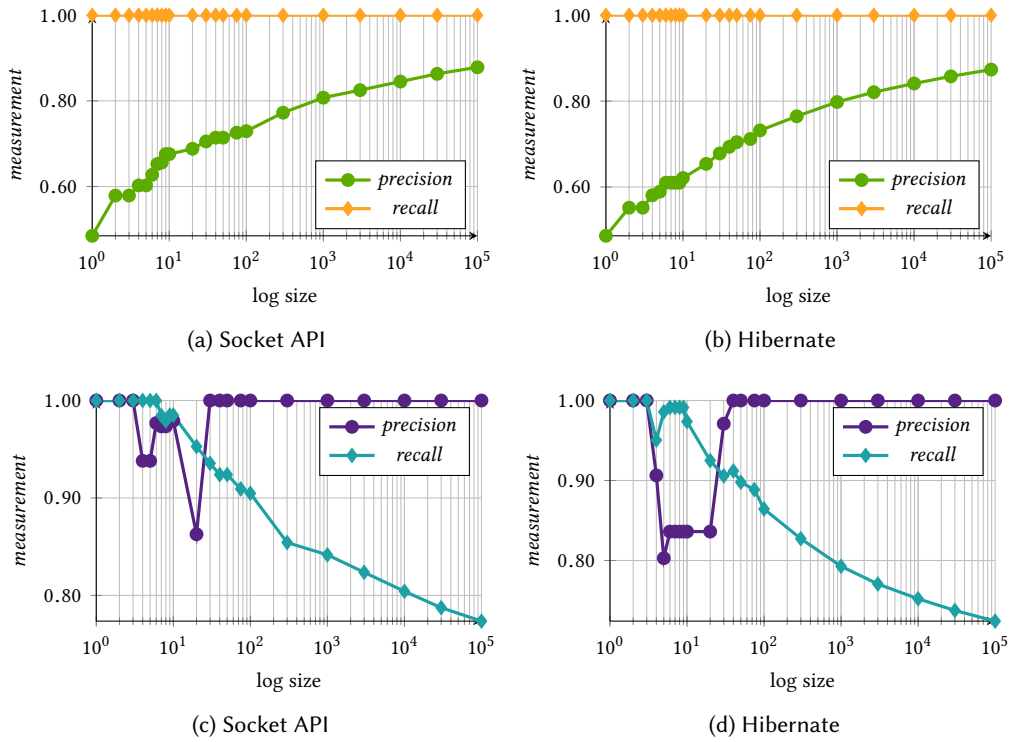
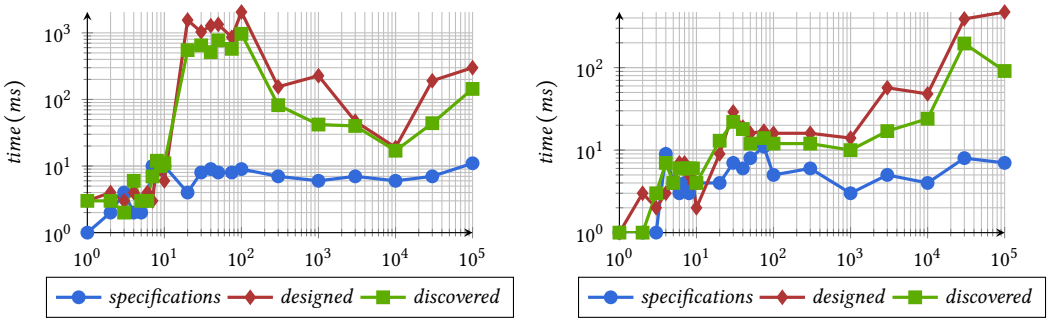


Fig. 15. Eigenvalue-based precision and recall of: designed specifications w.r.t. their executions, (a) and (b), and discovered specifications w.r.t. the executions they were discovered from, (c) and (d).

7.3.2 Scalability Evaluation on Synthetic Data. In the evaluation on the real-world data, the characteristics of logs in terms of the variety and number of traces differed a lot. To perform a consistent analysis, next, we report on the values and computation times of the eigenvalue-based quotients for the simulated logs and specifications, both designed and discovered, discussed in Section 7.2.

Fig. 15 plots the measured eigenvalue-based precision and recall values. Fig. 15(a) and Fig. 15(b) show the values for, respectively, the Socket API and Hibernate specification w.r.t. the simulated logs. Because all the logs are composed of executions of the specifications, all the recall values are equal to 1.0. As can be seen from the plots, with the increase of the number of traces in logs, the precision values increase, which is consistent with the fact that the eigenvalue-based precision is monotone. Fig. 15(c) and Fig. 15(d) show precision and recall values for the corresponding discovered specifications w.r.t. the simulated logs. As can be observed from the plots, the recall values tend to decrease with the increase of the log size. The fact that precision values tend to be 1.0 suggests that this particular configuration of the discovery technique constructs specifications that do not generalize beyond the behaviour seen in the logs.

Fig. 16 plots times of computing the eigenvalue-based quotients reported in Fig. 14 and Fig. 15; note the use of logarithmic scales for both axis. Each plotted value reports the overall time of computing two quotients for the corresponding event log shown on the x-axis. The blue circles denote the times of computing the coverage quotients for *specifications* from Fig. 14. The red diamonds show the times of computing precision and recall for the *designed* specifications reported in Fig. 15(a) and Fig. 15(b). Finally, the green squares report the times of computing precision and recall for the *discovered* specifications reported in Fig. 15(c) and Fig. 15(d).



(a) Socket API (b) Hibernate

Fig. 16. Wallclock-time of computing the eigenvalue-based quotients on Ryzen 5 2600X with 64GB of RAM.

The experimental results, both on the real-world and synthetic datasets, tell us that the size of the input is not enough to determine the runtime of the method. Instead, the rate of convergence of a largest eigenvalue computation depends on other properties of the adjacency matrices of the underlying automata, e.g., on the difference between the largest and the second-largest eigenvalue. We conclude the scalability experiments with the insight that our current implementation of the method shows variability in performance depending on the convergence of the underlying eigenvalue computation.

8 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 [41], reactive systems [23], and agent programming [28], to mention but a few. Section 8.1 outlines noticeable notions of behavioural equivalence and behavioural comparison, including inheritance and similarity. Then, Section 8.2 describes the evolution of the precision and recall measures for behavioural comparison in the field of process mining, along with highlights on commonalities and dissimilarities to our approach. Finally, Section 8.3 reports on previous research on behavioural comparison in software engineering, again emphasising the similarities and differences with our technique.

8.1 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 [104]. We remark that the systems 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 systems of computation in this context is the one that guarantees that any step performed in one system can be mimicked by the other one, and vice versa [32]. This idea is the basis for the notion of *bisimulation* [70]. On rooted labelled transition systems (a super-class of the systems we analyse), bisimulation imposes that from the initial state onward, possible actions must coincide between the systems and inductively lead to states that are bisimilar as well. *Weak bisimulation* [70] relaxes bisimulation in that it considers only observable actions, i.e., it is permitted that systems guarantee bisimulation on non-τ transitions only, as τ transitions can be added as prefix- or suffix-moves to that extent. *Branching bisimulation* enforces weak bisimulation by requiring that the same set of choices is offered before and after each unobservable action [103].

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 [41]. *Completed* trace equivalence adds the condition that, if systems have sink states from which no further action is possible, they must be reachable in systems 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 system is extended by a second one.

Kunze and Weske [55] 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 satisfied only if the language of a system is included in the language of another system at the same level of abstraction. In the light of that definition, our research thus focuses on behavioural inheritance [7], and specifically trace inheritance, between dynamic systems. 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 [33] to the set of systems' 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 [53]. Noticeable examples include the n-gram similarity [68], comparing systems by the shared allowed n-long sub-sequences in systems' respective traces. Despite the efficiency of the solution, the issue is that even if n-grams coincide, 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 [54]. The idea is to compare "footprints" of systems, 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 [114]. Despite being semantically rich, Polyvyanyy et al. [77] show 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 dynamic systems into precision and recall.

8.2 Precision and Recall in Process Mining

Process mining aims at extracting knowledge about processes from the digital data stored by organisations' IT systems [37]. Process mining is adopted to discover new facts, including process specifications themselves that were not documented before, compare the expected process behaviour with reported reality and detect deviations between the former and the latter [98]. It shows thus the inherent aim of finding and assessing the match between the behaviours of a dynamic system, in terms of to-be process specifications 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. [17] identify (replay) fitness, precision, generalisation, and simplicity as the four main quality dimensions for assessing the quality of process mining results [49].

A first precision measure called "behavioural appropriateness" is introduced in the seminal work of Rozinat and van der Aalst [87] as the degree of how much behaviour is permitted by the specification although not recorded in the 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. In this way, it is higher when sometimes-forward and sometimes-backward relation pairs shared between specification and log approximate the total amount of the specification. Conversely, it is lower if the specification allows for more variability. The assumption of total fitness of the log entails that the log cannot show more variability than the specification. 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.

The ETConformance approach avoids the complete exploration of the specification behaviour by traversal of the specification to solely reflect the traces recorded in the log [72]. 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 locality of the approach allows for efficient computation, with the downside that only behaviour close to the log is considered. Similarly, our approach assesses precision by quantifying the behavioural differences among states of a finite-state rooted labelled transitions system. However, it abstracts from the recorded runs of the involved specifications. Remarkably, Munoz-Gama and Carmona [72] also introduce advanced diagnostic measures to assess the severity of imprecisions and their stability factor with respect to small perturbations in the log.

An approach combining the concept of prefix automaton with the one of *alignments* [3] is proposed by van der Aalst et al. [97] 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 specification and the ones that were actually executed as per the prefix automaton, given the replay history. Adriansyah et al. [2] propose different precision measures based on the nature of the alignments to be considered. The underlying structure remains a prefix automaton as in [72], here augmented by associating weights to states. As in the approaches of [97] and [2], the precision measure proposed in this paper 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 specifications' languages, it abstracts from the replay and counting of events.

More recently, Leemans et al. [57] introduced precision and recall measures to compare the behaviour of specifications or logs, requiring a finite state automaton as the underlying structure for a state-to-state comparison as in [2, 72]. To cope with the high computational effort required by the intersection operations, a projection of both specifications 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 [57], 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 by van Dongen et al. [102] using the concept of anti-alignment first proposed in [19]. An anti-alignment is a finite trace of a given length which is accepted by the process specification, yet not in the log and sufficiently distant from any trace therein (where the trace distance can be computed by using edit distance [61], e.g.). To assess 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 specifications, thus abstracting from the number of occurrences of a trace. However, our approach does not require the iterative scan and comparison of specifications excluding parts of the behaviour, thus saving on computation time.

The Artificially Generated Negative Events technique (AGNEs) discovers process specifications out of logs enriched with artificially injected negative events [40]. 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. [116] introduce the so-called Parsing Measure (PM), which is based on the fraction of correctly parsed traces over all traces in the input log. As a derivative, the Continued Parsing Measure (CPM) provides a more fine-granular analysis, at the price of being bound to the specification of the underlying Heuristic Miner. Our notion of recall for a specification is also based on the measuring of the part of language not covering another behaviour. Noticeably, PM and CPM weigh the amount of incorrectly parsed traces, thus quantitatively assessing to which extent the divergences occur in the event log. Owing to our level of abstraction, we do not account for this assessment. 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 [87] counts the number of tokens consumed and produced during the replay of traces over the Petri net specification, 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* in [49]. Another token-based fitness measure, used in genetic process mining, accounts for trace frequency [29]. In contrast to [29, 87], we aim at defining measures that are not tailored to specific behaviour specification language, thus we do not rely on Petri net semantics to define recall.

The concept of alignment-based fitness introduced by van der Aalst et al. [97] 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 specification. 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 specification 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 specification language, as in the rationale of our investigation. Our technique does not allow the user to indicate costs. Providing this feature in our approach is an intriguing problem that could be addressed in future work. On the other hand, our approach does not resort on the computationally expensive finding of optimal runs on the input specifications.

We remark that especially the approaches described in [2, 57, 87, 97] not only propose precision and recall measures and algorithms for their computation, but provide also techniques to illustrate where and in how far deviations occur between the log and the specification. The integration of those powerful diagnostic tools with our approach delineates interesting plans for future research.

To conclude, [95] recently defined five requirements (there named *axioms*) that a precision measure should guarantee, in a strive for the general definition of fundamental properties that should be satisfied by process mining quality measures. The authors show that neither of aforementioned simple behavioural appropriateness [87], advanced behavioural appropriateness [87], ETC precision [72], AGNEs specificity [40], or PCC precision [57] comply with their requirements for precision. By design, our approach fulfils all those requirements instead, as shown in Section 7.

8.3 Behavioural Comparison in Software Engineering

In software engineering, a noticeable body of literature on automaton-based specification mining [5, 65] have proposed highly relevant contributions towards the behavioural comparison of state machines.

Javert [39] generates complex system specifications stemming from mined patterns. To that end, the technique applies sound composition rules of branching and sequencing on discovered simple patterns, thus achieving good scalability. Similarly to our solution, Javert resorts on automata theory for the composition steps and the representation of models. Our approach thus complements Javert in that it can measure the precision and recall of those returned models.

Shoham et al. [90] adopt an automata-based approach to automatically mine the specification of client interactions with APIs for object-oriented libraries. Their approach resorts on the notion of quotient automata to abstract on the representation of behaviour through an equivalence relation over states. Notice that the notion of behavioural quotient we propose applies to languages for obtaining a measurable comparison of systems behaviour regardless of their model's structure. Our language quotient framework is thus separate and integrable with the technique of Shoham et al. [90], which could be employed to take advantage of their effective removal of spurious patterns.

Lo and Khoo [62] propose a framework called QUARK (QUality Assurance framewoRK) for empirically assessing the automata generated by different miners. Their assumption is that two models have to be compared: one reference and one reverse-engineered from API interactions. This context is similar to ours in that we also compare a reference process specification with another behavioural abstraction, in our case stemmed from a set of execution traces of a process. In their approach, they compute accuracy in terms of trace similarity. They first collect two samples of randomly generated traces, one per model. The precision is the proportion of samples generated by the reverse-engineered model that are accepted by the reference automaton. Dually, the recall is the proportion of traces that are generated by the reference automaton, and are accepted by the reverse-engineered one. Our approach moves in the opposite direction: we abstract from traces and compare systems, rather than comparing traces generated by the systems. Remarkably, Lo and Khoo [62] also propose measures that deal with probabilistic finite automata, based upon the Hidden Markov Models comparison. Their study suggests the extension of our approach toward the analysis of probabilistic models as an opportunity for future research.

The use of simulated traces for system comparison, first reported in [56] and applied in QUARK [62], has been later criticised by Walkinshaw et al. [110]. A problem is that it is virtually impossible to cover the whole behaviour of a system by random walks. This problem is of high severity especially because some faulty executions might remain unexplored by a random sample, which is of high relevance in software testing [111, 118]. To address this bias, Walkinshaw et al. [111] propose an adaptation of the original Vasilevski/Chow W-Method [13, 22]. Their technique is aimed at generating test sets that cover all distinguishable runs of the model. Furthermore, they refine the notions of precision and recall to account for not only the traces that are mutually accepted by the compared models, but also to inspect the capability of the two to reject traces that are not compliant with the target behaviour. In our context, to-be-rejected traces are not considered as we assume the log to stem from registered correct system runs. However, we see in this aspect an endeavour for future work: an extension of our language-quotient based approach that accounts for the semantic discrimination of runs that ended up in positive outcomes from those that do not, similarly to what was done by Ponce de León et al. [81] and Chesani et al. [21].

Walkinshaw and Bogdanov extend their seminal work [110] in two directions. First, they expand the comparison measures with classical data mining ones such as specificity and balanced classification rate. Second, they introduce the LTSDiff algorithm, which compares models under a structural

perspective, rather than a behavioural one. In this paper, we do not consider the structural similarity, thus being model-agnostic and not imposing requirements on the determinism or minimality of input systems. However, our technique could be improved by integrating the cognitive-like, iterative approach of the LTSDiff algorithm, based on an intermediate results expansion starting from landmarks [92] (i.e., matching subsets of the inputs).

Quante and Koschke [84] first consider a measure for model comparison taking into account the language of involved automata without the analysis of generated traces. They devise to that extent an approach similar to that of edit distance. A minimised union automaton is first created between the input ones. Thereupon, a concurrent synchronous run is executed on each of the models and the union automaton. It determines the number of edits, that is, the transitions to be removed from the union (never traversed) or added to the input model (unfolded self-loops). The final measure is computed by averaging the distances in terms of edits of the models from the union automaton. Our approach revolves around language comparison based on the analysis of automata as well. However, it discriminates between precision and recall, thus giving a more precise picture of the accuracy of the mined model with respect to the reference of the log.

Pradel et al. [82] use a variant of the k-tails algorithm [12] to compare mined and reference models. To that extent, they first generate the union of the finite automata given as input models. Then, they adapt the k-tails algorithm to approximate the matching of those states from which common (sub)sequences of length k can be generated. Such states are then merged. Precision is computed based on the number of shared transitions between the mined model and the intersection of the reference model with the automaton subject to k-tails merging. Recall is computed analogously but switching mined and reference model. The usage of k-tails to merge states allows for the processing of models mined from noisy or incomplete traces. On the other hand, the fact that matches are not exact and subject to a proper choice of k may lead to an inaccuracy of results, as emphasised by Walkinshaw and Bogdanov [109]. As in [82], our approach considers a language abstraction of systems for comparison purposes, without generating trace sets. In contrast to it, we do not resort to structural approximations over the input specifications. On the one hand, it favours accuracy. On the other hand, an adaptation of our approach to account for noise, as in [82], is an interesting direction for future work.

Interesting research avenues for future work stem from the extension of language measures to cater for more expressive models than automata-based behaviours labelled by the sole activity name. Berg et al. [10] present an algorithm that extends the transition labels of inferred automata with propositional guards on function parameter values, based on queries over the observed runs of protocol implementations. Later, Lorenzoli et al. [66] with GK-tail and Walkinshaw et al. [112] with MINT (Model Inference Technique) propose techniques to infer Extended Finite State Machines (EFMSs), namely automata with guards on data stored in the program memory of the program, from a set of program traces. Emam and Miller [38] improve on the existing EFMS inference algorithms with a stochastic-based approach to include in the discovered models of behaviour the probabilities that determine the likelihood of transitions. The techniques proposed by Narayan et al. [74] discover behavioural rules based on Timed Regular Expressions (TREs), which are equivalent to timed automata [6], to cater for constraints related to real-time. Krismayer et al. [52] illustrate a technique to mine constraints out of event logs that store the information of software systems operating in the cyber-physical domain. The analyzed constraints express rules on the sequence of actions, exert limitations on time spans, and predicate on attribute values of the events. These works inspire interesting future endeavours for our research to measure precision and recall of models of behaviour including data and time aspects.

9 DISCUSSIONS

This section summarizes the main results of this work and the lessons we learned on the way to obtaining them (Section 9.1), discusses threats that could have influenced the validity of the reported conclusions (Section 9.2), and suggests how the presented results may contribute to software engineering practices (Section 9.3).

9.1 Results and Learned Lessons

For over a decade, through the design of various measures and analytics, the process mining community shaped the intuition underpinning the comparison of a specification of a dynamic system with its executions. Intuitively, a specification should allow for the behaviour seen in the executions and forbid other behaviour [98]. It is only recently that this intuition started to take a concrete form in terms of formal properties that such comparison measures should satisfy [95, 99]. A repertoire of properties a given measure satisfies can then be seen as a proxy to its usefulness, i.e., if a practitioner is interested in certain properties she should pick and use a measure that satisfies them. The work reported in [95] was the first attempt to propose such properties. On several informal occasions, the properties from [95] were criticized for being somewhat naïve. Indeed, they can be satisfied by a measure that, for example, returns zero for any input log and the most permissive specification, i.e., the one that accepts any word, and otherwise returns some constant greater than zero and less than or equal to one. Obviously, such a measure is not particularly useful.

One issue with the properties from [95] is that two specifications, one of which exhibits strictly more behaviour than the other, are allowed to have the same precision value with a given log. In this work, we strengthened the properties from [95] to require a less permissive specification to be more precise with respect to the log (see Lemma 4.2 and Lemma 4.3). As this requirement introduces an additional restriction, every measure that satisfies our properties is guaranteed to satisfy the corresponding less restrictive properties from [95]. Finally, all the other properties from [95] are trivially, by definition, satisfied by every precision measure that follows Definition 5.2.

In [99], 21 properties for conformance measures are proposed. Among those properties, two address both recall and precision measures, five are specifically concerned with recall measures, while six address precision measures. The properties for precision measures aim to diversify and strengthen the properties from [95]. Recently, in [93], it was shown that all the precision and recall properties from [99] hold for the precision and recall measures presented in Section 5. For example, Propositions 5 and 8 in [99] follow immediately from Lemma 4.2 and the fact that a language measure is deterministic (see Section 4.1), while Propositions 3 and 9 in [99] follow immediately from Lemma 4.3 and the definition of a language measure.

As of today (December 2019), there is no precision measure, other than the quotient (Definition 5.2) instantiated with the short-circuit measure (Definition 4.6) induced by the eigenvalue measure (Section 4.3), that satisfies the strict properties captured in Lemma 4.2 and Lemma 4.3, and all the properties for precision presented in [93, 95]. Note that Lemma 4.2 and Lemma 4.3 address both finite and infinite languages.

9.2 Threats to Validity

An important concern about an experiment is how valid its results are [119]. Note that threats to validity relate to our empirical analysis–formal properties of our measures are not subject to these threats. According to [25], there are four types of threats to the validity of experimental results: internal, construct, conclusion, and external validity. Next, we discuss several identified aspects that threaten the construct and conclusion validity of the results of our experiments reported in Section 7. Aspects that threaten construct validity refer to the extent to which the experiment

setting reflects the phenomenon that is studied. Aspects that threaten conclusion validity relate to the ability to make correct conclusions about the observed outcomes in response to the treatments of the experiment [119].

With respect to *construct validity*, we first focus on the threats of *incomplete* selections of subjects and their *random heterogeneity*. Our selection of precision and recall measures for the experiment was initiated with the six precision measures studied in [95] and, then, extended to nine precision and six recall measures, cf. Table 2 and Table 3. The selection was primarily driven by the availability of open-source implementations of the measures in ProM and CoBeFra frameworks [105] in 2017. Hence, the selection of the measures for experimentation may not be complete. In the recent study mentioned above, namely in [93], eight recall and eleven precision measures were evaluated. One of these eight recall measures is the eigenvalue-based recall presented in this work. From the remaining seven recall measures, six are also evaluated in Section 7; the baseline recall measure presented in [93] is equivalent to proper completion measure. Hence, one recall measure was studied in [93] but not evaluated in Section 7, viz. causal footprint recall [98]. Note, however, that causal footprint recall was shown in [93] to fulfil only four out of seven recall-related properties from [99], whereas our recall measure, as shown in [93], fulfils all the seven properties. Out of eleven precision measures evaluated in [93], one is the eigenvalue-based precision presented in this work, while seven are also evaluated in Section 7. The baseline precision measure from [93], not evaluated in this work, is undefined for specifications that encode infinite collections of executions and, thus, can be seen as a theoretical baseline measure with a rather limited practical applicability. Furthermore, we did not study behavioural precision [113] and weighted negative event precision [106], which both aim to improve the measure from [40] evaluated in Section 7. However, in [93], all the three measures from [40, 106, 113] were demonstrated to violate four out of eight properties for precision measures, which suggests that the measures are qualitatively similar. Note that our precision measure, as shown in [93], fulfils all the eight properties.

We further need to consider a potentially *restricted generalizability across constructs*, since several issues with measures of our comparison set may not render it useless. Also, the choice of properties to consider may be subject to discussion. While these threats cannot be discarded in their entirety since their definition and selection follows conceptual arguments, we observe that our definition of properties is consistent with those defined by other research [93, 95]. Also, we acknowledge that violating a property does not mean that it will be violated frequently or with high severity in a specific set of application scenarios.

An important threat to *conclusion validity* related to *'fishing'* for a specific result, and indeed, a biased selection of the measures to compare against would be problematic. However, our experiments considered a large collection of precision and recall measures that are commonly used and which also rely on different formal foundations. This restricts the potential impact of this threat. Considering *reliability of measures* and potentially *low statistical power*, we acknowledge that we proved the violation of certain properties through counterexamples, relying on the third-party implementations of the evaluated precision and recall measures. While we observed that not all of the tested synthetic examples lead to the respective violations, our main claims relate to the formal guarantees that our measure provides and which other measures miss. This is a formal argument that is not affected by statistical considerations. Moreover, our datasets are limited to the BPIC logs and specifications synthesized with one discovery algorithm. In the model-to-model comparisons, we have been limited to two designed models and a small set of discovered specifications based on one discovery technique. While we see no evidence for one of these aspects affecting the conclusion validity, they constitute a certain threat. Lastly, to exclude *random irrelevancies* in the experimental setting, we ran our experiments for several times to record average execution times, verifying that the same outcome is observed.

9.3 Software Engineering Practice

Behavioural specifications like the ones used in this paper are extensively used in practice for problem solving (95%) and documentation (91%) [47], with visual use case models (39%) and business process models (23%) being among the most popular ones [108]. The behavioural comparison of representations of dynamic systems is at the core of many software engineering techniques. Our measures, therefore, have a potential to influence software engineering practice. In the remainder, we discuss the implications for several exemplary areas, such as software configuration management, model-based software engineering, and software testing.

Software configuration management (SCM) [60] comprises models and methods to track, organize, and control the evolution of the artefacts involved in the development of a software system. It is motivated by changes in the requirements to address, the people involved in development, the policies and rules to obey, or a project's schedule. The behavioural quotients defined in our work may support several of common SCM practices once configuration items (CIs) such as source code modules, test cases, and requirements specifications have been identified. For instance, SCM requires the definition of baselines, formally established versions of CIs that structure the progress of a development project. An example is the functional baseline that describes an item's functional, interoperability, and interface characteristics [51]. In a functional configuration audit, as part of SCM, behavioural quotients can enable an assessment of the degree to which the baseline has been reached. Moreover, to assess the impact of change requests on CIs that capture behavioural information, such as UML activity diagrams or source code fragments, behavioural quotients can be used to quantify the impact of the respective request.

Turning to specific software development methodologies, we consider approaches for model-based software engineering (MBSE) [15]. In essence, they aim at structuring the development process around abstract models and automated code generation through model transformations. However, current MBSE practice faces challenges related to the maintenance of code generators that transform models into executable code, concerning the design of domain-specific languages (DSLs), and related to the integration within agile development projects [50]. Behavioural quotients may support initiatives to overcome these challenges by assessing the behavioural difference of models to identify required changes in code generators, by comparing instances defined in DSLs to assess their commonalities for consolidation, and by providing notions of model consistency to enable the identification of the impact of frequent changes of models.

As a final example, we refer to notions that may guide the definition of test cases for a particular system [73]. Considering approaches for white-box testing at the level of functional units, coverage is an important quality criterion [11, 96]. Behavioural quotients may be employed to assess the coverage achieved by a test suite, where the abstraction employed in the definition of the languages over which the quotients are computed enables the realization of various coverage criteria, such as function-based or branching-based coverage. As already discussed in Section 8.3, quotients may also be considered as a basis to quantify test results, once they are lifted to a model that distinguishes accepted and rejected test runs. By employing coarse-grained abstractions that hide the internals of units, in turn, this approach is also useful in an assessment of the results obtained through black-box testing.

10 CONCLUSION

This article proposed behavioural quotients as a means to relate the behaviours of dynamic 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 other behaviours. Three example language measures are put forward in the article: one over finite, one over irreducible

regular languages, and one over regular languages. The language measure over regular languages is based on the notion of topological entropy. It is used to instantiate behavioral quotients into language coverage, precision, and recall measures for software engineering and process mining. The reported evaluation results demonstrated that the proposed quotients can be computed in a reasonable time and qualitatively outperform (based on the property of the monotonicity) all the existing measures for precision in process mining.

Future work on behavioural quotients will 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 [77, 114], declarative models [35, 100], and hybrid representations [30, 67] in light of their underlying expressibility as finite-state automata [31, 34, 77]. Furthermore, one can propose new language measures for instantiating behavioural quotients and study interpretations and computational complexities of these measures. Moreover, language quotients can be improved to account for multiplicity and similarity of words. The quotients proposed in this article abstract from multiplicities of words and consider words as being distinct even if they differ only in a single symbol. Also, the works on the automated inference of models of behaviour including data attributes, time, and probabilities in their transitions [38, 52, 66, 74, 112] inspire an interesting future avenue for our research, i.e., to measure precision and recall of such extended models. Another context in which one can investigate the applicability and adaptation of our approach is that of the models of behaviour expressing distributed systems invariants [42]. Finally, one can design new quality measures that relate arbitrary numbers of behaviours (not just behaviours of a specification and its execution log), e.g., to establish a basis for comparing results of various process querying methods [79], models of behaviour that summarise traces at varying levels of abstraction [43], and different behavioural representations [83].

The recent observation that all the state-of-the-art precision measures in process mining fail to satisfy some basic desired properties [95], initiated a discussion on what properties should the standard quality measures, like precision, recall, and generalization, possess [99]. The precision and recall defined as language quotients, such as the entropy-based measures, satisfy all the properties proposed in [95, 99] (see Section 5). This result is due to the fact that these measures are defined as ratios over language measures. Consequently, they satisfy the properties of non-negativity, have null sets, and are strictly monotone (see [94] for details on the standard properties of measures). Therefore, we propose to shift the focus of the discussion from the desired properties of the quality measures to the desired properties of measures over languages that are used to define them. For example, it is interesting to study if an additional requirement of *additivity* or *sub-additivity* over a language measure used to instantiate precision and recall quotients leads to their useful properties.

The monotonicity property allows comparing measured values over behaviours, but not reasoning over their absolute values. Indeed, the difference in the measured values over two languages in the containment relation has no particular meaning. Also, it is not established which concrete precision values denote precise or imprecise models with respect to a given event log. Future works will tackle these problems in dialogue with domain experts.

The devised behavioural quotients were tested using real-world and synthetic logs of IT systems that govern the execution of business processes and synthetic logs of software specifications. In Section 9.3, we discussed the use of behavioural quotients for software configuration management, model-based software engineering, and software testing. However, future work will need to adapt to those use cases the behavioural quotients we present in this work, as yet unforeseen obstacles may arise in their immediate adoption in software engineering practices.

ACKNOWLEDGMENTS

Artem Polyvyanyy was partly supported by the Australian Research Council Discovery Project DP180102839. 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. The work of Claudio Di Ciccio and Jan Mendling received funding from the EU H2020 programme under the MSCA-RISE agreement 645751 (RISE_BPM) and the Austrian Research Promotion Agency (FFG) grant 861213 (CitySPIN). Claudio Di Ciccio was partly supported by the MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza University of Rome. We would like to thank Anna Kalenkova for her review of our manuscript and comments that helped to improve it.

REFERENCES

[1] Andrew Abbott and Alexandra Hrycak. 1990. Measuring resemblance in sequence data: An optimal matching analysis of musicians' careers. *American journal of sociology* 96, 1 (1990), 144–185.

[2] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2015. Measuring precision of modeled behavior. *Inf. Syst. E-Business Management* 13, 1 (2015), 37–67.

[3] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M.P. van der Aalst. 2011. Conformance Checking Using Cost-Based Fitness Analysis. In *EDOC*. IEEE Computer Society, 55–64.

[4] Nasir Ali, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2013. Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links. *IEEE Trans. Software Eng.* 39, 5 (2013), 725–741. https://doi.org/10.1109/TSE.2012.71

[5] Glenn Ammons, Rastislav Bodík, and James R. Larus. 2002. Mining specifications. In *POPL*. ACM, 4–16. https://doi.org/10.1145/503272.503275

[6] Eugene Asarin, Paul Caspi, and Oded Maler. 2002. Timed regular expressions. *J. ACM* 49, 2 (2002), 172–206. https://doi.org/10.1145/506147.506151

[7] Antonius André Basten. 1998. *In terms of nets: System design with Petri nets and process algebra.* Ph.D. Dissertation.

[8] Amit Basu and Akhil Kumar. 2002. Research commentary: Workflow management issues in e-business. *Inf.Syst.Res.* 13, 1 (2002), 1–14.

[9] Nicholas Berente, Stefan Seidel, and Hani Safadi. 2019. Research Commentary - Data-Driven Computationally Intensive Theory Development. *Inf. Syst. Res.* 30, 1 (2019), 50–64. https://doi.org/10.1287/isre.2018.0774

[10] Therese Berg, Bengt Jonsson, and Harald Raffelt. 2006. Regular Inference for State Machines with Parameters. In *FASE*. Springer, 107–121. https://doi.org/10.1007/11693017_10

[11] Stefan Berner, Roland Weber, and Rudolf K. Keller. 2007. Enhancing Software Testing by Judicious Use of Code Coverage Information. In *ICSE*. IEEE Computer Society, 612–620. https://doi.org/10.1109/ICSE.2007.34

[12] Alan W. Biermann and Jerome A. Feldman. 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Trans. Computers* 21, 6 (1972), 592–597. https://doi.org/10.1109/TC.1972.5009015

[13] Kirill Bogdanov, Mike Holcombe, Florentin Ipate, L. Seed, and Salim K. Vanak. 2006. Testing methods for X-machines: a review. *Formal Asp. Comput.* 18, 1 (2006), 3–30. https://doi.org/10.1007/s00165-005-0085-6

[14] Egon Börger. 2005. Abstract state machines and high-level system design and analysis. *Theor. Comput. Sci.* 336, 2–3 (2005), 205–207. https://doi.org/10.1016/j.tcs.2004.11.006

[15] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. *Model-Driven Software Engineering in Practice.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00441ED1V01Y201208SWE001

[16] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Joerg Becker. 2016. Comprehensible Predictive Models for Business Processes. *MIS Q.* 40, 4 (2016), 1009–1034.

[17] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2014. Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity. *Int. J. Cooperative Inf. Syst.* 23, 1 (2014), 1–40.

[18] Tullio Ceccherini-Silberstein, Antonio Machi, and Fabio Scarabotti. 2003. On the entropy of regular languages. *TCS* 307, 1 (2003), 93–102.

[19] Thomas Chatain and Josep Carmona. 2016. Anti-alignments in Conformance Checking - The Dark Side of Process Models. In *ICATPN (LNCS)*, Vol. 9698. Springer, 240–258.

[20] Kwang-Ting Cheng and A. S. Krishnakumar. 1993. Automatic Functional Test Generation Using the Extended Finite State Machine Model. In *DAC*. ACM Press, 86–91. https://doi.org/10.1145/157485.164585

[21] Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. 2009. Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. *T. Petri Nets and Other Models of Concurrency* 2 (2009), 278–295. https://doi.org/10.1007/978-3-642-00899-3_16

[22] Tsun S. Chow. 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Software Eng.* 4, 3 (1978), 178–187. https://doi.org/10.1109/TSE.1978.231496

[23] Edmund M. Clarke, Orna Grumberg, and Doron Peled. 2001. *Model Checking*. MIT Press. I–XIV, 1–314 pages.

[24] Jonathan E. Cook and Alexander L. Wolf. 1998. Discovering Models of Software Processes from Event-Based Data. *ACM Trans. Softw. Eng. Methodol.* 7, 3 (1998), 215–249. https://doi.org/10.1145/287000.287001

[25] T.D. Cook and D.T. Campbell. 1979. *Quasi-experimentation: design & analysis issues for field settings*. Rand McNally College.

[26] Benjamin Cornwell. 2015. *Social sequence analysis: Methods and applications*. Vol. 37. Cambridge University Press.

[27] Marco D'Ambros, Harald C. Gall, Michele Lanza, and Martin Pinzger. 2008. Analysing Software Repositories to Understand Software Evolution. In *Software Evolution*, Tom Mens and Serge Demeyer (Eds.). Springer, 37–67. https://doi.org/10.1007/978-3-540-76440-3_3

[28] Giuseppe De Giacomo, Alfonso Emilio Gerevini, Fabio Patrizi, Alessandro Saetti, and Sebastian Sardiña. 2016. Agent planning programs. *Artif. Intell.* 231 (2016), 64–106.

[29] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. 2007. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.* 14, 2 (2007), 245–304.

[30] Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. 2015. Fusion Miner: Process discovery for mixed-paradigm models. *Decision Support Systems* 77 (2015), 123–136.

[31] Johannes De Smedt, Claudio Di Ciccio, Jan Vanthienen, and Jan Mendling. 2017. Model Checking of Mixed-Paradigm Process Models in a Discovery Context - Finding the Fit Between Declarative and Procedural. In *BPM workshops*. Springer, 74–86. https://doi.org/10.1007/978-3-319-58457-7_6

[32] Stéphane Demri, Valentin Goranko, and Martin Lange. 2016. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge University Press.

[33] Elena Deza and Michel Deza. 2006. *Dictionary of Distances*. North-Holland. I–XV, 1–391 pages.

[34] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. 2017. Resolving inconsistencies and redundancies in declarative process models. *Information Systems* 64 (2017), 425–446. https://doi.org/10.1016/j.is.2016.09.005

[35] Claudio Di Ciccio and Massimo Mecella. 2015. On the Discovery of Declarative Control Flows for Artful Processes. *ACM TMIS* 5, 4 (2015), 24:1–24:37.

[36] Kumar Dookhitram, Ravindra Boojhawon, and Muddun Bhuruth. 2009. A new method for accelerating Arnoldi algorithms for large scale Eigenproblems. *Mathematics and Computers in Simulation* 80, 2 (2009), 387–401. https://doi.org/10.1016/j.matcom.2009.07.009

[37] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. 2018. *Fundamentals of Business Process Management*. Springer.

[38] Seyedeh Sepideh Emam and James Miller. 2018. Inferring Extended Probabilistic Finite-State Automaton Models from Software Executions. *ACM Trans. Softw. Eng. Methodol.* 27, 1 (2018), 4:1–4:39. https://doi.org/10.1145/3196883

[39] Mark Gabel and Zhendong Su. 2008. Javert: fully automatic mining of general temporal properties from dynamic traces. In *SIGSOFT FSE*. ACM, 339–349. https://doi.org/10.1145/1453101.1453150

[40] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. 2009. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research* 10 (2009), 1305–1340.

[41] Roberto Gorrieri and Cristian Versari. 2015. *Introduction to Concurrency Theory - Transition Systems and CCS*. Springer.

[42] Stewart Grant, Hendrik Cech, and Ivan Beschastnikh. 2018. Inferring and asserting distributed system invariants. In *ICSE*. ACM, 1149–1159. https://doi.org/10.1145/3180155.3180199

[43] Abdelwahab Hamou-Lhadj and Timothy Lethbridge. 2006. Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System. In *ICPC*. IEEE Computer Society, 181–190. https://doi.org/10.1109/ICPC.2006.45

[44] David Harel. 1987. Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Program.* 8, 3 (1987), 231–274.

[45] John E. Hopcroft. 1971. *An n Log n Algorithm for Minimizing States in a Finite Automaton*. Technical Report. Stanford.

[46] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to automata theory, languages, and computation, 3rd Edition*. Addison-Wesley.

[47] John Hutchinson, Jon Whittle, and Mark Rouncefield. 2014. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming* 89 (2014), 144–161.

[48] Akira Imakura and Tetsuya Sakurai. 2018. Block SS-CAA: A complex moment-based parallel nonlinear eigensolver using the block communication-avoiding Arnoldi procedure. *Parallel Comput.* 74 (2018), 34–48. https://doi.org/10.

1016/j.parco.2017.11.007
[49] Gert Janssenswillen, Niels Donders, Toon Jouck, and Benoît Depaire. 2017. A comparative study of existing quality measures for process discovery. *IS* 71 (2017), 1–15.
[50] Oliver Kautz, Alexander Roth, and Bernhard Rumpe. 2018. Achievements, Failures, and the Future of Model-Based Software Engineering. In *The Essence of Software Engineering*, Volker Gruhn and Rüdiger Striemer (Eds.). Springer, 221–236. https://doi.org/10.1007/978-3-319-73897-0_13
[51] Jessica Keyes. 2004. *Software configuration management.* Auerbach Publications.
[52] Thomas Krismayer, Rick Rabiser, and Paul Grünbacher. 2019. A Constraint Mining Approach to Support Monitoring Cyber-Physical Systems. In *CAiSE (LNCS)*. Springer, 659–674. https://doi.org/10.1007/978-3-030-21290-2_41
[53] Matthias Kunze. 2013. *Searching business process models by example.* Ph.D. Dissertation. University of Potsdam.
[54] Matthias Kunze, Matthias Weidlich, and Mathias Weske. 2015. Querying process models by behavior inclusion. *SoSyM* 14, 3 (2015), 1105–1125.
[55] Matthias Kunze and Mathias Weske. 2016. *Behavioural Models - From Modelling Finite Automata to Analysing Business Processes.* Springer.
[56] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. 1998. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In *ICGI (LNCS)*. Springer, 1–12. https://doi.org/10.1007/BFb0054059
[57] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. 2018. Scalable process discovery and conformance checking. *SoSyM* 17, 2 (2018), 599–631.
[58] Richard B Lehoucq. 2001. Implicitly restarted Arnoldi methods and subspace iteration. *SIAM J. Matrix Anal. Appl.* 23, 2 (2001), 551–562.
[59] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. 1998. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* SIAM.
[60] Alexis Leon. 2015. *Software configuration management handbook.* Artech House.
[61] Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dok.* 10, 8 (1966), 707–710.
[62] David Lo and Siau-Cheng Khoo. 2006. QUARK: Empirical Assessment of Automaton-based Specification Miners. In *WCRE*. IEEE Computer Society, 51–60. https://doi.org/10.1109/WCRE.2006.47
[63] David Lo and Siau-Cheng Khoo. 2006. SMArTIC: towards building an accurate, robust and scalable specification miner. In *FSE*. ACM, 265–275.
[64] David Lo, Siau-Cheng Khoo, and Chao Liu. 2007. Efficient mining of iterative patterns for software specification discovery. In *SIGKDD*. ACM, 460–469. https://doi.org/10.1145/1281192.1281243
[65] David Lo, Siau-Cheng Khoo, Jiawei Han, and Chao Liu. 2011. *Mining software specifications: methodologies and applications.* CRC Press.
[66] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. 2008. Automatic generation of software behavioral models. In *ICSE*. ACM, 501–510. https://doi.org/10.1145/1368088.1368157
[67] Fabrizio Maria Maggi, Tijs Slaats, and Hajo A. Reijers. 2014. The Automated Discovery of Hybrid Processes. In *BPM*. Springer, 392–399.
[68] Bendick Mahleko, Andreas Wombacher, and Peter Fankhauser. 2005. Process-annotated service discovery facilitated by an n-gram-based index. In *EEE*. IEEE, 2–8.
[69] Leonardo Mariani and Mauro Pezzè. 2005. Behavior Capture and Test: Automated Analysis of Component Integration. In *ICECCS*. IEEE Computer Society, 292–301. https://doi.org/10.1109/ICECCS.2005.25
[70] Robin Milner. 1982. *A Calculus of Communicating Systems.* Springer.
[71] Frank R. Moore. 1971. On the Bounds for State-Set Size in the Proofs of Equivalence Between Deterministic, Nondeterministic, and Two-Way Finite Automata. *IEEE Trans. Comput.* 20, 10 (1971), 1211–1214.
[72] Jorge Munoz-Gama and Josep Carmona. 2011. Enhancing precision in process conformance: Stability, confidence and severity. In *CIDM*. IEEE, 184–191.
[73] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The art of software testing.* John Wiley & Sons.
[74] Apurva Narayan, Greta Cutulenco, Yogi Joshi, and Sebastian Fischmeister. 2018. Mining Timed Regular Specifications from System Traces. *ACM Trans. Embedded Comput. Syst.* 17, 2 (2018), 46:1–46:21. https://doi.org/10.1145/3147660
[75] William Parry. 1964. Intrinsic Markov Chains. *Trans. Amer. Math. Soc.* 112, 1 (1964), 55–66.
[76] Brian T. Pentland. 2003. Conceptualizing and measuring variety in the execution of organizational work processes. *Man. Sci.* 49, 7 (2003), 857–870.
[77] Artem Polyvyanyy, Abel Armas-Cervantes, Marlon Dumas, and Luciano García-Bañuelos. 2016. On the expressive power of behavioral profiles. *Formal Asp. Comput.* 28, 4 (2016), 597–613.
[78] Artem Polyvyanyy and Anna Kalenkova. 2019. Monotone Conformance Checking for Partially Matching Designed and Observed Processes. In *International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019.*

IEEE, 81–88. https://doi.org/10.1109/ICPM.2019.00022
[79] Artem Polyvyanyy, Chun Ouyang, Alistair Barros, and Wil M. P. van der Aalst. 2017. Process querying: Enabling business intelligence through query-based process analytics. *DSS* 100 (2017), 41–56.
[80] Artem Polyvyanyy and Matthias Weidlich. 2013. Towards a Compendium of Process Technologies—The jBPT Library for Process Model Analysis. In *CAiSE Forum (CEUR Workshop Proceedings)*, Vol. 998. CEUR-WS, 1–8. http://ceur-ws.org/Vol-998/Paper14.pdf
[81] Hernán Ponce de León, Lucio Nardelli, Josep Carmona, and Seppe K. L. M. vanden Broucke. 2018. Incorporating negative information to process discovery of complex systems. *Inf. Sci.* 422 (2018), 480–496. https://doi.org/10.1016/j.ins.2017.09.027
[82] Michael Pradel, Philipp Bichsel, and Thomas R. Gross. 2010. A framework for the evaluation of specification miners based on finite state machines. In *ICSM*. IEEE Computer Society, 1–10. https://doi.org/10.1109/ICSM.2010.5609576
[83] Johannes Prescher, Claudio Di Ciccio, and Jan Mendling. 2014. From Declarative Processes to Imperative Models. In *SIMPDA*. CEUR-WS, 162–173.
[84] Jochen Quante and Rainer Koschke. 2007. Dynamic Protocol Recovery. In *WCRE*. IEEE Computer Society, 219–228. https://doi.org/10.1109/WCRE.2007.24
[85] Michael O. Rabin and Dana S. Scott. 1959. Finite Automata and Their Decision Problems. *IBM J. Res. Dev.* 3, 2 (1959), 114–125.
[86] Steven P. Reiss and Manos Renieris. 2001. Encoding Program Executions. In *ICSE*, Hausi A. Müller, Mary Jean Harrold, and Wilhelm Schäfer (Eds.). IEEE Computer Society, 221–230.
[87] Anne Rozinat and Wil M.P. van der Aalst. 2008. Conformance checking of processes based on monitoring real behavior. *IS* 33, 1 (2008), 64–95.
[88] Anirudh Santhiar, Omesh Pandita, and Aditya Kanade. 2014. Mining Unit Tests for Discovery and Migration of Math APIs. *ACM Trans. Softw. Eng. Methodol.* 24, 1 (2014), 4:1–4:33. https://doi.org/10.1145/2629506
[89] Eugene Seneta. 2006. *Non-Negative Matrices and Markov Chains.* Springer.
[90] Sharon Shoham, Eran Yahav, Stephen J. Fink, and Marco Pistoia. 2008. Static Specification Mining Using Automata-Based Abstractions. *IEEE Trans. Software Eng.* 34, 5 (2008), 651–666. https://doi.org/10.1109/TSE.2008.63
[91] Michael Sipser. 2012. *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.
[92] Molly E. Sorrows and Stephen C. Hirtle. 1999. The Nature of Landmarks for Real and Electronic Spaces. In *COSIT (LNCS)*. Springer, 37–50. https://doi.org/10.1007/3-540-48384-5_3
[93] Anja F. Syring, Niek Tax, and Wil M. P. van der Aalst. 2019. Evaluating Conformance Measures in Process Mining Using Conformance Propositions. *Trans. Petri Nets and Other Models of Concurrency* 14 (2019), 192–221. https://doi.org/10.1007/978-3-662-60651-3_8
[94] T. Tao. 2013. *An Introduction to Measure Theory.* American Mathematical Society.
[95] Niek Tax, Xixi Lu, Natalia Sidorova, Dirk Fahland, and Wil M.P. van der Aalst. 2018. The imprecisions of precision measures in process mining. *IPL* 135 (2018), 1–8.
[96] Javier Tuya, Claudio de la Riva, María José Suárez Cabal, and Raquel Blanco. 2016. Coverage-Aware Test Database Reduction. *IEEE Trans. Software Eng.* 42, 10 (2016), 941–959. https://doi.org/10.1109/TSE.2016.2519032
[97] Wil M.P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. 2012. Replaying history on process models for conformance checking and performance analysis. *WIDM* 2, 2 (2012), 182–192.
[98] Wil M. P. van der Aalst. 2016. *Process Mining—Data Science in Action* (2nd ed.). Springer.
[99] Wil M. P. van der Aalst. 2018. Relating Process Models and Event Logs – 21 Conformance Propositions. In *ATAED (CEUR Workshop Proceedings)*, Vol. 2115. CEUR-WS.org, 56–74.
[100] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. 2009. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D* 23, 2 (2009), 99–113.
[101] Boudewijn F. van Dongen. 2012. Dutch Financial Institute's Event Log. Dataset of the BPI Challenge 2012.
[102] Boudewijn F. van Dongen, Josep Carmona, and Thomas Chatain. 2016. A Unified Approach for Measuring Precision and Generalization Based on Anti-alignments. In *BPM*. Springer, 39–56.
[103] Rob J. van Glabbeek. 1993. The Linear Time - Branching Time Spectrum II. In *CONCUR*. Springer, 66–81.
[104] Rob J. van Glabbeek and Ursula Goltz. 2001. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.* 37, 4/5 (2001), 229–327. https://doi.org/10.1007/s002360000041
[105] Seppe vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. 2013. A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. In *CIDM*. IEEE, 254–261.
[106] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. 2014. Determining Process Model Precision and Generalization with Weighted Artificial Negative Events. *IEEE Trans. Knowl. Data Eng.* 26, 8 (2014), 1877–1889. https://doi.org/10.1109/TKDE.2013.130

[107] Birgit Vogel-Heuser, Alexander Fay, Ina Schaefer, and Matthias Tichy. 2015. Evolution of software in automated production systems: Challenges and research directions. *J. Syst. Softw.* 110 (2015), 54–84.

[108] Stefan Wagner, Daniel Méndez Fernández, Michael Felderer, Antonio Vetrò, Marcos Kalinowski, Roel Wieringa, Dietmar Pfahl, Tayana Conte, Marie-Therese Christiansson, Desmond Greer, et al. 2019. Status quo in requirements engineering: A theory and a global family of surveys. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 2 (2019), 9.

[109] Neil Walkinshaw and Kirill Bogdanov. 2013. Automated Comparison of State-Based Software Models in Terms of Their Language and Structure. *ACM Trans. Softw. Eng. Methodol.* 22, 2 (2013), 13:1–13:37. https://doi.org/10.1145/2430545.2430549

[110] Neil Walkinshaw, Kirill Bogdanov, and Ken Johnson. 2008. Evaluation and Comparison of Inferred Regular Grammars. In *ICGI (LNCS)*, Vol. 5278. Springer, 252–265. https://doi.org/10.1007/978-3-540-88009-7_20

[111] Neil Walkinshaw, John Derrick, and Qiang Guo. 2009. Iterative Refinement of Reverse-Engineered Models by Model-Based Testing. In *FM (LNCS)*. Springer, 305–320. https://doi.org/10.1007/978-3-642-05089-3_20

[112] Neil Walkinshaw, Ramsay Taylor, and John Derrick. 2016. Inferring extended finite state machine models from software executions. *Empirical Software Engineering* 21, 3 (2016), 811–853. https://doi.org/10.1007/s10664-015-9367-7

[113] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. 2011. A robust F-measure for evaluating discovered process models. IEEE, 148–155. https://doi.org/10.1109/CIDM.2011.5949428

[114] Matthias Weidlich, Jan Mendling, and Mathias Weske. 2011. Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Trans. Software Eng.* 37, 3 (2011), 410–429.

[115] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. 2011. Process compliance analysis based on behavioural profiles. *Information Systems* 36, 7 (2011), 1009–1025.

[116] A.J.M.M. Weijters, Wil M.P. van der Aalst, and Ana Karla Alves de Medeiros. 2006. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP* 166 (2006), 1–34.

[117] Mathias Weske. 2012. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition.* Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28616-2

[118] Elaine J. Weyuker. 1983. Assessing Test Data Adequacy through Program Inference. *ACM Trans. Program. Lang. Syst.* 5, 4 (1983), 641–655. https://doi.org/10.1145/69575.357231

[119] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering.* Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

First version March 2019; Revised January 2020; Accepted March 2020