IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 March 2024, accepted 11 April 2024, date of publication 18 April 2024, date of current version 3 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3391234

RESEARCH ARTICLE

Comparing Conformance Checking for Decision Mining: An Axiomatic Approach

ADAM BANHAM[1], ARTHUR H. M. TER HOFSTEDE[1], SANDER J. J. LEEMANS[2,3,4], FELIX MANNHARDT[5], ROBERT ANDREWS[1], AND MOE T. WYNN[1], (Member, IEEE)

[1]School of Information Systems, Queensland University of Technology, Brisbane, QLD 4000, Australia
[2]Department of Mathematics and Computer Science, RWTH Aachen University, 52062 Aachen, Germany
[3]Teaching and Research Area of Business Process Management Foundations and Engineering, RWTH Aachen University, 52074 Aachen, Germany
[4]Fraunhofer, Institut für Angewandte Informationstechnik, 52074 Aachen, Germany
[5]Department of Mathematics and Computer Science, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands

Corresponding author: Adam Banham (adam.banham@hdr.qut.edu.au)

The work of Adam Banham was supported in part by Australian Government Research Training Program Scholarship; and in part by Queensland University of Technology, Centre for Data Science Scholarship.

ABSTRACT Process mining uses historical executions of business processes (as recorded in an event log) to uncover and describe the process' behaviour as a process model. The goal of conformance checking is to ensure that the model is of high quality and is a good representation of the event log, thus assuring the model is a solid foundation for subsequent analysis of the process. To date, few conformance checking approaches have considered model quality beyond what is determined by the execution order of process activities. In data-aware process models, which are generated by decision mining techniques, process activities are annotated with conditions to represent the decision-making of a process. Such models are more expressive than those that represent only process activities. With the current notions of conformance checking, it is unclear what properties determine the quality of these data-aware process models. To address this gap, we introduce desirable properties, as axioms, for conformance checking of data-aware models. Our contribution is threefold: i) we present a generalisation that abstracts from the representation of data-aware models, ii) we present nine axioms of desirable properties for data-aware conformance checking, and iii) we define two measures for model recall and precision. Using our axioms as a yardstick, we compare our proposed recall and precision measures with existing measures. Our experimental results show that existing measures exhibit limited adherence to our axioms; while, our two proposed measures exhibit high adherence to our axioms.

INDEX TERMS Process mining, data perspective, conformance checking, decision mining, data-aware conformance checking.

I. INTRODUCTION

Process mining is a field of data science that uses observed, historical data from an organisation's business process(es) to understand the actual behaviour and performance of the process(es) [1]. Such historical data is recorded in event logs, consisting of process instances (traces), where each trace describes a series of process activities that unfolded over time [9]. Process discovery techniques [23] exploit these traces to recreate the structuring of activities (control-flow) in the form of a process model. Then, by using conformance checking [11] the quality of a given process model against an event log can be quantified by considering the intersection of observed and described behaviour to measure concepts such as precision, recall, generalisation and simplicity [10], [24]. However, consensus has yet to be reached on the intuitive properties for conformance checking [2], [6], [40], [41].

To motivate which intuitive properties for conformance measures are desirable, we now provide a high-level summary of existing discussions [2], [6], [10], [40], [41]. Much of the existing work derives from [10], which showed how the behaviour in an event log, process model, and the underlying system can be disjoint or overlapping. In [2] and [41], the authors observed that existing conformance

The associate editor coordinating the review of this manuscript and approving it for publication was Wojciech Sałabun.

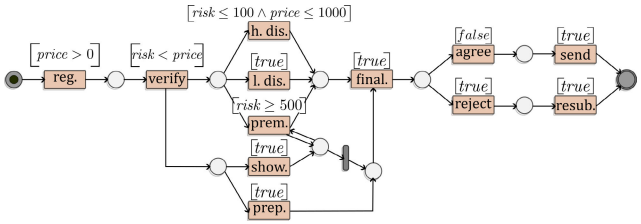


FIGURE 1. An example Petri net with data, with guards above transitions.

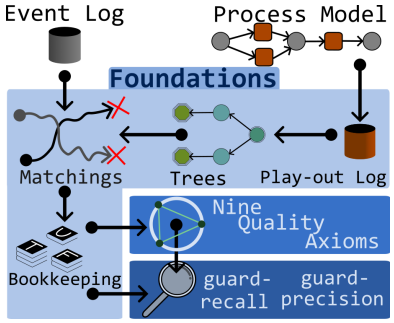


FIGURE 2. Our contributions.

measures were not formulated against a set of desirable properties. Hence, such measures were not easy to interpret or to compare against each other. This issue was confirmed through evaluations in [6] and [40]. Thus, using [10], the authors of [2], [40], and [41] propose desirable properties, or propositions/axioms, for conformance checking limited to the control-flow perspective (sequences of activities).

Conformance checking beyond control-flow is also important. In [31], the author considers multi-perspective process mining, which includes activities, resources, data attributes, and expressions (*guards*) for activities. Although [31] introduced conformance checking that considers guards, it did not consider desirable properties. Therefore, we extend existing discussions [2], [31], [40], [41], by considering desirable properties for data-aware process models [26], [31], [38].

Existing data-aware conformance checking [12], [19], [33], [35] predominantly utilises an alignment-based approach [4] to firstly compute a mapping between a log and a model, and then compute a distance between them [12], [33], [35]. The distance between an alignment and trace in the log, to some extent, is subjective as it is reliant on a domain-relevant cost-model, which specifies what steps are realistic and what are not (and are therefore penalised more). Furthermore, the cost-model defines which alignments are considered ''optimal''.

A property of alignments that must be taken into account when comparing models and an event log, is that existing measurements [12], [33], [35] have their state-space uniquely tied to the process model (and underlying cost-model). Thus, when comparing a single model with many logs, it is easy to identify the most compliant log. However, it is challenging to identify the highest quality model from a collection of models when considering a single log, as any model-log distance measurement is not guaranteed to be comparable to another due to the possibility that each model and underlying cost-model inducing a larger/smaller state-space. Comparing decision mining techniques to identify which technique excels in model-recall or model-precision [2], [3], [10], [11], requires just such a comparison between many models with a single log. Hence, in this paper, we address the challenge of identifying the highest quality model when considering a single log, and the desirable properties that should be considered.

To motivate data-aware conformance checking, we now consider a representation of a data-aware process model (see Fig. 1). This model describes the purchase of a financial product and depicts the concepts of choices, data and guards.

Central to our discussion in this paper is the notion of 'choices' in models. A choice represents a state in the model where a set of (more than one) process activities could be executed next, and from which, only one activity can happen. For instance, in Fig. 1, after the completion of 'final', both 'agree' and 'reject' are possible next activities, but the occurrence of one means the other cannot occur. Transitions in this model have been annotated with *guards*. Some guards, e.g., the guard for 'h.dis', are defined over process data attributes, while others, i.e. guards for 'show' and 'agree', are indifferent to data.

The semantics of guards may differ. For instance, the first two guards (for 'reg.' and 'verify') are domain policies that should be enforced in practice, and are challenging to discover (see [27]). Furthermore, one could consider guards that prescribe how data should change after a process activity [16], [28]. However, the following three guards (for 'h.dis.', 'l.dis.', 'prem.') are guards that determine, from data, what process activity should occur next. The latter is a specific concern for decision mining techniques [26], [31], [38] and we emphasise this viewpoint in this paper.

The quantification for the quality of guards is often restricted to their respective model representations [16], [26], [31], [33], [35]. To be able to compare any data-aware model with an event log, we need to forego specific representations of these models. While we visually represent models using Petri nets with data (DPN) notation, our approach does not rely on DPNs. Note that we require the control-flow of a model to always have the option to complete (see [1, clause 3 of Def.3.7]), but we do not restrict our approach to models that are relaxed data sound (see [31, Def. 5.1]) as existing decision mining techniques do not guarantee this property in their outcomes [26], [31], [38].

Fig. 2 outlines our contributions. Our foundation section (Section V) presents a generalised modelling formalism. This formalism is our means for quantification through: (i) inducing a set of execution paths from an executable process model [2] or a *play-out log*, (ii) constructing a *finite transition tree* from these execution paths based on a given event log, (iii) mapping traces in the log to paths through the tree through a *matching*, (iv) collecting

diagnostic information about the guards within the tree through *bookkeeping functions* for conformance checking. After which, we formulate desirable properties for data-aware conformance checking using our bookkeeping through nine quality axioms (Section VI), extending previous discussions in [2], [6], and [40]. To addresses these axioms, we proposing two new measures, guard-recall and guard-precision (Section VII). Finally, we evaluate our proposed measures and existing techniques [19], [33], [35] for adherence to our axioms (Section VIII).

The remainder of the paper is organised as follows: Section II provides a deeper look into the problem space for decision mining and how it affects conformance checking. Section III presents related work. Section IV introduces background and notation. Section IX summarises our contributions.

II. PROBLEM MOTIVATION

At a very high level, decision mining finds 'human-readable' guards for activities in a process model [29]. These guards guide the execution of a process model at runtime, and may lead to a deterministic execution. As observed in [25], different techniques exist for achieving this goal. Such techniques may either consider an existing control-flow and annotate the control-flow with guards, or adopt a decision-aware approach to process discovery [25], [29]. In this work, we focus on the control-flow annotation approach and its properties.

A typical decision mining technique using the annotation approach will consist of the following steps for a given model and event log: a) identify decision points in the model, b) find a control-flow mapping between the log and the model, c) construct classification problems for each decision point, d) learn a classifier for each problem, and then e) extract guards over a set of attributes from each classifier, which are then annotated to activities within the model.

A natural consequence of applying this approach is that all possible outcomes (i.e. data-aware models) will have the same control-flow, with differences between the outcomes arising due to the restrictions on model behaviour imposed by the various, different guard annotations. As such, we will discuss desirable qualities for conformance checking when considering models derived from the same control-flow and ensuring measurements over these models are comparable.

An important consideration for conformance checking is that measures should not be bound to representation, e.g. Petri nets or extensions thereof, such as Fig. 1. However, this does pose a problem with step a) identifying decision points or choices between process activities, as it requires some understanding of the representation of the model. To address this issue, we use an abstraction of process models, i.e. transition trees [20]. This model abstraction meets our need for clearly identifying decision points, however some cases might result in an infinite tree when considering loops and concurrency, but given our focus on what is observed these are largely not a concern.

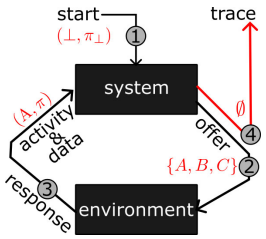


FIGURE 3. The conceptualisation of a system reacting to data and process activities [20], where the system signals that a trace is complete when it presents an empty offer $\varnothing$.

A further justification for the use of transition trees comes from the generalisation of process execution described in [20] and illustrated in Fig. 3. Here, process execution is conceived as a system which reacts to process activities and data (1 or 3), by presenting offers (i.e. a set of process activities) to an environment (2). Next, the environment responds to the offer, with a process activity and additional data back to the system (3). Then, the system adapts to the response, and this cycle of offers and responses is repeated until a process execution is completed (4). The iterative relationship between system and environment not only builds a trace, but is naturally represented as a transition tree. We use transition trees in our theory for process models in the rest of this paper and show that executable models can be represented by a tree.

We use the offers generated by the system to position conformance checking regarding precision and recall. For example, where a decision mining outcome restricts all offers presented at runtime by the system to only a single process activity, the process is entirely deterministic, and hence, precision is maximised. If any offer(s) consist of more than one process activity, the process is non-deterministic, implying precision will be less than maximal. Likewise, if any offer(s) are empty at runtime, the process will terminate early, which will negatively impact on recall. As such, we will introduce these issues as the notions of guard-precision and guard-recall, a specialisation of data-aware conformance checking for the quantification of data-aware models.

Typically, for step b), finding a control-flow mapping between the event log and model, an alignment procedure [4] is used. An alignment between a log and a model is a case-specific sequence of steps, which are either *sync steps* (an event is matched with an execution step), *log steps* (an event without an execution step), or *model steps* (an execution step without an event). For Petri net-based representations, alignment procedures that find the "optimal" alignments have been studied at length [4], [12], [13], [19], [33], [39].

However, in step c), constructing classification problems for each decision point, we generalise the alignment to include only *sync* and *log* steps. Such generalisation ensures that only behaviour observed in the log is used in learning guards for the model. We refer to this generalisation of a control-flow alignment as a "*matching*" and use it throughout the rest of the paper.

III. RELATED WORK

In this section, we discuss techniques for discovering data-aware models (i.e. decision mining techniques), and existing frameworks for comparing these techniques. Then, we provide an in-depth discussion of the data-aware conformance checking papers published to date.

Decision mining was first presented in [37], where token replay was used to generate examples for classification problems that were solved using a decision tree, after which, the decision tree could be used to generate a 'human' readable expression based on process attributes. These expressions were then attached to transitions in a Petri net to denote why a process activity would occur based on data. The next advancement came in [26], where the authors used alignments [4] to generate examples for the classification problems. This approach has greater flexibility than token replay and can handle unfitting event logs and process models. In [31], the author presented a strategy for extracting expressions, where 'overlapping' rules could be constructed. Lastly, in [38], the authors used a variety of preprocessing steps to include time series data in decision mining techniques.

Several studies have used decision mining to understand decision-making within processes. In [14], the author considered patient triage in an emergency department and used decision mining to determine age groups for triage behaviour. In this study, F_1 score was used to evaluate effectiveness locally. Similarly in [32], the authors considered a Dutch emergency department and worked with domain experts to understand patient readmission using decision mining. In this case, the conformance measure presented in [33] was used to evaluate effectiveness. In contrast, [8] used decision mining to identify business rules between process variants, comparing the similarity of generated expressions.

In [22], the authors noted the difficulty in comparing decision mining techniques, as many tailored their evaluation for the case; thus, they presented a framework to evaluate and compare techniques in an artificial setting using simulation. The difficultly in comparing outcomes was also noted in [7], where external data sources were integrated into event logs, greatly expanding the state-space that both [33] and [35] use. Additionally, an ad-hoc measure was introduced to study the number and support for the discovered guards.

In summary, many different methods have been used to evaluate the effectiveness of decision mining techniques, but so far, consensus on the quantification of the quality of data-aware models has not been reached.

A. DATA-AWARE CONFORMANCE CHECKING

In terms of data-aware conformance checking, we separate our discussion between verification [15], [17], [18], [36] and the quantification of recall or precision notions [10], [11], [12], [19], [33], [34], where the latter is our focus.

The difference between verification and quantification is that verification focuses on the properties of a model, while quantification is about the distance or disagreement between a log and a model. Verification of properties for a data-aware model like reachability [36], relaxed data soundness [31], data soundness [15], [18] is important to decision mining. In some cases, we can even repair unsound data-aware models into data sound versions [17], which is helpful as none of the aforementioned decision mining techniques [26], [31], [37], [38] provide any guarantees about their outcomes. However, we do not consider verification properties within the scope of our discussion, as they are typically focused on the model rather than log-model comparisons.

We now describe the existing data-aware conformance checking techniques in the literature related to either recall or precision notions [10], [11]. The first data-aware conformance checking technique we are aware of, proposed in de Leoni and van der Aalst [12], uses Integer Linear Programming (ILP) to compute an alignment between trace and model, which accounts for all perspectives. Alongside the alignment procedure proposed in [12], the authors presented a conformance checking technique for recall, that considered the guards of a DPN. This technique applies a staged approach using the following steps: (1) compute a control-flow alignment, (2) an ILP problem is solved for the optimal number of changes needed to make the trace compliant with the model's guards, (3) compute the recall using the harmonic-mean of (1) and (2). A downside of performing (1) and (2) in sequence, is that a sub-optimal alignment could be returned by (1) if it describes an execution which is more costly than another, when both control-flow and guards are considered.

To address this downside, Mannhardt et al. [33] proposed an extended set of alignment moves and computing one holistic alignment, instead of the staged approach in [12]. As such, the technique uses Multiple Integer Linear Programming (MILP) to solve for an alignment which included suitable write conditions for a given trace and DPN. The benefit of this approach was that each alignment move could be given a unique cost. However, this approach needs many more MILP problems to be solved than [12], which is time consuming but always returns the holistic optimal alignment.

Also presented in [33] was a data-aware conformance checking technique for recall. This proposed recall technique [33] computes a ratio between the optimal alignment and the worst alignment for a given trace and a DPN. A downside of this approach is that both the cost-model and guards within the DPN affect the optimal and reference alignment, in turn defining a unique state-space for the resulting recall measurement and making comparison between measurements challenging.

Another extension was presented in Felli et al. [19], which introduces the 'Computing Conformance Modulo Theories' (CoCoMoT) framework to compute alignments in a similar vein to [12] and [33]. The CoCoMoT framework revolves around using Satisfiability Modulo Theories (SMT) and solves an SMT problem for the minimal number of edits to make a trace compliant with a given DPN. We note that this

technique focuses exclusively on the alignment problem and, unlike [12], [33], does not present a recall measure.

The CoCoMoT framework [19] performs the following steps: (1) first, the log is pre-processed to be a set of unique traces, (2) trace clustering is performed to find subsets of equivalence trace classes, (3) a representative is selected from each class, (4) a SMT problem is encoded and solved for each representative and then an alignment is decoded from the SMT solution. After which, the decoded alignment can be used to denote the distance between the original trace and the alignment. The benefit of this approach is that only one problem per representative is needed, reducing the computation needed in comparison to [12] and [33].

The only data-aware conformance checking technique for precision, that we are aware of, is presented in [34]. A requirement of the technique in [34] is that the log must be perfectly compliant with the given model including system behaviour (i.e. silent actions or silent transitions). However, the authors show that this requirement can be overcome by applying the alignment procedure in [33] on the given log. Then, for each state of the model, the technique computes the observed and possible behaviour for that state. After which, a ratio is computed between all observed and all possible behaviour seen at states of the model. A downside of this approach is that due to the alignment performed, the original data in the log may be changed to be compliant to guards. For non-compliant logs, this downside may lead to non-intuitive precision values.

In these existing techniques, the dominant discussion focused on addressing the alignment problem for data-aware models, and to date, little has been presented concerning the quantification of quality for data-aware models. This gap can be seen in our overview, in Table 1, where most work around data-aware models either focus on discovery or methods for efficient quantification without a discussion of desirable properties.

TABLE 1. An overview of the discussed literature categorised into work that only considers control-flow, work that focuses on verifying (data-aware or not) model properties, and those that consider the discovery or conformance of guards in data-aware models. We are unaware of any work that proposes desirable properties for the latter category.

	Control-Flow	Model Focus	Data-Aware
Discovery	[1, 23]	[17]	[26, 31, 37, 38]
Conformance	[2, 4, 6, 10, 11, 40, 41]	[21, 22]	[12, 19, 33, 35]
Desirable Properties	[2, 6, 23, 40, 41]	[15, 17, 18, 36]	**missing**

IV. PRELIMINARIES

In this section, we introduce notations, event logs, and transition trees as a process model formalism.

Functions. To denote a function, we write $f : X \mapsto Z$, where f is a function that maps each element of the set X to an element of the set Z. Function f is a bijection if no two different elements of X are mapped to the same element in Z and all elements in Z have an original, i.e. an element from X that is mapped to it.

Bags. A *bag* over X is a function $f : X \mapsto \mathbb{N}$. Function f captures the frequency of each element from X in the bag. To denote a bag, we use square brackets and only denote non-zero frequencies as superscripts, e.g. $[x_1{}^1, x_2{}^5, x_3{}^{10}]$.

Sequences. We use angle brackets to denote sequences, e.g. $\langle a, b, c\rangle$. A sequence can be constructed by a first element, the head, say h, and another sequence, say t, the tail. We will denote such a sequence as $h : t$. We widen this notation and write $t : h$ to extend t with an element h.

To denote the length of a sequence, say $s = \langle a, b, c\rangle$, we write $|s|$; e.g. $|s| = 3$. To access an element of a sequence s, we write s_i such that $1 \leq i \leq |s|$, e.g. $s_3 = c$. When we want to access the last element of a non-empty sequence s, we will use the function last, defined as: $\mathsf{last}(s) = s_{|s|}$. To denote the concatenation of two sequences, we use the concatenation operator $+$, e.g. $\langle a, b\rangle + \langle c, d\rangle = \langle a, b, c, d\rangle$. To take a segment from position i up to and including position j from a sequence s, we use the notation $s[i : j]$. For example, $\langle a, b, c, d\rangle[2 : 3] = \langle b, c\rangle$. By definition, if $i > j$, $s[i : j] = \langle\rangle$. Also, if $j > |s|$ and $i \leq |s|$, then $s[i : j] = s[i : |s|]$.

Universes. We now define some universes that we will build upon.

Definition 1 (Universes): $\mathcal{U}_{ev}$ is the universe of event identifiers; $\mathcal{U}_{case}$ is the universe of case identifiers; $\mathcal{U}_{act}$ is the universe of process activities; $\mathcal{U}_{time}$ is the universe of timestamps; $\mathcal{U}_{att}$ is the universe of process attribute identifiers; $\mathcal{U}_{val}$ is the universe of possible values that process attributes can take. The latter universe includes $\perp$, a value that can be used to indicate that an attribute is undefined. All these sets are mutually exclusive.

Data. An *assignment* is our representation of data and is a function from process attributes to values.

Definition 2 (Assignments): $\Pi = \mathcal{U}_{att} \mapsto \mathcal{U}_{val}$ is the set of all possible attribute assignments.

As a process executes, assignments need to be correspondingly updated. We use an override $\oplus$ operator as an operator on assignments, where the values of the right assignment take precedence over the left assignment.

Definition 3 (Overriding Assignments): Let $\pi_1, \pi_2 \in \Pi$ be assignments, then $\pi_1 \oplus \pi_2 \in \Pi$ is an assignment defined as follows. For all $a \in \mathcal{U}_{att}$:

$$\pi_1 \oplus \pi_2(a) = \textit{if } \pi_2(a) \neq \perp \textit{ then } \pi_2(a) \textit{ else } \pi_1(a).$$

Guards. Guards are used to guide process execution. For the remainder of the paper, we assume no particular language for the specification of guards. In the context of a particular assignment, a guard may evaluate to *true* or *false* or it may not be possible to evaluate the guard in this context.

Definition 4 (Guards): $\mathcal{U}_{grd}$ is the universe of guards. Then for any guard $g \in \mathcal{U}_{grd}$, we can access the attributes that the guard is formed over through $\mathsf{var} : \mathcal{U}_{grd} \mapsto 2^{\mathcal{U}_{att}}$, and we can evaluate guards in the context of attribute assignments. In particular, for a guard $g \in \mathcal{U}_{grd}$ and an assignment $\pi \in \Pi$,

$[\![g]\!](\pi)$ *denotes the evaluation of guard* g *in the context of assignment* π *and yields true* (T)*, false* (F)*, or undefined* (U).

Events. An event, represented by a unique identifier, describes a state change in a process execution. Each *event identifier* is associated with an assignment of process attributes, a process activity, a case identifier, and a timestamp.

Definition 5 (Events): For every event $e \in \mathcal{U}_{ev}$*, we can access its attribute assignment through the function* $\#\colon \mathcal{U}_{ev} \mapsto \Pi$*, its activity through the function* $\mathsf{act}\colon \mathcal{U}_{ev} \mapsto \mathcal{U}_{act}$*, its case identifier through the function* $\mathsf{case}\colon \mathcal{U}_{ev} \mapsto \mathcal{U}_{case}$*, and its timestamp through the function* $\mathsf{time}\colon \mathcal{U}_{ev} \mapsto \mathcal{U}_{time}$.

Traces. A *sequence* of events σ denotes a *trace* or a single execution of a process if all events – all different – refer to the same case and the time ordering is preserved.

Definition 6 (Trace): A trace σ *is a nonempty sequence of events, i.e.* $\sigma \in \mathcal{U}_{ev}^{+}$*, such that*:

1) All events of σ *belong to the same case*:

$$\forall_{1\le i<j\le|\sigma|}[\mathsf{case}(\sigma_i) = \mathsf{case}(\sigma_j)];$$

2) The sequence of events in σ *adheres to their timing*:

$$\forall_{1\le i<j\le|\sigma|}[\mathsf{time}(\sigma_i) \le \mathsf{time}(\sigma_j)];$$

3) All events in σ *are different*:

$$\forall_{1\le i<j\le|\sigma|}[\sigma_i \neq \sigma_j].$$

For convenience, we write $\mathsf{case}(\sigma)$ to access the case of a trace and to obtain the activity sequence of a trace σ, or the *variant* of the trace, we write $\hat{\sigma} = \langle \mathsf{act}(\sigma_1), \ldots, \mathsf{act}(\sigma_{|\sigma|})\rangle$. Next, to evaluate guards, access is needed to the assignment that holds after a particular position of a trace.

Definition 7 (Running Assignment): Let σ *be a trace, let* $j \in \mathbb{N}$ *be the trace position, and let* $\pi \in \Pi$ *be an initial assignment. Then, the function* μ *yields the running assignment of the trace defined as*:

$$\mu(\pi, \langle\rangle, j) = \pi \text{ and}$$

$$\mu(\pi, e\colon\sigma, j) = \begin{cases} \mu(\pi \oplus \#(e), \sigma, j-1) & \text{if } j > 0, \\ \pi & \text{otherwise.} \end{cases}$$

For the rest of the paper, we write $\mu_{\sigma,j}$ instead of $\mu(\varnothing, \sigma, j)$.

Logs. An *event log* contains the completed executions of a process over some period.

Definition 8 (Event Logs): An event log L *is a set of traces, such that no two traces* $\sigma', \sigma'' \in L, \sigma' \neq \sigma''$*, share a case*:

$$\mathsf{case}(\sigma') \neq \mathsf{case}(\sigma'').$$

At times we are only interested in the control-flow variants observed in an event log.

Definition 9 (Control-flow Variants): Given an event log L*, the set of its control-flow variants* $\mathcal{V}_L \subset \mathcal{U}_{act}^{+}$ *is defined as*:

$$\mathcal{V}_L = \left\{\hat{\sigma} \mid \sigma \in L\right\}.$$

To represent event logs in this paper, we do not show event identifiers, and instead showcase the variants and bags over the sequences of assignments that occurred for those variants. For example see the representation below, which shows a view on an event log with two process variants and two different assignments for one of these variants:

$$\mathrm{L} = \Big\{\langle a,b,c\rangle \mapsto [\langle \pi_1,\pi_2,\pi_3\rangle^1, \langle \pi_1,\pi_4,\pi_3\rangle^3], \langle a,d,c\rangle \mapsto [\langle \pi_1,\pi_5,\pi_3\rangle^2]\Big\}.$$

Enlarging. Often we refer to a log which has been enlarged with respect to distribution of variants and assignments. To be able to enlarge a log, we first need to be able to clone a trace.

Definition 10 (Cloning Traces): Let $\sigma \in \mathcal{U}_{ev}^{+}$ *be a trace. Then,* $\overline{\sigma}$ *is a clone of* σ *which satisfies the following properties*:

1) the clone is also a trace;
2) the length of the trace is preserved: $|\sigma| = |\overline{\sigma}|$*;*
3) the sequence of process activities is preserved: $\hat{\overline{\sigma}} = \hat{\sigma}$*;*
4) the sequence of assignments is preserved:

$$\forall_{1\le i\le|\sigma|}[\#(\sigma_i) = \#(\overline{\sigma}_i)];$$

5) the trace and its clone do not share events:

$$\forall_{1\le i\le|\overline{\sigma}|,1\le j\le|\sigma|}[\overline{\sigma}_i \neq \sigma_j];$$

6) the cloned trace belongs to a different case:

$$\mathsf{case}(\overline{\sigma}) \neq \mathsf{case}(\sigma).$$

Now that we can clone a trace, we can enlarge an event log by cloning the traces in the log a number of times.

Definition 11 (Enlarging Event Logs): Let L *be an event log, and let* $k \in \mathbb{N}$ *be a integer scaling factor s.t.* $k \ge 1$*. For each* $\sigma \in L$*, we assume* $k-1$ *clones:* $\overline{\sigma}_2, \ldots, \overline{\sigma}_k$ *with none of these clones sharing case identifiers or events among them or with any other such clones created for other traces in log* L*. Then,* L^k *denotes the enlarging of the event log via cloning, defined as*:

$$L^k = L \cup \left\{\overline{\sigma}_j \mid \sigma \in L \wedge 2 \le j \le k\right\}.$$

For example, consider the following enlargement of an event log:

$$\textit{if } \mathrm{L} = \Big\{\langle a,b,c\rangle \mapsto [\langle \pi_1,\pi_2,\pi_3\rangle^1, \langle \pi_1,\pi_4,\pi_3\rangle^3], \langle a,d,c\rangle \mapsto [\langle \pi_1,\pi_5,\pi_3\rangle^2]\Big\},$$

$$\textit{then } \mathrm{L}^3 = \Big\{\langle a,b,c\rangle \mapsto [\langle \pi_1,\pi_2,\pi_3\rangle^3, \langle \pi_1,\pi_4,\pi_3\rangle^9], \langle a,d,c\rangle \mapsto [\langle \pi_1,\pi_5,\pi_3\rangle^6]\Big\}.$$

Note that each instance in the bags on the right represents a unique trace; therefore, enlargement of the log in our visualisation is seen as multiplying these frequencies by the enlargement, i.e. $1\cdot 3$, $2\cdot 3$ and $3\cdot 3$.

Trees. We will abstract from a specific representation for process models but assume that they capture control-flow and data dependencies. We will use transition trees [20] as our abstraction to denote process models.

Definition 12 (Models): A process model $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *is a rooted and non-cyclic tree, with* N *a set of nodes,* F *a set of flows, a root node* $n_r \in N$*, a set*

of terminal nodes $N_f \subseteq N$, *a function* $\mathsf{src}\colon F \mapsto N$ *to yield the source node of a flow, a function* $\mathsf{tgt}\colon F \mapsto N$ *to yield the target node of a flow, a function* $\mathsf{gd}\colon F \mapsto \mathcal{U}_{grd}$ *to yield the guard of a flow, and a function* $\mathsf{actf}\colon F \mapsto \mathcal{U}_{act}$ *to yield the activity of a flow.*

In the context of a model $\mathrm{M} = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$, we use the following shorthand notation. If there is an $f \in F$ such that $\mathsf{src}(f) = n$ and $\mathsf{tgt}(f) = n'$ for $n, n' \in N$, then we can write $n \to n'$ and $n \xrightarrow{f} n'$. To access the flows that are outgoing from a given node n, we write $n\bullet$ to denote:

$$n\bullet = \left\{ f \in F \mid n \xrightarrow{f} n' \right\}.$$

As a convenience, we write $\mathsf{splits}(\mathrm{M})$ to denote flows in the process model whose source node is source node of at least one other flow in the model, i.e. flows at splits in the model:

$$\mathsf{splits}(\mathrm{M}) = \{ f \in F \mid |\mathsf{src}(f)\bullet| > 1 \}.$$

To determine whether two process models M and M′ have the same behavioural structure, we require the existence of an isomorphism between them: $\mathrm{M} \simeq \mathrm{M}'$. For an in-depth consideration of isomorphism see [30]. Two process trees are isomorphic if and only if there exists a bijection between their flows and nodes that preserves sources and targets of flows as well as their activities. Such a bijection is termed an isomorphism, denoted as $\mathsf{iso}^{M}_{M'}$ or iso for short, when the models involved are clear. Note that this bijection is only considering the control-flow of the models in question.

V. FOUNDATIONS

This section introduces our formal quantification of quality for data-aware models. First, we introduce an example data-aware model, which we use to demonstrate our transformation to a model-agnostic form for data-aware conformance checking. Next, we denote how we combine guards through binary operators on guards referred to as reinforcing or weakening and their formal properties. These properties allow for simplifications when combining guards from partial executions of a model consisting of infinitely many silent actions (τ) followed by a non-silent action or when we need to denote alternative execution paths leading to the same visible action.

In the following two subsections, we introduce how we produce a model-agnostic form of any executable data-aware process model, such that the form conforms with our conceptual view of a system reacting to an environment (see Fig. 3). Our next step is to introduce how we study the intersection of an event log and our transition tree through a generalisation of alignments called a matching, which only considers synchronised and log moves to avoid the propagation of dead assignments. Lastly, we combine all of these steps to present our quantification mechanism, called bookkeeping, that allows us to say, for a part of our tree, how often it was traversed and how often particular outcomes of the associated guards occurred.

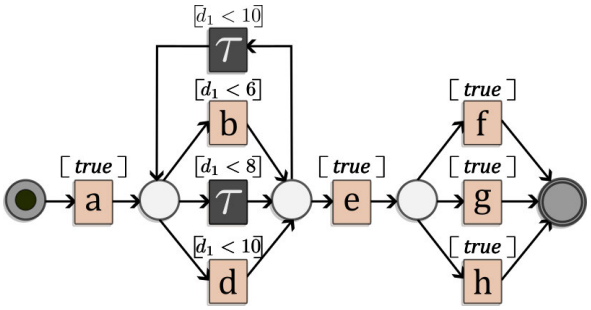


FIGURE 4. Example Petri net with data.

Together, these advancements will be used to present desirable properties for data-aware conformance checking in Section VI and to define guard-recall and guard-precision in Section VII.

A. RUNNING EXAMPLE

In Fig. 4, we introduce a example model for demonstration purposes. This model has one write constraint, i.e. 'a' writes attribute d_1 which has an integer domain. This model contains silent actions (transitions labelled with τ) and a rework loop that can occur silently. In this model, we can see that infinite system behaviour can occur between 'a' and 'e'. For simplicity, we will reference the guards in Fig. 4 in the following manner:

$$\varrho_{\mathsf{T}} \equiv \mathit{true}, \quad g_1 \equiv d_1 < 6, \quad g_2 \equiv d_1 < 8, \quad g_3 \equiv d_1 < 10.$$

B. STRENGTHENING AND WEAKENING GUARDS

We now advance our theory around guards. Our motivation for these advancements is that we will need to unfold executions of a model, and will be required to combine a series of guards into a single guard. Such a case appears in Fig. 4, where the two silent transitions (τ) can be fired many times and we wish to encode their guards alongside the guard of the next visible fired transition (either b, c, or e). However, there can be many partial executions that lead to the firing of 'e' after 'a' in our model, and we wish to capture these cases as well. To be able to encode these cases, we need a simplification process over a series of guards.

Our first step toward this goal is to introduce the concept of neutral guards, which yield the same value regardless of the assignment.

Definition 13 (Neutral Guards): A guard $g \in \mathcal{U}_{grd}$ *is neutral iff for any assignments* $\pi, \pi' \in \Pi$, $[\![g]\!](\pi) = [\![g]\!](\pi')$. *A guard is referred to as truth neutral,* $\mathsf{truth}(g)$, *iff for any assignment* $\pi \in \Pi$, $[\![g]\!](\pi) = \mathsf{T}$, *while it is referred to as undefined neutral,* $\mathsf{undef}(g)$, *iff for any assignment* $\pi \in \Pi$, $[\![g]\!](\pi) = \mathsf{U}$.

While we abstract from the form of guards, we will be required to reason about their evaluation behaviour. Thus, we now formally introduce equivalence between two guards.

Definition 14 (Equivalent Guards): Guards $g_1, g_2 \in \mathcal{U}_{grd}$ *are equivalent,* $g_1 \equiv g_2$, *iff for all* $\pi \in \Pi$, $[\![g_1]\!](\pi) = [\![g_2]\!](\pi)$.

To create stricter guards, we introduce the *reinforce* operator $\otimes$. Reinforcing a guard with another, means that the reinforcement is a guard that returns undefined if *either* guard is undefined, returns true if *both* guards are true, otherwise it returns false.

Definition 15 (Reinforcing Guards): Let $g_1, g_2 \in \mathcal{U}_{grd}$ be guards, and let $\pi \in \Pi$ be an assignment. Then, we define the reinforce operator $\otimes$ over guards as:

$$[\![g_1 \otimes g_2]\!](\pi) = \begin{cases} \mathsf{U} \text{ if } [\![g_1]\!](\pi) = \mathsf{U} \text{ or } [\![g_2]\!](\pi) = \mathsf{U}, \\ \mathsf{T} \text{ if } [\![g_1]\!](\pi) \text{ and } [\![g_2]\!](\pi), \\ \mathsf{F} \text{ otherwise}. \end{cases}$$

From this definition, the following proposition immediately follows.

Proposition 1 (Properties of $\otimes$): Let $g_1, g_2, g_3, \varrho_{\mathsf{T}} \in \mathcal{U}_{grd}$ be guards such that $\mathsf{truth}(\varrho_{\mathsf{T}})$. Then, it follows that the reinforce operator $\otimes$ has the following properties:

a. if $g_1 \equiv g_2$ then $g_1 \otimes g_2 \equiv g_1$; (*idempotent*)
b. $g_1 \otimes g_2 \equiv g_2 \otimes g_1$; (*commutative*)
c. $(g_1 \otimes g_2) \otimes g_3 \equiv g_1 \otimes (g_2 \otimes g_3)$; (*associative*)
d. $g_1 \otimes \varrho_{\mathsf{T}} \equiv \varrho_{\mathsf{T}} \otimes g_1 \equiv g_1$. (*neutral element*)

Using these properties, then the following reduction can be applied to any reinforcement of guards.

Lemma 1 (Strong Idempotence of $\otimes$): Let $g_1, \ldots, g_n \in \mathcal{U}_{grd}$ be guards and let $g \in \mathcal{U}_{grd}$ be another guard, then if $g \equiv g_i$ for some $1 \leq i \leq n$:

$$\bigotimes_{1 \leq i \leq n} g_i \otimes g \equiv \bigotimes_{1 \leq i \leq n} g_i.$$

To create laxer guards, we introduce the *weaken* operator $\ominus$. Weakening a guard with another, means that the weakening is a guard that returns undefined if *both* guards are undefined, it returns true if *either* guard is true, otherwise it returns false.

Definition 16 (Weakening Guards): Let $g_1, g_2 \in \mathcal{U}_{grd}$ be guards, and let $\pi \in \Pi$ be an assignment. Then we define the weaken operator $\ominus$ over guards as:

$$[\![g_1 \ominus g_2]\!](\pi) = \begin{cases} \mathsf{U} \text{ if } [\![g_1]\!](\pi) = \mathsf{U} \text{ and } [\![g_2]\!](\pi) = \mathsf{U}, \\ \mathsf{T} \text{ if } [\![g_1]\!](\pi) \text{ or } [\![g_2]\!](\pi), \\ \mathsf{F} \text{ otherwise}. \end{cases}$$

Similar to before, the following proposition arises from Def. 16.

Proposition 2 (Properties of $\ominus$): Let $g_1, g_2, g_3, \varrho_{\mathsf{U}} \in \mathcal{U}_{grd}$ be guards such that $\mathsf{undef}(\varrho_{\mathsf{U}})$. Then, it follows that the weaken operator $\ominus$ has the following properties:

a. if $g_1 \equiv g_2$ then $g_1 \ominus g_2 \equiv g_1$; (*idempotent*)
b. $g_1 \ominus g_2 \equiv g_2 \ominus g_1$; (*commutative*)
c. $(g_1 \ominus g_2) \ominus g_3 \equiv g_1 \ominus (g_2 \ominus g_3)$; (*associative*)
d. $g_1 \ominus \varrho_{\mathsf{U}} \equiv \varrho_{\mathsf{U}} \ominus g_1 \equiv g_1$. (*neutral element*)

Using these properties, we can always apply the following reduction to any weakening of guards.

Lemma 2 (Strong Idempotence of $\ominus$): Let $g_1, \ldots, g_n \in \mathcal{U}_{grd}$ be guards and let $g \in \mathcal{U}_{grd}$ be another guard, then if $g \equiv g_i$ for some $1 \leq i \leq n$:

$$\bigominus_{1 \leq i \leq n} g_i \ominus g \equiv \bigominus_{1 \leq i \leq n} g_i.$$

1) EXAMPLE REDUCTION

Let us consider the following reinforcement r over the guards from our running example (Fig. 4), where $\mathsf{truth}(\varrho_{\mathsf{T}})$:

$$r = (((\varrho_{\mathsf{T}} \otimes g_2) \otimes g_3) \otimes g_2) \otimes g_3.$$

Now, we consider if r is in its simple form, i.e. contains the minimal amount of reinforcement to represent the same behaviour. To do this, we apply Lemma 1 and Prop. 1 as seen below:

$$\begin{aligned} r &= (((\varrho_{\mathsf{T}} \otimes g_2) \otimes g_3) \otimes g_2) \otimes g_3 \\ &\quad (\textit{neutral element}) \\ &= ((g_2 \otimes g_3) \otimes g_2) \otimes g_3 \\ &\quad (\textit{commutative}) \\ &= ((g_3 \otimes g_2) \otimes g_2) \otimes g_3 \\ &\quad (\textit{associative}) \\ &= (g_3 \otimes (g_2 \otimes g_2)) \otimes g_3 \\ &\quad (\textit{idempotence}) \\ &= (g_3 \otimes g_2) \otimes g_3 \\ &\quad (\textit{strong idempotence}) \\ &= g_3 \otimes g_2 \end{aligned}$$

The above reduction is important to show, as Fig. 4 may produce infinite reinforcement of guards from system behaviour. But, we can always simplify these many infinite reinforcement down to handful of finite reinforcements.

C. PLAYING OUT A MODEL

We now show how we abstract away from the representation of a model through the executable play-outs from the model.

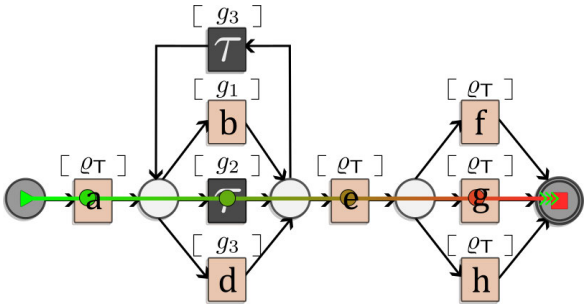
A single execution of a data-aware model generates two sequences of the same length, one containing activities and another of guards. However, the sequence of activities may include silent actions, which cannot be encoded into a transition tree. Therefore, we introduce $\mathsf{playout}$ to proliferate guards from silent activities to non-silent activities. This proliferation creates a play-out trace and to denote proper termination, we use a halt symbol denoted as '*halt*'.

Definition 17 (Play-out Traces): Let $a \in \mathcal{U}_{act} \cup \{\tau\}$ be an activity or the silent action, and $tl_a \in (\mathcal{U}_{act} \cup \{\tau\})^*$ be a sequence of activities with silent actions (τ), $g \in \mathcal{U}_{grd}$ be a guard, $tl_g \in \mathcal{U}^*_{grd}$ be a sequence of guards such that $|a : tl_a| = |g : tl_g|$, and $g', \varrho_{\mathsf{T}} \in \mathcal{U}_{grd}$ be guards such that $\mathsf{truth}(\varrho_{\mathsf{T}})$. Then, to compute a play-out trace using these sequences, we use the function $\mathsf{playout}$ defined as follows:

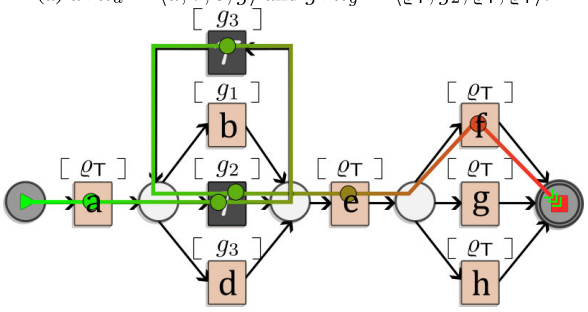
$$\begin{aligned} &\mathsf{playout}(\langle\rangle, \langle\rangle, g') = \langle(\mathit{halt}, \varrho_{\mathsf{T}})\rangle \text{ and} \\ &\mathsf{playout}(a : tl_a, g : tl_g, g') \\ &= \begin{cases} (a, g' \otimes g) : \mathsf{playout}(tl_a, tl_g, \varrho_{\mathsf{T}}) & \text{if } a \neq \tau, \\ \mathsf{playout}(tl_a, tl_g, g' \otimes g) & \text{otherwise}. \end{cases} \end{aligned}$$

1) EXAMPLE PLAY-OUT

For example, in Fig. 5 we show two execution of the models and the associated sequences of activities and guards. Let us



(a) $a : tl_a = \langle a, \tau, e, g \rangle$ and $g : tl_g = \langle \varrho_\mathsf{T}, g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle$.



(b) $a : tl_a = \langle a, \tau, \tau, \tau, e, f \rangle$ and $g : tl_g = \langle \varrho_\mathsf{T}, g_2, g_3, g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle$.

FIGURE 5. Example executions for our running example.

consider how these executions are handled by the playout function. Starting with the execution described in Fig. 5a, where we apply Lemma 1 and Prop. 1 as we denote the recursion of the playout function below:

$$
\begin{aligned}
&\mathsf{playout}(a : \langle \tau, e, g \rangle, \varrho_\mathsf{T} : \langle g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T}) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(\tau : \langle e, g \rangle, g_2 : \langle \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T}) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(e : \langle g \rangle, \varrho_\mathsf{T} : \langle \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T} \otimes g_2) \\
&= (a, \varrho_\mathsf{T}) : ((e, \varrho_\mathsf{T} \otimes g_2) : \mathsf{playout}(g : \langle \rangle, \varrho_\mathsf{T} : \langle \rangle, \varrho_\mathsf{T})) \\
&= (a, \varrho_\mathsf{T}) : ((e, \varrho_\mathsf{T} \otimes g_2) : ((g, \varrho_\mathsf{T}) : \mathsf{playout}(\langle \rangle, \langle \rangle, \varrho_\mathsf{T}))) \\
&= (a, \varrho_\mathsf{T}) : ((e, \varrho_\mathsf{T} \otimes g_2) : ((g, \varrho_\mathsf{T}) : \langle (\mathit{halt}, \varrho_\mathsf{T}) \rangle)) \\
&= \langle (a, \varrho_\mathsf{T}), (e, \varrho_\mathsf{T} \otimes g_2), (g, \varrho_\mathsf{T}), (\mathit{halt}, \varrho_\mathsf{T}) \rangle
\end{aligned}
$$

In the recursion, some reinforcements were reduced, e.g. $\varrho_\mathsf{T} \otimes g_2 \otimes \varrho_\mathsf{T}$ *to* $\varrho_\mathsf{T} \otimes g_2$ or $\varrho_\mathsf{T} \otimes \varrho_\mathsf{T}$ *to* ϱ_T. We now perform the same demonstration for the execution described in Fig. 5b:

$$
\begin{aligned}
&\mathsf{playout}(a : \langle \tau, \tau, \tau, e, f \rangle, \varrho_\mathsf{T} : \langle g_2, g_3, g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T}) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(\tau : \langle \tau, \tau, e, f \rangle, g_2 : \langle g_3, g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T}) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(\tau : \langle \tau, e, f \rangle, g_3 : \langle g_2, \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T} \otimes g_2) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(\tau : \langle e, f \rangle, g_2 : \langle \varrho_\mathsf{T}, \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T} \otimes g_2 \otimes g_3) \\
&= (a, \varrho_\mathsf{T}) : \mathsf{playout}(e : \langle f \rangle, \varrho_\mathsf{T} : \langle \varrho_\mathsf{T} \rangle, \varrho_\mathsf{T} \otimes g_2 \otimes g_3) \\
&= (a, \varrho_\mathsf{T}) : ((e, \varrho_\mathsf{T} \otimes g_2 \otimes g_3) : \mathsf{playout}(f : \langle \rangle, \varrho_\mathsf{T} : \langle \rangle, \varrho_\mathsf{T})) \\
&= (a, \varrho_\mathsf{T}) : ((e, \varrho_\mathsf{T} \otimes g_2 \otimes g_3) : ((f, \varrho_\mathsf{T}) : \langle (\mathit{halt}, \varrho_\mathsf{T}) \rangle)) \\
&= \langle (a, \varrho_\mathsf{T}), (e, \varrho_\mathsf{T} \otimes g_2 \otimes g_3), (f, \varrho_\mathsf{T}), (\mathit{halt}, \varrho_\mathsf{T}) \rangle
\end{aligned}
$$

D. CONSTRUCTING A TREE

We now introduce how derive a transition tree from the executable play-out traces of a model. Firstly, we formally introduce a play-out log from a process model and access functions for play-out traces.

Definition 18 (Play-out Logs): Let $\varrho_\mathsf{T} \in \mathcal{U}_{grd}$ *be a guard such that* $\mathsf{truth}(\varrho_\mathsf{T})$*, let M be a model, and let* $I \subseteq (\mathcal{U}_{act} \cup \{\tau\})^+ \times \mathcal{U}^+_{grd}$ *denote the set of possible execution instances generated from the model. Then, the play-out log* $L_M \subset (\mathcal{U}_{act} \times \mathcal{U}_{grd})^+$ *for M is defined as*:

$$L_M = \left\{ \mathsf{playout}(s_a, s_g, \varrho_\mathsf{T}) \mid (s_a, s_g) \in I \right\}.$$

To denote a play-out trace, we write ρ. Furthermore, we will use access functions, poact and pogd, for play-out traces to retrieve either the activities or guards within the play-out.

Definition 19 (Accessing Play-out Traces): Let L_M *be a play-out log, and let* $\rho \in L_M$ *be a play-out trace, then, we access the variant for the play-out trace using the function* $\mathsf{poact} : L_M \mapsto \mathcal{U}^+_{act}$*, and we access the sequence of guards for the play-out trace using the function* $\mathsf{pogd} : L_M \mapsto \mathcal{U}^+_{grd}$*. Both of these functions,* poact *and* pogd *are defined as*:

$$\mathsf{poact}(\langle\rangle) = \langle\rangle \textit{ and } \mathsf{poact}((a, g) : \rho) = a : \mathsf{poact}(\rho),$$
$$\mathsf{pogd}(\langle\rangle) = \langle\rangle \textit{ and } \mathsf{pogd}((a, g) : \rho) = g : \mathsf{pogd}(\rho).$$

We now formally introduce a directed acyclic graph (*dag*) which has been derived from a play-out log. To clarify, our graphs are always constructed in the context of an event log, as we use the longest observed trace to limit the construction of a possible infinite transition tree. By using the longest observed trace length, we also limit the length of the play-out traces used to derive these graphs.

Definition 20 (Play-out Dags): Let M be a model, let L_M *be a play-out log of that model, let k be an integer for the longest length of observed traces, then a dag from the play-out log,* $\mathsf{dag}_{L_M} = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$*, is defined as:*

$$n_r = \langle\rangle, \tag{1}$$
$$N = \{\mathsf{poact}(\rho)[1 : i] \mid \rho \in L_M \wedge 0 < i < min(k + 1, |\rho|)\} \cup \{n_r\}, \tag{2}$$
$$N_f = \{\mathsf{poact}(\rho)[1 : i] \mid \rho \in L_M \wedge 0 \leq i < min(k + 1, |\rho|) \wedge \mathsf{poact}(\rho)_{i+1} = \mathit{halt}\}, \tag{3}$$
$$F = \{(\mathsf{poact}(\rho)[1 : i], (\mathsf{poact}(\rho)_{i+1}, \mathsf{pogd}(\rho)_{i+1}), \mathsf{poact}(\rho)[1 : i + 1]) \mid \rho \in L_M \wedge 0 \leq i < min(k + 1, |\rho| - 1)\}. \tag{4}$$

However, this definition does have a drawback, it can always produce a directed acyclic graph but not always a tree. If the play-out log comes from a process model with silent actions or duplicated activities, our approach may define more than one flow between two nodes in the graph. Having multiple flows between the same two nodes is problematic for our discussions of precision as we need to consider if a path of the graph only had guards evaluated to be true for the taken path. Hence, we now introduce a flow reduction step to ensure only one flow exists between nodes, which when combined with Def. 20 always reduces a play-out dag to a play-out tree.

Definition 21 (Flow Reduction): *Let* $\mathsf{dag}_{L_M} = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *be a play-out dag, let* $A = \{\mathsf{actf}(f) \mid f \in F\}$ *be the set of activity labels, and* $a \in A$ *be a label. Then,* $\mathsf{tree}_{L_M} = (N, F', n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *is the play-out tree of* dag_{L_M} *where* F *is replaced with* F'*, such that only one flow exists in* F' *between two nodes* $n, n{:}a \in N$*, which already had at least one flow between them in* F*, defined as:*

$$F' = \{(n, (a, \bigominus_{f \in F, n \xrightarrow{f} n:a} \mathsf{gd}(f)), n{:}a) \mid n \in N \wedge n \rightarrow n{:}a\}.$$

For the rest of this paper, we refer to a *play-out tree* as a play-out dag constructed from Def. 20 that has had flow reduction (Def. 21) applied to its flows.

1) EXAMPLE TREE

Continuing with our running example, we now consider generating a play-out log for Fig. 4 and constructing a play-out tree. For construction of our tree, let us assume that the longest trace length is four, i.e. for Def. 20, we assume $k = 4$.

We show a partial derivation of the play-log $\mathrm{L_M}$ for Fig. 4 below, with the caveat that we will only need play-out traces that are shorter than six elements:

$$\begin{aligned}
\mathrm{L_M} = \{&\langle (a, g_\mathsf{T}), (b, g_1), (e, g_\mathsf{T}), (f, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (b, g_1), (e, g_\mathsf{T}), (g, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (b, g_1), (e, g_\mathsf{T}), (h, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (e, g_2 \otimes g_\mathsf{T}), (f, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (e, g_2 \otimes g_\mathsf{T}), (g, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (e, g_2 \otimes g_\mathsf{T}), (h, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_3), (e, g_\mathsf{T}), (f, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_3), (e, g_\mathsf{T}), (g, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_3), (e, g_\mathsf{T}), (h, g_\mathsf{T}), (halt, \varrho_\mathsf{T})\rangle\\
&\langle (a, g_\mathsf{T}), (b, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (f, g_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (b, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (g, g_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (b, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (h, g_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (f, g_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (g, g_\mathsf{T})\rangle,\\
&\langle (a, g_\mathsf{T}), (d, g_1), (b, g_3 \otimes g_1), (e, g_\mathsf{T}), (h, g_\mathsf{T})\rangle, \ldots\}.
\end{aligned}$$

Next, we showcase the derivation of Def. 20 in Fig. 6 and the resulting directed graph using the play-out log. This demonstration shows that the flow reduction step is required to ensure tree is constructed, as Fig. 6h clearly shows that many flows can exist between two nodes. The comparison between the resulting directed graph produced by Def. 20, and the flow reduced tree using Def. 21 is shown in Fig. 7. In this comparison, the flows which have been reduced into a single flow are highlighted in red.

E. FINDING A MATCHING

In this section, we introduce the notion of a *matching* between variants of a log and paths in a model, i.e. a control-flow focused alignment which only consists of synchronised and log moves (as per our problematisation in Section II).

Paths. For a particular process model M, the set of paths-with-skips through this model is denoted as P^M. We will use P if the context is clear. Each $p \in P$ takes a particular form: a sequence of flows potentially alternated with skips, i.e. $P \subseteq (F \cup \{\gg\})^*$; with $\gg$ denoting the skip symbol, a recognition that we cannot always follow a particular step in the model. Note that paths in M do not have skips (treating M as a graph), but paths through M may have them. The function noskip removes all skips from a path and is defined as follows:

$$\mathsf{noskip}(\langle\rangle) = \langle\rangle \textit{ and}$$
$$\mathsf{noskip}(s{:}p) = \begin{cases} s{:}\mathsf{noskip}(p) & \textit{if } s \neq \gg, \\ \mathsf{noskip}(p) & \textit{otherwise.} \end{cases}$$

Matching. A matching function between a log and a model captures how variants in the log are mapped onto paths in the model. This mapping may not be perfect, i.e. it may imply that guards are violated, process activities may not match between events and flows, or some paths may not end in final nodes. Note, that at this stage we are not placing any restrictions on the matching to be optimal. Furthermore, a matching is a generalisation of alignments [4], which only considers log models (i.e. $\gg$) or sync moves (i.e the path denotes a flow) which may or may not be ideal.

Definition 22 (Matchings): *Let L be a log, let* $\mathcal{V}_L$ *be the set of variants captured in L, let* $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *be a model, and let* $P \subset (F \cup \{\gg\})^*$ *be the set of paths through M. Then, a matching function* $\gamma : \mathcal{V}_L \mapsto P$ *maps control-flow variants to paths in the model. For any variant* $\hat{\sigma} \in \mathcal{V}_L$ *and its reduced path* $p = \mathsf{noskip}(\gamma(\hat{\sigma}))$ *the following holds*:

1) *the reduced path always starts at the root: if* $|p| > 0$ *then* $n_r \xrightarrow{p_1} \mathsf{tgt}(p_1)$*;*
2) *the reduced path is between connected flows: for all* $1 \leq i < |p| : \mathsf{src}(p_i) \xrightarrow{p_i} \mathsf{src}(p_{i+1})$*;*
3) *and the length of the path is never longer than the variant:* $|\gamma(\hat{\sigma})| \leq |\hat{\sigma}|$.

1) ONE TO MANY MATCHINGS

We now describe finding a suitable collection of candidate paths to use as a matching function, where the goal is to find "optimal" paths through the model for a given variant. To find these paths one might use an alignment procedure to efficiently find an "optimal" path for a variant, but in this section we abstract from the method used. Instead, we focus on describing a 'many matching', which produces all paths that are least costly or optimal in the best case. The difference between a matching and a many matching, is that the former is a one-to-one while the latter is a one-to-many relationship.

Costing. To consider if a path is optimal for a variant, we consider the quality of this pairing. The quality of the pairing is impacted by differences in length, nonmatching activities at corresponding positions, and improper termination. We use the function cost to determine this quality, where

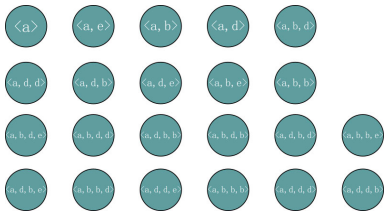
$$n_r = \langle\rangle$$

(a) The root node of the play-out dag.

(b) Tree components related to root node.

$$\{\langle a\rangle, \langle a,b\rangle, \langle a,d\rangle, \langle a,e\rangle, \langle a,b,b\rangle, \langle a,b,d\rangle,$$
$$\langle a,b,e\rangle, \langle a,d,d\rangle, \langle a,d,b\rangle, \langle a,d,e\rangle, \langle a,b,b,b\rangle,$$
$$\langle a,b,b,d\rangle, \langle a,b,b,e\rangle, \langle a,b,d,b\rangle, \langle a,b,d,d\rangle,$$
$$\langle a,b,d,e\rangle, \langle a,d,d,b\rangle, \langle a,d,d,d\rangle, \langle a,d,d,e\rangle,$$
$$\langle a,d,b,b\rangle, \langle a,d,b,d\rangle, \langle a,d,b,e\rangle\}$$

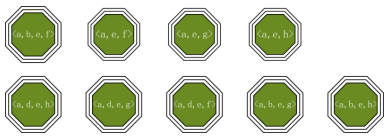
(c) The set non-terminal nodes for the play-out dag.



(d) Tree components related to non-terminal nodes.

$$N_f = \{\langle a,e,f\rangle, \langle a,e,g\rangle, \langle a,e,h\rangle,$$
$$\langle a,b,e,f\rangle, \langle a,b,e,g\rangle, \langle a,b,e,h\rangle,$$
$$\langle a,d,e,f\rangle, \langle a,d,e,g\rangle, \langle a,d,e,h\rangle\}$$

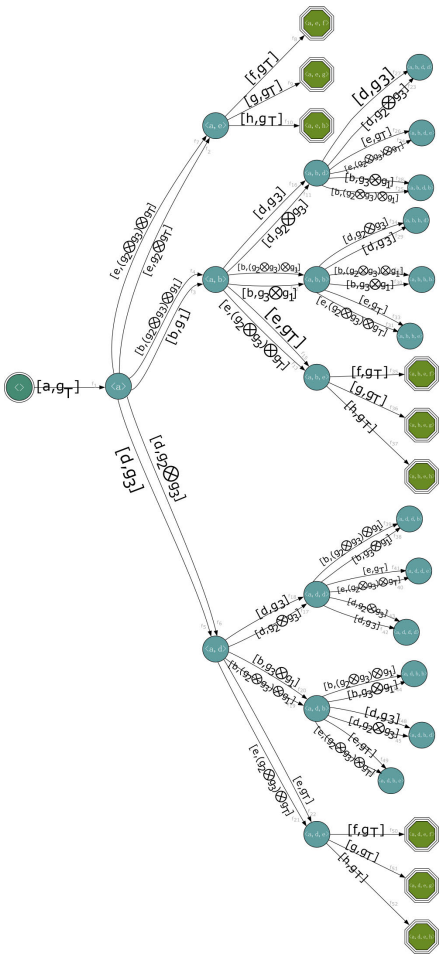
(e) The set of terminal nodes for the play-out dag.



(f) Tree components related to terminal nodes .

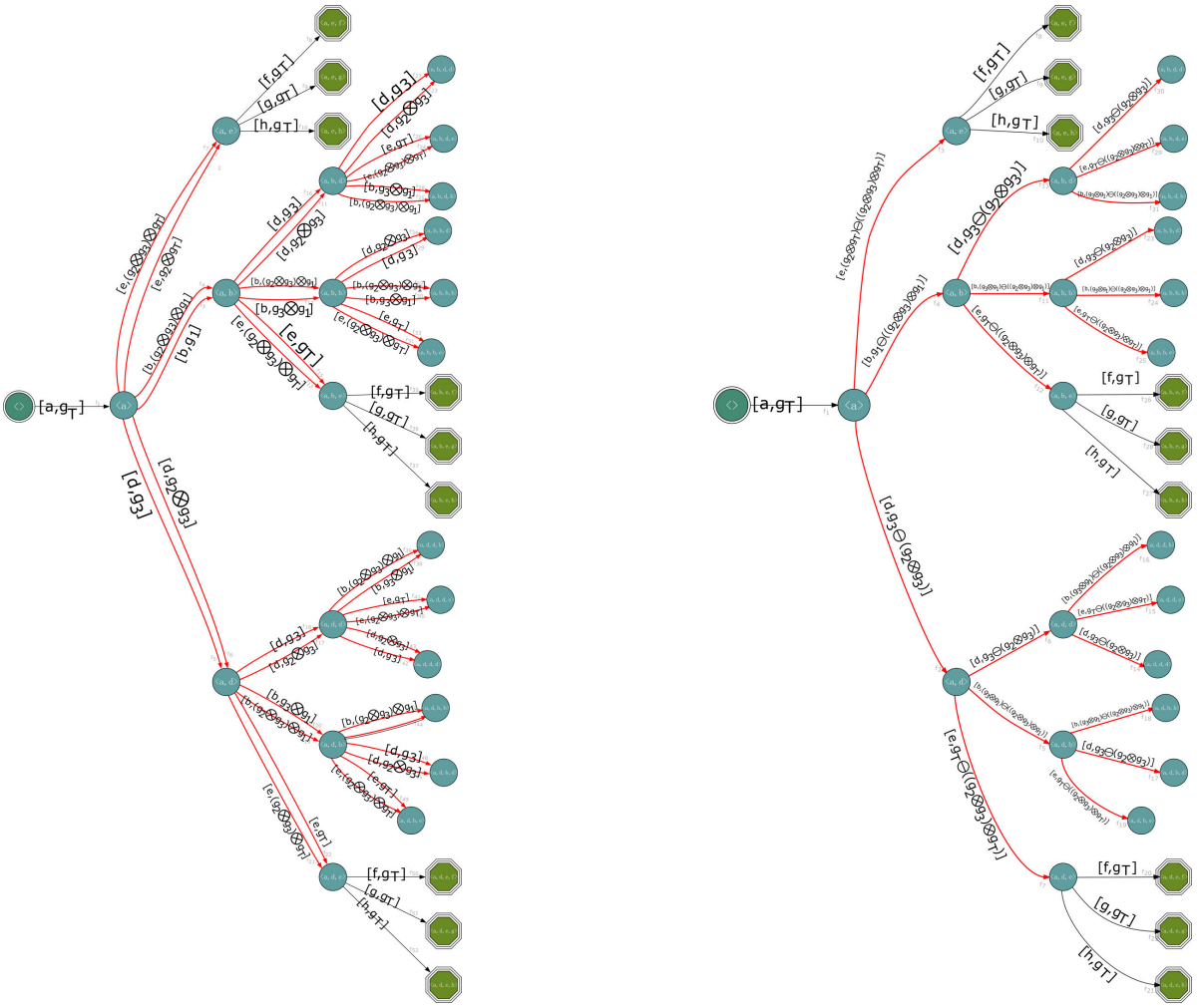
$$F = \{(\langle\rangle, (a, g_\top), \langle a\rangle), (\langle a\rangle, (b, g_1), \langle a,b\rangle),$$
$$(\langle a\rangle, (e, g_2 \otimes g_\top), \langle a,e\rangle),$$
$$(\langle a\rangle, (b, (g_2 \otimes g_3) \otimes g_1), \langle a,b\rangle),$$
$$(\langle a\rangle, (e, (g_2 \otimes g_3) \otimes g_\top), \langle a,e\rangle),$$
$$(\langle a\rangle, (d, g_3), \langle a,d\rangle),$$
$$(\langle a\rangle, (d, g_2 \otimes g_3), \langle a,d\rangle),$$
$$(\langle a,b\rangle, (b, g_3 \otimes g_1), \langle a,b,b\rangle),$$
$$(\langle a,b\rangle, (b, (g_3 \otimes g_2) \otimes g_1), \langle a,b,b\rangle),$$
$$(\langle a,b\rangle, (d, g_3), \langle a,b,d\rangle),$$
$$(\langle a,b\rangle, (e, g_\top), \langle a,b,e\rangle),$$
$$(\langle a,b\rangle, (d, g_3 \otimes g_2), \langle a,b,d\rangle),$$
$$(\langle a,b\rangle, (e, (g_3 \otimes g_2) \otimes g_\top), \langle a,b,e\rangle),$$
$$(\langle a,d\rangle, (d, g_3), \langle a,d,d\rangle),$$
$$(\langle a,d\rangle, (d, g_3 \otimes g_2), \langle a,d,d\rangle),$$
$$(\langle a,d\rangle, (b, g_3 \otimes g_1), \langle a,d,b\rangle),$$
$$(\langle a,d\rangle, (b, (g_3 \otimes g_2) \otimes g_1), \langle a,d,b\rangle),$$
$$(\langle a,d\rangle, (e, g_\top), \langle a,d,e\rangle),$$
$$(\langle a,d\rangle, (e, (g_3 \otimes g_2) \otimes g_\top), \langle a,d,e\rangle),$$
$$(\langle a,e\rangle, (f, g_\top), \langle a,e,f\rangle),$$
$$(\langle a,e\rangle, (g, g_\top), \langle a,e,g\rangle),$$
$$(\langle a,e\rangle, (h, g_\top), \langle a,e,h\rangle),$$
$$\ldots,$$
$$(\langle a,d,e\rangle, (h, g_\top), \langle a,d,e,h\rangle)\}$$

(g) Partial derivation of flows for the play-out dag.



(h) Connected directed graph without flow reduction.

FIGURE 6. **A step-wise introduction to our derivation of a play-out tree from a set of play-out traces.**



(a) play-out dag without flow reduction.

(b) Reduced play-out tree using flow reduction.

FIGURE 7. **Comparison of outcomes after flow reduction has occurred. Changed flows are highlighted in red.**

cost yields 0 to denote an optimal pairing and anything else denotes a less-than-optimal pairing.

Definition 23 (Cost of a Path): Let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *be a play-out tree, let* $p \in P_M$ *be path through this tree, and let* $\hat{\sigma} \in \mathcal{U}^+_{act}$ *be a variant. Then,* cost *is a function that yields a non-zero integer for a non-optimal path or zero for an optimal path*:

$$\mathsf{cost}(p, \hat{\sigma}) = ||\hat{\sigma}| - |\mathsf{noskip}(p)|| + \sum_{1 \le i \le |p|} \left[\mathsf{actf}(p_i) \neq \hat{\sigma}_i\right] + \left[\mathsf{tgt}(\mathsf{last}(\mathsf{noskip}(p))) \notin N_f\right].$$

Now, we formally denote our process for selecting candidate paths, which prioritises the optimal paths over non-optimal paths and results in a many matching Γ. Furthermore, in-line with Def. 22 of a matching, we only consider paths through the model of at most the length of the variant. To denote this process, we use the function Γ, which takes a variant $\hat{\sigma}$, and yields the set of the least costly paths through a model for a variant.

Definition 24 (Many Matching): Let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ *be a play-out tree, and let* P *be the set of all paths through and in* M*, and let* $\hat{\sigma} \in \mathcal{U}^+_{act}$ *be a variant, then we define*:

$$\Gamma(i, \hat{\sigma}) = \left\{s \in P \mid \mathsf{cost}(s, \hat{\sigma}) = i \wedge |s| \le |\hat{\sigma}|\right\}.$$

Now $\Gamma(\hat{\sigma}) = \Gamma(i, \hat{\sigma})$ *for the minimal* $i \in \mathbb{N}$ *for which* $\Gamma(i, \hat{\sigma}) \neq \varnothing$.

Weighing. At times, we will need to be able to determine the likelihood of a path in the context of a variant. As such we now introduce a weighting function ω, which in the context of a set of paths T and a given path p and a variant $\hat{\sigma}$, yields the likelihood that the path is the true system path for the variant. Thus, if the given set of paths only contains optimal paths, ω acts as a full-distribution, i.e. $\sum_{p \in T} \omega(\hat{\sigma}, p) = 1$, otherwise ω acts as sub-distribution, i.e. $\sum_{p \in T} \omega(\hat{\sigma}, p) < 1$.

Definition 25 (Weighting Paths): Let L *be an event log, let* $\hat{\sigma} \in \mathcal{V}_L$ *be a variant, let* M *be a play-out tree, let* $T = \Gamma(\hat{\sigma})$ *be the least costly paths of the variant through the model, let* $p \in T$ *be one such path, and let* $\kappa \in \mathbb{R}$ *denote a small cost-offset such that* $0 < \kappa < 1$*. Then,* $\omega : \mathcal{V}_L \times T \mapsto \mathbb{R}$ *is a function which takes a variant and a path and yields the*

quality of this pairing, defined as:

$$\omega(\hat{\sigma}, p) = \frac{1}{|T|} \cdot \kappa^{\,\mathsf{cost}(p,\hat{\sigma})}.$$

For the rest of the paper, we will assume κ to be 0.9.

2) EXAMPLE MATCHING

We now show the working for constructing the set of likely candidates paths for a single trace, when considering Fig. 7. Consider the following trace, $\sigma = \langle \overset{a}{e_1}, \overset{b}{e_2}, \overset{z}{e_3} \rangle$, and the following partial derivation of P_M which considers paths to nodes and paths to nodes with one skip:

Assuming the following flows from Fig. 7:

$f_1 = \big(\langle\rangle, (a, g_\mathsf{T}), \langle a\rangle\big)$,
$f_2 = \big(\langle a\rangle, (d, (g_3) \ominus (g_2 \otimes g_3)), \langle a, d\rangle\big)$,
$f_3 = \big(\langle a\rangle, (e, (g_2 \otimes g_3) \ominus ((g_2 \otimes g_3) \otimes g_\mathsf{T})), \langle a, e\rangle\big)$,
$f_4 = \big(\langle a\rangle, (b, (g_1) \ominus ((g_2 \otimes g_3) \otimes g_1)), \langle a, b\rangle\big)$,
$f_5 = \big(\langle a, d\rangle, (b, (g_3 \otimes g_1) \ominus ((g_2 \otimes g_3) \otimes g_1)), \langle a, d, b\rangle\big)$,
$f_6 = \big(\langle a, d\rangle, (d, g_3 \ominus (g_2 \otimes g_3)), \langle a, d, d\rangle\big)$,
$f_7 = \big(\langle a, d\rangle, (e, g_\mathsf{T} \ominus ((g_2 \otimes g_3) \otimes g_\mathsf{T})), \langle a, d, e\rangle\big)$,
$f_8 = \big(\langle a, e\rangle, (f, g_\mathsf{T}), \langle a, e, f\rangle\big)$,
$f_9 = \big(\langle a, e\rangle, (g, g_\mathsf{T}), \langle a, e, g\rangle\big)$,
$f_{10} = \big(\langle a, e\rangle, (h, g_\mathsf{T}), \langle a, e, h\rangle\big)$,
$f_{11} = \big(\langle a, b\rangle, (b, (g_3 \otimes g_1) \ominus ((g_2 \otimes g_3) \otimes g_1)), \langle a, b, h\rangle\big)$,
$f_{12} = \big(\langle a, b\rangle, (e, g_\mathsf{T} \ominus ((g_2 \otimes g_3) \otimes g_\mathsf{T})), \langle a, b, e\rangle\big)$,
$f_{13} = \big(\langle a, b\rangle, (d, g_3 \ominus (g_2 \otimes g_3)), \langle a, b, d\rangle\big)$.

Then, we consider a partial derivation of P_M :

$$\begin{aligned} P_M = \{&\langle f_1, f_2, f_5\rangle, \langle f_1, f_2, f_6\rangle, \langle f_1, f_2, f_7\rangle, \langle f_1, f_3, f_8\rangle,\\ &\langle f_1, f_3, f_9\rangle, \langle f_1, f_3, f_{10}\rangle, \langle f_1, f_4, f_{11}\rangle, \langle f_1, f_4, f_{12}\rangle,\\ &\langle f_1, f_4, f_{13}\rangle, \langle \gg\rangle, \langle \gg, f_1\rangle, \langle f_1, \gg\rangle, \langle \gg, f_1, f_2\rangle,\\ &\langle f_1, \gg, f_2\rangle, \langle f_1, f_2, \gg\rangle, \langle \gg, f_1, f_3\rangle, \langle f_1, \gg, f_3\rangle,\\ &\langle f_1, f_3, \gg\rangle, \langle \gg, f_1, f_4\rangle, \langle f_1, \gg, f_4\rangle, \langle f_1, f_4, \gg\rangle, \ldots\}. \end{aligned}$$

For σ, we can see that no optimal path exists as there is no flow labeled with 'z' (f_{1-13}). Moreover, any sequence in P_M will occur a cost from either not being the same length $||\sigma| - |\mathsf{noskip}(p)||$, or from improper termination $\mathsf{tgt}(\mathsf{last}(\mathsf{noskip}(p))) \notin N_f$. Next, we show the working up until Γ yields a non-empty set, with the cost of a candidate displayed above it in red, below:

Applying cost to P_M we see the following:

$$\begin{aligned} P_M = \Big\{&\overset{0+2+1}{\langle f_1, f_2, f_5\rangle}, \overset{0+2+1}{\langle f_1, f_2, f_6\rangle}, \overset{0+2+1}{\langle f_1, f_2, f_7\rangle}, \overset{0+2+0}{\langle f_1, f_3, f_9\rangle},\\ &\overset{0+2+0}{\langle f_1, f_3, f_8\rangle}, \overset{0+2+0}{\langle f_1, f_3, f_9\rangle}, \overset{0+2+0}{\langle f_1, f_3, f_{10}\rangle}, \overset{0+1+1}{\langle f_1, f_4, f_{11}\rangle},\\ &\overset{0+1+1}{\langle f_1, f_4, f_{12}\rangle}, \overset{0+1+1}{\langle f_1, f_4, f_{13}\rangle}, \overset{3+1+1}{\langle \gg\rangle}, \overset{2+2+1}{\langle \gg, f_1\rangle},\\ &\overset{2+1+1}{\langle f_1, \gg\rangle}, \overset{1+3+1}{\langle \gg, f_1, f_2\rangle}, \overset{1+2+1}{\langle f_1, \gg, f_2\rangle}, \overset{1+2+1}{\langle f_1, f_2, \gg\rangle},\\ &\overset{1+3+1}{\langle \gg, f_1, f_3\rangle}, \overset{1+2+1}{\langle f_1, \gg, f_3\rangle}, \overset{1+2+1}{\langle f_1, f_3, \gg\rangle}, \overset{1+3+1}{\langle \gg, f_1, f_4\rangle},\\ &\overset{1+2+1}{\langle f_1, \gg, f_4\rangle}, \overset{1+1+1}{\langle f_1, f_4, \gg\rangle}, \overset{1+0+1}{\langle f_1, f_4\rangle}, \ldots\Big\}. \end{aligned}$$

Then, we incrementally apply Γ to find the minimal i:

$$\begin{aligned} &\Gamma(\sigma, 0) = \varnothing, \quad \Gamma(\sigma, 1) = \varnothing,\\ &\Gamma(\sigma, 2) = \{\langle f_1, f_3, f_8\rangle, \langle f_1, f_3, f_9\rangle, \langle f_1, f_3, f_{10}\rangle,\\ &\qquad \langle f_1, f_4, f_{11}\rangle, \langle f_1, f_4, f_{12}\rangle, \langle f_1, f_4, f_{13}\rangle, \langle f_1, f_4\rangle\}. \end{aligned}$$

Thus, for σ we construct a many-matching $\Gamma(\sigma)$ that returns a set of seven non-optimal paths. Where each candidate path has a weighted share, computed by ω, of 0.115 out of a maximal share of 0.142 if they were optimal paths.

F. DIAGNOSING A MODEL

In this section, we introduce how we quantify the quality of guards, and how to count their evaluations using 'bookkeeping' functions.

Bookkeeping. Our recall discussions consider how often a guard evaluated to a particular outcome when an associated flow is presented in the matching. Thus, the following functions capture information about guard evaluations given a flow and an evaluation outcome and we refer to these functions as traversal bookkeeping functions. In the following definitions, we will use the Iverson bracket: $[\phi] = 1$ if ϕ evaluates to *true* and $[\phi] = 0$ if ϕ evaluates to *false*.

Definition 26 (Traversal Bookkeeping): Let L be a log, let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be a model, let γ be a matching between them, and let $f \in F$ be a flow. Then, $\mathsf{Grd}^{\rightarrow}_{\mathsf{T}}, \mathsf{Grd}^{\rightarrow}_{\mathsf{F}}, \mathsf{Grd}^{\rightarrow}_{\mathsf{U}}$ are functions that compute the traversal bookkeeping for a flow, for a particular outcome of a guard, defined as:

$$\mathsf{Grd}^{\rightarrow}_{\mathsf{T}}(f, \gamma, L) = \sum_{\sigma \in L} \Big[\exists_{1 \le i < |\sigma|} [\gamma_{\hat{\sigma},i} = f \wedge [\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\Big], \tag{5}$$

$$\mathsf{Grd}^{\rightarrow}_{\mathsf{F}}(f, \gamma, L) = \sum_{\sigma \in L} \Big[\exists_{1 \le i < |\sigma|} [\gamma_{\hat{\sigma},i} = f \wedge \neg[\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\Big], \tag{6}$$

$$\mathsf{Grd}^{\rightarrow}_{\mathsf{U}}(f, \gamma, L) = \sum_{\sigma \in L} \Big[\exists_{1 \le i < |\sigma|} [\gamma_{\hat{\sigma},i} = f \wedge [\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1}) = \mathsf{U}]\Big]. \tag{7}$$

Next, for our precision discussions, we need to understand how restricted were choices of the system given the matching and how many times a guard could have been evaluated to be true. Thus, we use exploratory bookkeeping functions to capture information about guard evaluations that could have been evaluated for a given flow, i.e. we consider all paths (and their associated assignments) that visited the source node for the given flow.

Definition 27 (Exploratory Bookkeeping): Let L be a log, let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be a model, let γ be a matching between them, and let $f \in F$ be a flow. Then, $\mathsf{Grd}_{\mathsf{T}}, \mathsf{Grd}_{\mathsf{F}}, \mathsf{Grd}_{\mathsf{U}}$ are functions that compute the exploratory bookkeeping for a flow, for a particular outcome

of a guard, defined as:

$$\mathsf{Grd_T}(f,\gamma,L) = \sum_{\sigma\in L}\Big[\exists_{1\le i<|\sigma|}[\mathsf{src}(\gamma_{\hat{\sigma},i})=\mathsf{src}(f)\wedge[\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\Big], \quad (8)$$

$$\mathsf{Grd_F}(f,\gamma,L) = \sum_{\sigma\in L}\Big[\exists_{1\le i<|\sigma|}[\mathsf{src}(\gamma_{\hat{\sigma},i})=\mathsf{src}(f)\wedge\neg[\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\Big], \quad (9)$$

$$\mathsf{Grd_U}(f,\gamma,L) = \sum_{\sigma\in L}\Big[\exists_{1\le i<|\sigma|}[\mathsf{src}(\gamma_{\hat{\sigma},i})=\mathsf{src}(f)\wedge[\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})=\mathsf{U}]\Big]. \quad (10)$$

Now using the bookkeeping functions, we can determine if two flows from separate models have equivalent bookkeeping.

Definition 28 (Relaxed Guard Equivalence): In the context of model M (with flow f) and M' (with flow f') and with respect to log L with matchings γ (between L and M) and γ' (between L and M'), flows f and f' are relaxed guard equivalent, $\mathsf{GrdEq}(f,f',\gamma,\gamma')$*, iff*:

$$\mathsf{Grd_T}(f,\gamma,L)=\mathsf{Grd_T}(f',\gamma',L)\wedge\mathsf{Grd_F}(f,\gamma,L)=\mathsf{Grd_F}(f',\gamma',L)\wedge\mathsf{Grd_U}(f,\gamma,L)=\mathsf{Grd_U}(f',\gamma',L).$$

In the case that an isomorphism exists between the models, the following corollary exists.

Corollary 1: Guard equivalence implies relaxed guard equivalence. Let L be a log, let $M=(N,F,n_r,N_f,\dots)$ and M' be models such that an isomorphism iso *exists between these models, let γ be a matching between L and M, let γ' be its isomorphic equivalent between L and M', and let $f\in F$ be a flow.*

$$\mathsf{gd}(f)\equiv\mathsf{gd}(\mathsf{iso}(f))\implies\mathsf{GrdEq}(f,\mathsf{iso}(f),\gamma,\gamma').$$

Agreement. While not being the direct goal of data-aware conformance checking, we need to understand the agreement between a log and a model. To this end, we need to know whether events were skipped, whether process activities always matched, and whether traces always ended properly.

Definition 29 (Optimal Matchings): Let L be a log, let M be a model, and let γ be a matching between them. When there are no skips, no activity mismatches, and all traces end in terminal nodes, then the matching is considered optimal:

$$\mathsf{optimal}(L,\gamma,M)\equiv\sum_{\sigma\in L}\mathsf{cost}(\gamma(\hat{\sigma}),\hat{\sigma})=0.$$

VI. QUALITY AXIOMS

In this section we characterise the notions of *guard-recall* and *guard-precision* (grec and gprec). These notions help us position recall and precision in the context of data in a way similar to how recall and precision were positioned in a control-flow context in earlier work [2], [40], [41]. Guard-recall and guard-precision are functions that require a log, a matching, and a model to quantify a value between min and max (inclusive). While the minimal value can be zero and the maximum can be one, we do not require approaches to adhere to this notion, as some metrics take on values greater than one [5]. In this section, we formulate a number of axioms that guard-recall and guard-precision should at least satisfy.

Representation. We follow Proposition 1 presented by Syring, Tax, and van der Aalst [40] and require that guard-recall and guard-precision are deterministic functions, i.e. their outcome is fully determined by their input: the log, the model, and the matching (Axioms 6 and 9, where $k=1$). Proposition 2 of [40] requires independence from process representation. In our case, we have chosen transition trees as an abstraction of process models, so our approach does not fully abide by this proposition. However, regarding the data perspective, we do not prescribe how guards should be formulated and abide by the proposition (Axioms 1 and 2).

Axiom 1 (Inspired by Proposition 2 in [40]): Let L be a log, let $M=(N,F,n_r,N_f,\dots)$ be a model, let M' be another model using a different guard representation such that an isomorphism iso *exists between these models, let γ be a matching between L and M, and let γ' be its isomorphic equivalent between L and M'. Then,* grec *is fully determined by the behaviour observed in the log and the behaviour of guards (representation of guards does not matter):*

$$\forall_{f\in F}\,\mathsf{GrdEq}(f,\mathsf{iso}(f),\gamma,\gamma')\implies\mathsf{grec}(L,\gamma,M)=\mathsf{grec}(L,\gamma',M').$$

Axiom 2 (Inspired by Proposition 2 in [40]): Let L be a log, let $M=(N,F,n_r,N_f,\dots)$ be a model, let M' be another model using a different guard representation such that an isomorphism iso *exists between these models, let γ be a matching between L and M, and let γ' be its isomorphic equivalent between L and M'. Then,* gprec *is fully determined by the behaviour observed in the log and the behaviour of guards (representation of guards does not matter):*

$$\forall_{f\in F}\,\mathsf{GrdEq}(f,\mathsf{iso}(f),\gamma,\gamma')\implies\mathsf{gprec}(L,\gamma,M)=\mathsf{gprec}(L,\gamma',M').$$

Guard-Recall. Ideally, flows with guards that do not evaluate to true are not taken. Guard-recall quantifies if guards have been formed over the observed data and if they evaluate to true when taken. Hence, when comparing similar models with the same event log, guard-recall takes on a larger value for the model with more guard evaluations that were true (Axiom 3). This Axiom implies that if a guard evaluates to false or couldn't be evaluated, then guard-recall is negatively affected.

Axiom 3: Let L be a log, let $M=(N,F,n_r,N_f,\dots)$ be a model, let M' be another model with iso *an isomorphism between these models, let γ be a matching between L and M and let γ' be its isomorphic equivalent between L and M'. Guard-recall takes on a larger value for the model with more guard evaluations that were true*:

$$\sum_{f\in F}\left(\overrightarrow{\mathsf{Grd_T}}(f,\gamma,L)-\overrightarrow{\mathsf{Grd_T}}(\mathsf{iso}(f),\gamma',L)\right)<0\implies\mathsf{grec}(L,\gamma,M)<\mathsf{grec}(L,\gamma',M').$$

Furthermore, if no guard evaluates to true then guard-recall takes on the minimal value (Axiom 4).

Axiom 4: Let L be a log, let $M=(N,F,n_r,N_f,\dots)$ be a model, and let γ be a matching between them. Guard-recall

takes on the minimal value if and only if, no guard was ever evaluated to true when taken:

$$\forall_{f \in F}[\mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma, L) = 0] \iff \mathsf{grec}(L, \gamma, M) = \mathsf{min}.$$

Next, it could be the case, that the given matching is not optimal meaning that some events in the log may not have been considered in the bookkeeping functions. Hence, using a non-optimal matching, opposed to an optimal matching, even when evaluations are similar, will negatively affect guard-recall (Axiom 5).

Axiom 5: Let L be a log, let M be a model, and let γ, γ' be different matching functions between log and model. Guard-recall takes on a larger value for the matching that is optimal, even if evaluations are similar:

$$\Big(\sum_{f \in M} \mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma, L) \leq \sum_{f \in M} \mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma', L) \wedge \neg\mathsf{optimal}(L, \gamma, M) \wedge \mathsf{optimal}(L, \gamma', M)\Big) \implies \mathsf{grec}(L, \gamma, M) < \mathsf{grec}(L, \gamma', M).$$

We argue that enlarging an event log should not influence guard-recall (Axiom 6).

Axiom 6 (Proposition 6 in [40]): Let L be a log, let $M = (N, F, n_r, N_f, \dots)$ be a model, let γ be a matching between L or L^k and M, and let $k \geq 1$ be an integer factor. Guard-recall cannot take on a different value if a log is enlarged:

$$\mathsf{grec}(L, \gamma, M) = \mathsf{grec}(L^k, \gamma, M).$$

Guard-Precision. Ideally, flows with guards only evaluate to true when they are observed to be taken. Hence, guard-precision quantifies if guards only evaluated to true when the associated flow is observed and taken. When comparing similar models with the same event log, guard-precision takes on a larger value for the model with fewer guard evaluations that were true (Axiom 7).

Axiom 7: Let L be a log, let $M = (N, F, n_r, N_f, \dots)$ be a model, let M' be another model with iso *an isomorphism between these models, let γ be a matching between L and M and let γ' be its isomorphic equivalent matching between L and M'. Guard-precision takes on a larger value for the model with fewer guard evaluations that were true when the flow was not taken*:

$$\sum_{f \in F}\big(\mathsf{Grd}_{\mathsf{T}}(f, \gamma, L) - \mathsf{Grd}_{\mathsf{T}}(\mathsf{iso}(f), \gamma', L)\big) > 0 \implies \mathsf{gprec}(L, \gamma, M) < \mathsf{gprec}(L, \gamma', M').$$

Next, guard-precision takes on the maximal value if and only if all guards only evaluated to true when they are observed and taken, and no control-flow issues exist in the matching function (Axiom 8).

Axiom 8: Let L be a log, let $M = (N, F, n_r, N_f, \dots)$ be a model, and let γ be a matching between them. Guard-precision takes on the maximal value if and only if, the matching is optimal and guards only evaluated to true when taken:

$$\big(\mathsf{optimal}(L, \gamma, M) \wedge \forall_{f \in F}[\mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma, L) = \mathsf{Grd}_{\mathsf{T}}(f, \gamma, L)]\big) \iff \mathsf{gprec}(L, \gamma, M) = \mathsf{max}.$$

Furthermore, we argue that enlarging an event log should not influence guard-precision (Axiom 9).

Axiom 9 (Proposition 11 in [40]): Let L be a log, let $M = (N, F, n_r, N_f, \dots)$ be a model, let γ be a matching between L or L^k and M, and let $k \geq 1$ be an integer factor. Guard-precision cannot take on a different value if a log is enlarged:

$$\mathsf{gprec}(L, \gamma, M) = \mathsf{gprec}(L^k, \gamma, M).$$

We should note that many things are implied by Axiom 8 beyond what was previously stated. Firstly, the following always holds for any flow f not part of a split in the model:

$$f \notin \mathsf{splits}(\mathrm{M}) \implies \mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma, \mathrm{L}) = \mathsf{Grd}_{\mathsf{T}}(f, \gamma, \mathrm{L}).$$

Hence, a model without a split may have perfect guard-precision as the function can only be affected by splits. Also implied by Axiom 8, is that guard-precision does not penalise a model-log combination with unobserved behaviour and unobserved guards, as the following holds for any flow $f \in F$:

$$\nexists_{\sigma \in \mathrm{L}, 1 \leq i \leq |\sigma|}[\gamma_{\hat{\sigma},i} = f] \implies \mathsf{Grd}_{\mathsf{T}}^{\rightarrow}(f, \gamma, \mathrm{L}) = \mathsf{Grd}_{\mathsf{T}}(f, \gamma, \mathrm{L}) = 0.$$

Hence, a model with unobserved flows for a given log may have perfect guard-precision. To quantify if the control-flow is imprecise, one should not use guard-precision as control-flow imprecision is outside the scope of guard-precision.

We present these axioms as the minimal set to consider data-aware quantification and do not claim them to account for all cases. For example, future work may consider if data-aware conformance checking should quantify models that are not relaxed data sound.

VII. PROPOSED MEASURES

In this section, we formally introduce a guard-recall measure and a guard-precision measure according to the extended theory in Section V.

A. GUARD-RECALL

We first introduce our guard-recall function grec. Guard-recall quantifies if guards have been formed over observed data attributes and if they evaluate to true when taken. Ideally, guard-recall will allow us to differentiate between data-aware models that have guards that evaluate to false or undefined more often than others with respect to an event log. Our formulation uses bookkeeping functions as the axioms do, but allows for one-to-many matchings (see Def. 24). Starting with a traversal bookkeeping function which uses one-to-many matchings to count guard evaluations that were true.

Definition 30 (Weighted Traversal Bookkeeping): Let L be a log, let M be a model, let L_M be the play-out log of M (Def. 18), let $M' = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be the play-out tree (Def. 20) with flow reduction (Def. 21), let Γ be a one-to-many matching between L and M' (Def. 24), let ω be a weighting function for paths (Def. 25), and let $f \in F$ be a flow. Then, $\mathsf{MGrd}_{\mathsf{T}}^{\rightarrow}$ is a function that yields the weighted

sum of evaluations that were true for a given flow:

$$\mathsf{MGrd}^{\rightarrow}_{\mathsf{T}}(f, \Gamma, L) = \sum_{\sigma \in L} \sum_{\rho \in \Gamma(\hat{\sigma})} \omega(\hat{\sigma}, \rho) \cdot \left[\exists_{1 \le i \le |\rho|} [\rho_i = f \wedge [\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\right].$$

Next, using weighted traversal bookkeeping, we now formally introduce our guard-recall measure.

Definition 31 (Guard-Recall): Let L be a log, let M be a model, let L_M be the play-out log of M (Def. 18), let $M' = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be the play-out tree (Def. 20) with flow reduction (Def. 21), let Γ be a one-to-many matching between L and M' (Def. 24), Then, we propose grec *is a function taking a log L, a matching Γ, and a tree M', defined as*:

$$\mathsf{grec}(L, \Gamma, M') = \frac{\sum_{f \in F} \mathsf{MGrd}^{\rightarrow}_{\mathsf{T}}(f, \Gamma, L)}{\sum_{\sigma \in L} \sum_{\rho \in \Gamma(\hat{\sigma})} \frac{1}{|\Gamma(\hat{\sigma})|} \cdot |\sigma|}.$$

Our definition for grec consists of a numerator which is the total weighted sum of guard evaluations that were true, accounting for whether they are optimal paths or not; while the denominator is the weighted sum of the flows recorded in the paths, where each flow is given the optimal share.

B. GUARD-PRECISION

We now introduce our guard-precision function gprec. Guard-precision quantifies whether guards have been formed over the observed data attributes and whether they are only evaluated to true when the associated flow is taken in paths. Ideally, guard-precision will allow us to differentiate between data-aware models that have guards that are more exclusive than another with respect to an event log. Like the previous formulation, our proposal for gprec is constructed using bookkeeping functions. As such, we introduce an exploratory bookkeeping function that uses a one-to-many matching to count all the possible guard evaluations that were true.

Definition 32 (Weighted Exploratory Bookkeeping): Let L be a log, let M be a model, let L_M be the play-out log of M (Def. 18), let $M' = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be the play-out tree (Def. 20) with flow reduction (Def. 21), let Γ be a one-to-many matching between L and M' (Def. 24), let $f \in F$ be a flow, then, $\mathsf{MGrd}_{\mathsf{T}}$ is a function that yields the total number of possible evaluations that were true for that flow:

$$\mathsf{MGrd}_{\mathsf{T}}(f, \Gamma, L) = \sum_{\sigma \in L} \sum_{\rho \in \Gamma(\hat{\sigma})} \frac{\left[\exists_{1 \le i \le |\rho|} [\mathsf{src}(f) = \mathsf{src}(\rho_i) \wedge [\![\mathsf{gd}(f)]\!](\mu_{\sigma,i-1})]\right]}{|\Gamma(\hat{\sigma})|}.$$

Next, using weighted exploratory bookkeeping, we formally introduce our guard-precision measure.

Definition 33 (Guard-Precision): Let L be a log, let M be a model, let L_M be the play-out log of M (Def. 18), let $M' = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be the play-out tree (Def. 20) with flow reduction (Def. 21), let Γ be a one-to-many matching between L and M' (Def. 24), and let $c \in \mathbb{R}$ be a constant factor s.t $c > 0$. Then, we propose gprec *is a function taking a log L, a matching Γ, and a tree M', defined as*:

$$\mathsf{gprec}(L, \Gamma, M') = \frac{c \cdot [\mathsf{optimal}(L, \gamma, M')] + \sum_{f \in F} \mathsf{MGrd}^{\rightarrow}_{\mathsf{T}}(f, \Gamma, L)}{c + \sum_{f \in F} \mathsf{MGrd}_{\mathsf{T}}(f, \Gamma, L)}.$$

Note that $\mathsf{MGrd}_{\mathsf{T}}(f, \Gamma, L)$ will always be at least as large as $\mathsf{MGrd}^{\rightarrow}_{\mathsf{T}}(f, \Gamma, L)$, so gprec will always be a ratio between zero and one. However, for the cases where for all $f \in F$: $\mathsf{MGrd}^{\rightarrow}_{\mathsf{T}}(f, \Gamma, L) = \mathsf{MGrd}_{\mathsf{T}}(f, \Gamma, L) = 0$, we include the constant c in Def. 33 so that the ratio is 1 in this case, to allow for adherence to Axiom 8, which requires guard-precision to return max.

VIII. EVALUATION

In this section, we evaluate adherence of existing measures [12], [19], [33], [35] and our two proposed measures (in Section VII), to the axioms proposed in Section VI. Our evaluation focuses not only on adherence to the axioms but also whether existing techniques encapsulate our notions of guard-recall or guard-precision. To achieve this, we adopt a 'counter-example' approach, as existing work has done [40], where measure by measure, axiom by axiom, we test for adherence. We determine that a measure does not adhere to the relevant axiom if even a single counter-example (drawn from exemplar logs and models) is found.

A. EXISTING TECHNIQUES

We selected, from existing literature, all known data-aware conformance checking techniques for which a working implementation exists. The existing techniques for a grec function are (i) grec_M using recall reported by [33], and (ii) grec_F using a weighted average of distances from equivalent trace classes reported by [19]. Then, the only existing technique for a gprec function is (iii) gprec_M using precision reported by [35]. For each of these techniques, we used the default cost-model for its underlying alignment technique. We formally introduce each technique in Appendix A, noting that for grec_F we propose a metric in a similar manner to the other techniques.

B. EXEMPLAR LOG AND MODEL

As in similar previous work [2], [40], [41], our evaluation makes use of counter-examples to test adherence to an axiom. Accordingly, we created a synthetic event log and a control-flow model for counter-examples. In terms of control-flow, the log and model are perfectly matching, to exclude control-flow issues. Our base model has one guard already introduced and this guard remains unchanged, i.e. the guard for activity 'a' is always true (see Fig. 8). This ensures that any constructed counter-example from this model could be relaxed data-sound [31, Def. 5.1], which requires at least one valid firing sequence from initial state to final state for the model.

Our exemplar event log L_E is shown below. It consists of nine variants and 18 sequences of assignments, and considers one attribute, d_1, with an integer domain [5 . . . 10]. However,

when testing for adherence to our axioms, we used an enlargement of this log, i.e. L_E^k with $k = 100$ (Def. 11).

$$\begin{aligned} L_E = \{&\langle a,b,e,f\rangle \mapsto [\langle \pi_1,\varnothing,\varnothing,\varnothing\rangle^3, \langle \pi_2,\varnothing,\varnothing,\varnothing\rangle^3],\\ &\langle a,b,e,g\rangle \mapsto [\langle \pi_1,\varnothing,\varnothing,\varnothing\rangle^2, \langle \pi_2,\varnothing,\varnothing,\varnothing\rangle^2],\\ &\langle a,b,e,h\rangle \mapsto [\langle \pi_1,\varnothing,\varnothing,\varnothing\rangle^1, \langle \pi_2,\varnothing,\varnothing,\varnothing\rangle^1],\\ &\langle a,c,e,f\rangle \mapsto [\langle \pi_3,\varnothing,\varnothing,\varnothing\rangle^3, \langle \pi_4,\varnothing,\varnothing,\varnothing\rangle^3],\\ &\langle a,c,e,g\rangle \mapsto [\langle \pi_3,\varnothing,\varnothing,\varnothing\rangle^2, \langle \pi_4,\varnothing,\varnothing,\varnothing\rangle^2],\\ &\langle a,c,e,h\rangle \mapsto [\langle \pi_3,\varnothing,\varnothing,\varnothing\rangle^1, \langle \pi_4,\varnothing,\varnothing,\varnothing\rangle^1],\\ &\langle a,d,e,f\rangle \mapsto [\langle \pi_5,\varnothing,\varnothing,\varnothing\rangle^3, \langle \pi_6,\varnothing,\varnothing,\varnothing\rangle^3],\\ &\langle a,d,e,g\rangle \mapsto [\langle \pi_5,\varnothing,\varnothing,\varnothing\rangle^2, \langle \pi_6,\varnothing,\varnothing,\varnothing\rangle^2],\\ &\langle a,d,e,h\rangle \mapsto [\langle \pi_5,\varnothing,\varnothing,\varnothing\rangle^1, \langle \pi_6,\varnothing,\varnothing,\varnothing\rangle^1]\}\\ &\textit{and } d_1 \in \mathcal{U}_{att} \textit{ with}:\\ \pi_1 = \{(d_1,5)\},\ &\pi_2 = \{(d_1,6)\},\ \pi_3 = \{(d_1,7)\},\\ \pi_4 = \{(d_1,8)\},\ &\pi_5 = \{(d_1,9)\},\ \pi_6 = \{(d_1,10)\}. \end{aligned}$$

C. COUNTER-EXAMPLES

In Fig. 8, we show three models used in our counter-examples. Fig. 8a, denotes a model with an ideal guard-precision annotation, but struggles with guard-recall as the guards are never enabled. Fig 8b, denotes a model with laxer guards in terms of guard-precision, but an improvement in guard-recall as some guards in the first choice can be enabled. Note, that for these two models, that activity 'e' has a guard that constricts the language of these models as d_1 must be written to be less than 5. Fig. 8c denotes a model which performs the best in terms of guard-recall, w.r.t. L_E^k, but also the worst in terms of guard-precision as the later choice always enables all options. Finally, all three models have an isomorphism up to control-flow with each other.

For most axioms, we constructed many counter-examples to show that a candidate does not adhere to an axiom. However, for Axioms 1 & 2, we provide straightforward reasoning to determine adherence as these relate to how candidates are formulated. For Axioms 6 & 9, we used any of the following matching γ, model M, log L_E^k combinations, and a further enlarged log L_E^K s.t. $k \ll K$ to test that for a measure m, $m(L_E^k, \gamma, M) = m(L_E^K, \gamma, M)$ holds.

For Axiom 3, we constructed possible counter-examples using: L_E^k, a matching γ and an isomorphic version γ', and any of the following pairs of models: (M:8a, M′:8b),(M:8a, M′:8c), or (M:8b, M′:8c). All of these pairs have been constructed to be such that they are strictly increasing in terms of (true) traversal bookkeeping from left to right. These counter-examples expect that for any measure that is a candidate for guard-recall, $\mathsf{grec}(L_E^k, \gamma, M) < \mathsf{grec}(L_E^k, \gamma', M')$ should hold, otherwise the candidate does not adhere.

The potential counter-examples for Axiom 4 used: L_E^k, a matching γ, and a modified model M of Fig. 8a, where the guard for 'a' was changed, preventing it from being evaluated to true w.r.t L_E^k. Such a guard could be $d_1 < 5$, $(d_1 \neq \bot \wedge d_1 = \bot)$ or simply false. The expectation was that for any candidate of guard-recall, $\mathsf{grec}(L_E^k, \gamma, M) = \mathsf{min}$ should hold, otherwise the candidate does not adhere.

Next, for Axiom 5 we used: L_E^k, a model M (Fig. 8b), an optimal matching γ s.t. $\mathsf{optimal}(L_E^k, \gamma, M)$, and a non-optimal γ' to construct possible counter-examples. To construct γ', we ensured all paths ended with the same flow (either 'f','g', or 'h'), thus for some traces the associated path would be optimal but not for all. Also, by using Fig. 8b and only changing the last flow in paths, the (true) traversal bookkeeping will be the same between matchings. These counter-examples expect that for any candidate for guard-recall, $\mathsf{grec}(L, \gamma, M) < \mathsf{grec}(L, \gamma', M)$ should hold, otherwise the candidate does not adhere.

Changing to guard-precision, the potential counter-examples for Axiom 7 used: L_E^k, a matching γ and an isomorphic version γ', and any of the following pairs of models: (M:8c, M′:8b), (M:8c, M′:8a), or (M:8b, M′:8a). All of these pairs have been constructed to be such that they are strictly decreasing in terms of (true) exploratory bookkeeping from left to right. The expectation of these was that for any candidate of guard-precision, $\mathsf{gprec}(L_E^k, \gamma, M) < \mathsf{gprec}(L_E^k, \gamma', M')$ should hold, otherwise the candidate does not adhere.

Next, for Axiom 8 we used: L_E^k, a model M (Fig. 8a), an optimal matching γ s.t. $\mathsf{optimal}(L_E^k, \gamma, M)$, and a non-optimal γ' to construct possible counter-examples. The expectation of this counter-example was that for any candidate of guard-precision, both $\mathsf{gprec}(L_E^k, \gamma, M) = \mathsf{max}$ and $\mathsf{gprec}(L_E^k, \gamma', M) < \mathsf{max}$ should hold, otherwise the candidate did not adhere. The usage of γ' was to ensure that the measure did indeed account for a less-than-optimal matching before returning max.

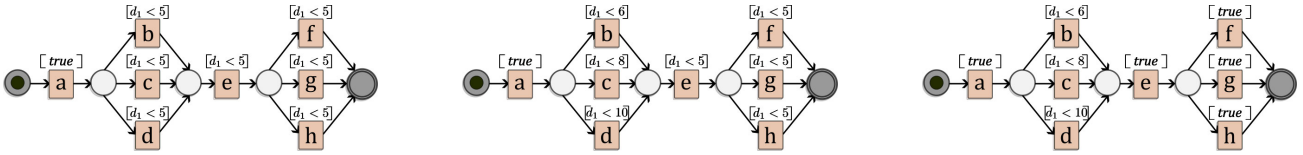
D. IMPLEMENTATION

We have implemented our proposed measures using Python. Our implementation takes an event log in the IEEE XES format and any DPN using the PNML format. Our implementation could be easily extended to consider any other executable data-aware process model, so long as a play-out log is derivable. Given that existing decision mining techniques only produce DPNs, we do not see this as a major limitation and it can be extended in future work. Our evaluation of existing techniques and our proposed measures, including all counter-examples, can be found here.[1] In this repository, we described the detailed steps taken for the evaluation to ensure the reproducibility of our results.

E. RESULTS

For each axiom, we investigated if a counter-example existed to show that a candidate does not adhere to that axiom (which can be found here[2]). If no negative counter-example was found, we include a proof sketch for adherence in these

[1] github.com/AdamBanham/data-aware-conformance
[2] github.com/AdamBanham/data-aware-conformance/tree/main/axioms



(a) $\text{grec}_M = 0.952$, $\text{grec}_F = -1$, $\text{gprec}_M \in \{0.908, 0.902, 0.901, 0.894, 0.847\}$, grec = 0.250, and gprec = 1.000.

(b) $\text{grec}_M = 0.952$, $\text{grec}_F = -1$, $\text{gprec}_M \in \{0.916, 0.908, 0.901, 0.900, 0.895, 0.894\}$, grec = 0.378, and gprec = 0.602.

(c) $\text{grec}_M = 0.977$, $\text{grec}_F = -0.488$, $\text{gprec}_M \in \{0.827, 0.825, 0.803, 0.796\}$, grec = 0.878, and gprec = 0.539.

FIGURE 8. A series of counterexamples using one log, where true guard evaluations are monotonically increasing from (a) to (c) with respect to L_E^k, where the number of compliant events in traces for models are: (a) 1, (b) 2, (c) 4. Measurements are denoted in sub captions. The variable d_1 has an integer domain of [5 . . . 10] and is written by 'a'.

TABLE 2. Counter-example testing across our axioms.

Axiom	grec_M	grec_F	grec	gprec_M	gprec
A1/A2	×	×	√	×	√
A3	×	×	√	-	-
A4	×	×	√	-	-
A5	-*	-*	√	-	-
A6/A9	√	√	√	×*	√
A7	-	-	-	×*	√
A8	-	-	-	×*	√

cases in Appendix B. We summarise our testing in Table 2, where '√' denotes adherence to an axiom, '×' denotes disregard for an axiom, '-' denotes irrelevance, and '*' denotes impediments during testing. Impediments could be caused by tightly coupled concepts or by the requirements of a candidate. Our proposed measures from Section VII grec (Def. 31) and gprec (Def. 33), are highlighted in Table 2.

We note that no existing grec technique adheres to Axiom 1, i.e., that guard representation should not matter, only behaviour does. That is, these techniques rely on guards being in a specific form (representation) so they can perform their optimisations. Also, we reason that gprec_M does not adhere to Axiom 2, as gprec_M requires traces that exactly match the model with no deviations in either control-flow or data perspective. Therefore, the implementation requires creating a log of fitting traces that fulfil guard evaluations and write constraints of a DPN [16], [31], and the resulting log will only contain assignments that evaluate guards to be true.

For the remaining axioms related to guard-recall, the following was observed for existing grec techniques. Consider Fig. 8, from (a) to (c), where the number of guard evaluations being true increases monotonically. As all candidates return the same value for Fig. 8a & 8b, they do not adhere to Axiom 3, which states that there should be a difference. All guard-recall candidates require a relaxed data sound model [19], [31], hence they cannot adhere to Axiom 4, as if no guard can be true when taken, the model cannot be relaxed data sound. Axiom 5 focuses on dissimilar matchings and as all existing guard-recall candidates do not encounter dissimilar matchings (i.e. they always use the "optimal" alignment), the axiom does not apply. Finally, given a log or an enlargement for any counter-example, candidates return the same value, showing adherence to Axiom 6.

As gprec_M requires compliant traces and uses [33] to generate compliant assignments, and these may take on a range of possible values, means that an implementation of gprec_M has the potential to be non-deterministic. In our testing, we noted that [33] does not consistently return the same assignments, this induces non-deterministic behaviour in gprec_M [35] for a given log and model, i.e gprec_M returned several values over ten unique runs shown in Fig. 8. The resulting behaviour from this requirement impedes gprec_M from adhering to any axiom, but with a deterministic implementation, it would theoretically adhere to many.

Now, consider Fig. 8, from (c) to (a), where the total number of guards evaluating to true is monotonically decreasing. Then, gprec_M does not return strictly increasing measurements across these models. Thus, it does not adhere to Axiom 7. Next, gprec_M does not adhere to Axiom 8, as it could not return max for Fig. 8a. Finally, given a log or enlargement for any counter-example, gprec_M does not return the same value, and thus does not adhere to Axiom 9.

Thus, our evaluation shows that the existing candidates for either guard-recall or guard-precision, do not adhere to all related axioms. In comparison to existing techniques, our proposed measures (grec or gprec) can cater for guards of any representation, as each uses bookkeeping, and our theory for guards does not optimise over the form of a guard. Hence, our proposed grec and gprec, adhere to Axioms 1 & 2.

As for Axiom 3, we consider Fig. 8, from (a) to (c), where the number of guard evaluations being true increases monotonically. In our case, grec produces measurements that strictly increase between these models, showing adherence to Axiom 3. Next, for Axiom 4, our proposed grec uses bookkeeping based on guards evaluating to true and returns zero if and only if no guards across all paths evaluate to true. We constructed several counter-examples with differing guard representations for this axiom and in each case grec returned zero (min) showing adherence to Axiom 4.

Then, for Axiom 5, we constructed counter-examples using Fig. 8b and L_E^k, where we compared using the optimal matching (Def. 29) and a strictly worst matching (e.g. all variants were mapped a path that lead to 'f'), thus the

explorative bookkeeping would either be the same or less as the model describes that no guard of latter choice can evaluate to true w.r.t L_E^k. In these cases, grec returned a larger value when using the optimal matching over non-optimal matching, showing adherence to Axiom 5. Furthermore, if we enlarge a log and compare it with the measurement using the original log, for any counter-example, grec and gprec returns the same measurement for the enlarged or original log, showing adherence to Axioms 6 & 9.

For our proposed gprec function, we consider Fig. 8, from (c) to (a), where the number of guard evaluations being true is monotonically decreasing. In such a case, gprec returns measurements that are strictly increasing across these models, showing adherence to Axiom 7. To consider adherence to Axiom 8, we considered Fig. 8a and various other guard representations with the same behaviour in terms of guard evaluations. However, we also created a counter-example model which only contained control-flow behaviour up to 'e' in Fig. 8a, to test that gprec does not return max as Axiom 8 describes. In all counter-examples for this axiom, our proposed gprec showed adherence by either returning max when expected or not max when expected.

Thus, existing techniques, or their implementations, do not adhere to many of our axioms for guard-recall or guard-precision. However, comparing against our proposed measures, our measures have been shown to outperform existing techniques in terms of adherence to the proposed axioms. Furthermore, the biggest difference can be seen between our gprec and the existing gprec_M, where the existing measure does not adhere to Axiom 9, but ours does. This finding means it is challenging for existing techniques to return a reliable measurement to compare decision mining techniques in terms of guard-precision. But, this is not the case for our technique gprec.

IX. CONCLUSION

In this article, we were motivated by a need to compare decision mining outcomes and were concerned that existing conformance checking may run into difficulties when quantifying an outcome. To address these concerns, we introduced the notion of guard-recall and guard-precision in terms of data-aware conformance checking through a novel quantification theory that is model agnostic for executable data-aware models as seen in Fig. 2. Next, we formalised these notions and outlined desirable properties for data-aware conformance checking as 9 axioms (4 related to work from [40]) which guard-recall and guard-precision functions should satisfy. Then, we tested existing techniques [19], [33], [35] against counter-examples for our proposed axioms and showed no existing candidates adhere entirely to these axioms. Hence, we saw a need to develop new measures for our notions of guard-recall and guard-precision, and have shown that our implementation adheres to our axioms. Our contribution of comparable conformance checking in terms of recall and precision, will allow future work to clearly quantify the differences between the efficiency of decision mining techniques.

There exist many avenues for future work, including investigating the possible existence of decision mining techniques that guarantee perfect guard-recall or guard-precision, or even both. Consideration of data-aware conformance checking could be extended to include generalisation and simplicity. A further area of work is investigating whether changing the semantics of guards to be policies, as seen in Fig. 1, affects the desirable properties of conformance checking. Finally, the sensitivity of our axioms to differences in data quality and process model quality should be investigated in a manner similar to [21].

APPENDIX A
EXISTING MEASURES

1) GUARD-RECALL TECHNIQUES

The first existing technique in the literature, proposed by de Leoni and van der Aalst [12], extends the classical alignment by using ILP to compute a data assignment matching the initial control-flow alignment. However, the original implementation of this technique is no longer available, thus we only consider the subsequent continuation of this work in [33] for our evaluation, which ensures that an optimal alignment is always produced.

The first technique considered in our evaluation, (grec_M), was proposed by Mannhardt et al. [33], and uses MILP to solve for an holistic alignment between a given trace and DPN. The approach applies a cost function ($\mathcal{K}$) which accounts for misalignments in the data-perspective, as well as the control-flow in a fully customisable way (i.e. the weights can be changed to penalise any one perspective more or less than the others). The formal computation of grec_M is noted below:

Definition 34 (grec_M as proposed by [33]): Let L be a log, let M be a model, let P be the set of paths through M, let $\gamma_O : L \mapsto P$ be the computed optimal alignments using [33], let $\gamma_R : L \mapsto P$ be the worst reference alignments according to [33], and let $\mathcal{K} : (L \times P) \mapsto \mathbb{R}$ be the cost function with respect to a given cost-model. Hence, grec_M is defined as:

$$\text{grec}_M(L, \gamma, M) = \frac{1}{|L|} \cdot \sum_{\sigma \in L} 1 - \frac{\mathcal{K}(\sigma, \gamma_O(\sigma))}{\mathcal{K}(\sigma, \gamma_R(\sigma))}.$$

Note that for our testing, we used the default cost-model for this technique where a log-move costs 1, a model-move costs 2, and any data-move costs 3.

The second candidate, (grec_F), uses the theory proposed in Felli et al. [19] to formulate a potential recall measurement. In [19], the CoCoMoT framework is used to compute multi-alignments in the a similar vein to [12] and [33]. However, their work focuses exclusively on an alignment problem and, unlike the already described techniques [12], [33], does not present a fitness/recall measure. Thus, we propose a metric using their theory for our evaluation.

To describe grec_F (from [19]), we take advantage of the standard cost function to convert an outcome from this

technique into a numeric. However, unlike other already described techniques, we report the weighted average of edits needed without a normalisation factor. Hence, grec_F will range between $[-\infty, 0]$, where 0 means that on average no edits were required to make traces compliant with the given DPN. The formal computation of a recall value derivable from [19] is noted below:

Definition 35 (Proposed grec_F using [19]): Let L be a log, let M be a model, let P be the set of paths through M, let $C \subset L$ be the set of trace classes computed by [19], let $\mathsf{rep} : L \mapsto C$ be the representative selection function for each class, let $\gamma_O : C \mapsto P$ be the optimal alignments for each class computed by [19], and let $\mathcal{K} : (C \times P) \mapsto [0, \infty]$ be the standard cost function in [19]. Then, we propose grec_F is defined as:

$$\mathsf{grec}_F(L, \gamma, M) = \frac{1}{|L|} \cdot \sum_{\sigma \in C} -1 \cdot |\{\sigma' \in L \mid \mathsf{rep}(\sigma') = \sigma\}| \cdot \mathcal{K}(\sigma, \gamma_O(\sigma))).$$

2) GUARD-PRECISION TECHNIQUE

We now introduce the only existing technique for guard-precision, gprec_M, proposed in Mannhardt et al. [35]. This technique computes a precision measurement by considering the 'compliant' version of the log, and computes a ratio between the observed and the possible behaviour.

To formally denote gprec_M, we need to introduce the notions of *possible* behaviour and *observed* behaviour. Firstly, we formally denote the possible behaviour for a given point in the model, as the set of possible next steps based on the assignments seen at that point.

Definition 36 (Possible Behaviour): Let L be a log, let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be a model, and $\hat{L} \subset (F \times \Pi)$ be the transformed compliant log with respect to M computed by [33]. Then, $\mathsf{pos} : F \mapsto \mathcal{P}(F \times \Pi)$ is a function yielding the set of possible behaviour, defined as:

$$\mathsf{pos}(f) = \left\{(f', \pi) \mid (f, \pi) \in \hat{L} \wedge f' \in \mathsf{src}(f)\bullet \wedge [\![\mathsf{gd}(f')]\!](\pi)\right\}.$$

Now, we formally denote the observed behaviour for a given point in the model, as the set of recorded next steps based on the assignments seen at that point.

Definition 37 (Observed Behaviour): Let L be a log, let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be a model, and $\hat{L} \subset (F \times \Pi)$ be the transformed compliant log with respect to M computed by [33]. Then, $\mathsf{obs} : F \mapsto \mathcal{P}(F \times \Pi)$ is a function yielding the set of observed behaviour, defined as:

$$\mathsf{obs}(f) = \left\{(f, \pi) \in \hat{L} \mid [\![\mathsf{gd}(f)]\!](\pi)\right\}.$$

Finally, we formally denote the candidate gprec_M as proposed by Mannhardt et al. [35] below:

Definition 38 (gprec_M as proposed by [35]): Let L be a log, let $M = (N, F, n_r, N_f, \mathsf{src}, \mathsf{tgt}, \mathsf{gd}, \mathsf{actf})$ be a model, and $\hat{L} \subset (F \times \Pi)$ be the transformed compliant log with respect to M computed by [33]. Hence, gprec_M is defined as:

$$\mathsf{gprec}_M(L, \gamma, M) = \frac{\sum_{f \in F} |\mathsf{obs}(f)|}{\sum_{f \in F} |\mathsf{pos}(f)|}.$$

APPENDIX B
PROOF SKETCHES

Axiom 3 states that guard-recall increases proportionally to the (true) traversal bookkeeping. Thus, between two isomorphic models, one log, and one matching, the model with larger (true) traversal bookkeeping yields a larger guard-recall.

For our proposed measure grec*, Axiom 3 holds.*

Reasoning. Based on Def. 31, our proposed measure yields a measurement based on weighted traversal bookkeeping function (Def. 30) in the numerator of the proposed ratio. Given that both of the log and matching are fixed, the bookkeeping function will yield a number that is proportional to the number of truthful guard evaluations for a given model. Hence, the model with larger (true) traversal bookkeeping will always yield a larger guard-recall.

Axiom 4 states that guard-recall only returns min when the (true) traversal bookkeeping is zero for a given model, log and matching.

For our proposed measure grec*, Axiom 4 holds.*

Reasoning. Based on Def. 31, our proposed measure will only return zero (min) when the weighted traversal bookkeeping function (Def. 30) returns zero. This bookkeeping only returns zero when no guard evaluations were true, in line with the axiom. Hence, our proposed measure only yields min when there were no guard evaluations that were true.

Axiom 5 states that guard-recall is negatively affected by the paths in a matching being non-optimal, in the context of a log, a model, an optimal and non-optimal matching.

For our proposed measure grec*, Axiom 5 holds.*

Reasoning. Based on Def. 31, our proposed measure is proportional to the weighted traversal bookkeeping (Def. 30). This bookkeeping is a weighted sum over guard evaluations, where the weight is based on the path in the matching being optimal. Optimal paths are given larger weights than non-optimal paths through the weight function (Def. 25). If the (true) traversal bookkeping between matchings is the same, then our proposed measure grec is larger for the matching that is optimal as the bookkeeping will assign more weight to the optimal case. If the non-optimal matching has strictly less guard evaluations that were true, then the weighted sum from the bookkeeping will be less than the optimal matching.

Axiom 6 states that guard-recall is not influenced by log enlargements.

For our proposed measure grec*, Axiom 6 holds.*

Reasoning. Based on Def. 31, our proposed measure is a ratio where the numerator and denominator are iterations over the traces in a log. As such, enlarging a log by cloning (Def. 11) will increase both numerator and denominator by a factor of k as the clones result in the same outcomes being counted k times. This factor k cancels out and the same ratio is returned regardless of the size of the factor.

Axiom 7 states that guard-precision is inversely proportional to (true) explorative bookkeeping (Eq. 8 in Def. 27).

Thus, guard-precision will be larger for a model that has a smaller sum over the explorative bookkeeping than another, for a given log and a matching.

For our proposed measure gprec*, Axiom 7 holds.*

Reasoning. Based on Def. 33, our proposed measure uses two bookkeeping functions, a weighted traversal function for the numerator and a weighted explorative function (Def. 32) for the denominator. The weighted explorative bookkeeping is proportional to the (true) explorative bookkeeping, thus if the explorative bookkeeping is larger, the denominator of our proposed ratio for gprec will be larger. As the denominator will always be at least as large as the numerator, our proposed ratio will be inversely proportional to the (true) explorative bookkeeping. Thus, a model will have a larger gprec than another, if the model has at least one flow where the explorative bookkeeping was smaller than its isomorphic reflection in the other model, reducing the size of the denominator in our proposed measure.

Axiom 8 states that guard-precision only returns max when both traversal and explorative bookkeeping are the same and the matching is optimal, w.r.t. to a log and a model.

For our proposed measure gprec*, Axiom 8 holds.*

Reasoning. Based on Def. 33, our proposed measure is a ratio between the weighted traversal bookkeeping (Def. 30) and the weighted explorative bookkeeping (Def. 32). As such this ratio can only be 1 (max), when both bookkeeping functions return the same value. The bookkeeping in the numerator can only return the same value as the bookkeeping in the denominator when the matching is optimal, as the denominator assigns weight to each path as if they were optimal. Hence, our proposed measure only returns max when the results of both forms of bookkeeping are the same, or if guards only evaluated to true when they were taken in paths within the matching.

Axiom 9 states that guard-precision is not influenced by log enlargements.

For our proposed measure gprec*, Axiom 9 holds.*

Reasoning. Based on Def. 33, our proposed measure is a ratio where the numerator and denominator are iterations over the traces in a log. As such, enlarging a log by cloning (Def. 11) will increase both numerator and denominator by a factor of k as the clones result in the same outcomes being counted k times. This factor k cancels out and the same ratio is returned regardless of the size of the factor.

ACKNOWLEDGMENT

The authors would like to thank Adam Burke, who took the time for a handful of "five" minute conversations to discuss formulation in earlier versions, which surely took longer.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining—Data Science in Action*. Berlin, Germany: Springer, 2016.

[2] W. M. P. van der Aalst, "Relating process models and event logs—21 conformance propositions," in *Proc. Int. Workshop Algorithms Theories Anal. Event Data Satell. Event Conf., 39th Int. Conf. Appl. Theory Petri Nets Concurrency Petri Nets, 18th Int. Conf. Appl. Concurrency Syst. Design*, vol. 2115, W. M. P. van der Aalst, R. Bergenthum, and J. Carmona, Eds. Bratislava, Slovakia, 2018, pp. 56–74. [Online]. Available: https://ceur-ws.org/Vol-2115/ATAED2018-56-74.pdf

[3] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[4] A. Adriansyah, "Aligning observed and modeled behavior," Ph.D. dissertation, Math. Comput. Sci., Eindhoven, The Netherlands, 2014.

[5] H. Alkhammash, A. Polyvyanyy, A. Moffat, and L. García-Bañuelos, "Entropic relevance: A mechanism for measuring stochastic process models discovered from event data," *Inf. Syst.*, vol. 107, Jul. 2022, Art. no. 101922.

[6] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, and M. L. Rosa, "Measuring fitness and precision of automatically discovered process models: A principled and scalable approach," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1870–1888, Apr. 2022.

[7] A. Banham, S. J. J. Leemans, M. T. Wynn, R. Andrews, K. B. Laupland, and L. Shinners, "XPM: Enhancing exogenous data visibility," *Artif. Intell. Med.*, vol. 133, Nov. 2022, Art. no. 102409.

[8] A. Bolt, M. de Leoni, and W. M. P. van der Aalst, "Process variant comparison: Using event logs to detect differences in behavior and business rules," *Inf. Syst.*, vol. 74, pp. 53–66, May 2018.

[9] J. vom Brocke, W. M. van der Aalst, T. Grisold, W. Kremser, J. Mendling, B. Pentland, J. Recker, M. Roeglinger, M. Rosemann, and B. Weber, "Process science: The interdisciplinary study of continuous change," *SSRN Electron. J.*, 2021.

[10] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity," *Int. J. Cooperat. Inf. Syst.*, vol. 23, no. 1, Mar. 2014, Art. no. 1440001.

[11] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking—Relating Processes Models*. Berlin, Germany: Springer, 2018.

[12] M. de Leoni and W. M. P. van der Aalst, "Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming," in *Business Process Managemen* (Lecture Notes in Computer Science), vol. 8094, F. Daniel, J. Wang, and B. Weber, Eds., Beijing, China. Berlin, Germany: Springer, 2013, pp. 113–129.

[13] B. F. van Dongen, "Efficiently computing alignments—Using the extended marking equation," in *Business Process Managemen* (Lecture Notes in Computer Science), vol. 11080, M. Weske, M. Montali, I. Weber, and J. vom Brocke, Eds., Sydney, NSW, Australia. Cham, Switzerland: Springer, 2018, pp. 197–214.

[14] D. Duma and R. Aringhieri, "Mining the patient flow through an emergency department to deal with overcrowding," in *Health Care Systems Engineering*, P. Cappanera, J. Li, A. Matta, E. Sahin, N. J. Vandaele, and F. Visintin, Eds. Cham, Switzerland: Springer, 2017, pp. 49–59.

[15] P. Felli, M. de Leoni, and M. Montali, "Soundness verification of data-aware process models with variable-to-variable conditions," *Fundamenta Informaticae*, vol. 182, no. 1, pp. 1–29, Sep. 2021.

[16] P. Felli, M. de Leoni, and M. Montali, "Soundness verification of decision-aware process models with variable-to-variable conditions," in *Proc. 19th Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, Aachen, Germany, Jun. 2019, pp. 82–91.

[17] P. Felli, M. Montali, and S. Winkler, "Repairing soundness properties in data-aware processes," in *Proc. ICPM*, 2023, pp. 41–48.

[18] P. Felli, M. Montali, and S. Winkler, "Soundness of data-aware processes with arithmetic conditions," in *Advanced Information Systems Engineering* (Lecture Notes in Computer Science), vol. 13295. Cham, Switzerland: Springer, 2022, pp. 389–406.

[19] P. Felli, A. Gianola, M. Montali, A. Rivkin, and S. Winkler, "Data-aware conformance checking with SMT," *Inf. Syst.*, vol. 117, Jul. 2023, Art. no. 102230.

[20] J. Hidders, M. Dumas, W. M. P. van der Aalst, A. H. M. T. Hofstede, and J. Verelst, "When are two workflows the same?" in *Proc. 11th Comput. Australian Theory Symp.*, vol. 41, M. D. Atkinson and F. K. H. A. Dehne, Eds. Newcastle, NSW, Australia, 2005, pp. 3–11. [Online]. Available: http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV41Hidders.html

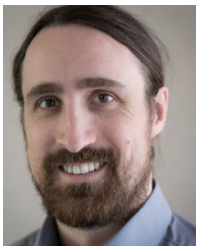
[21] G. Janssenswillen, N. Donders, T. Jouck, and B. Depaire, "A comparative study of existing quality measures for process discovery," *Inf. Syst.*, vol. 71, pp. 1–15, Nov. 2017.
[22] T. Jouck, M. de Leoni, and B. Depaire, "A framework to evaluate and compare decision-mining techniques," in *Business Process Managemen* (Lecture Notes in Computer Science), vol. 342, F. Daniel, Q. Z. Sheng, and H. Motahari, Eds., Sydney, NSW, Australia. Cham, Switzerland: Springer, 2018, pp. 482–493.
[23] S. J. J. Leemans, *Robust Process Mining With Guarantees—Process Discovery, Conformance Checking Enhancement* (Lecture Notes in Business Information Processing), vol. 440. Cham, Switzerland: Springer, 2022.
[24] S. J. J. Leemans, W. M. P. van der Aalst, T. Brockhoff, and A. Polyvyanyy, "Stochastic process mining: Earth movers' stochastic conformance," *Inf. Syst.*, vol. 102, Dec. 2021, Art. no. 101724.
[25] S. Leewis, K. Smit, and M. Zoet, "Putting decision mining into context: A literature study," in *Digital Business Transformation*, R. Agrifoglio, R. Lamboglia, D. Mancini, and F. Ricciardi, Eds. Cham, Switzerland: Springer, 2020, pp. 31–46.
[26] M. de Leoni and W. M. P. van der Aalst, "Data-aware process mining: Discovering decisions in processes using alignments," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, S. Y. Shin and J. C. Maldonado, Eds. Coimbra, Portugal, 2013, pp. 1454–1461.
[27] M. de Leoni, M. Dumas, and L. Garcia-Banuelos, "Discovering branching conditions from business process execution logs," in *Fundamental Approaches to Software Engineering* (Lecture Notes in Computer Science), vol. 7793, V. Cortellessa and D. Varro, Eds., Rome, Italy. Berlin, Germany: Springer, 2013, pp. 114–129.
[28] M. de Leoni, P. Felli, and M. Montali, "A holistic approach for soundness verification of decision-aware process models," in *Conceptual Modeling* (Lecture Notes in Computer Science), vol. 11157, J. Trujillo, K. C. Davis, X. Du, Z. Li, T. W. Ling, G. Li, and M. Lee, Eds., Xi'an, China. Cham, Switzerland: Springer, 2018, pp. 219–235.
[29] M. de Leoni and F. Mannhardt, "Decision discovery in business processes," in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya, Eds. Cham, Switzerland: Springer, 2019.
[30] S. Lindell, "A logspace algorithm for tree canonization (extended abstract)," in *Proc. 24th Annu. ACM Symp. Theory Comput.*, S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, Eds. Victoria, BC, Canada, 1992, pp. 400–404.
[31] F. Mannhardt, "Multi-perspective process mining," Ph.D. dissertation, Math. Comput. Sci., Eindhoven, The Netherlands, Feb. 2018.
[32] F. Mannhardt and D. Blinde, "Analyzing the trajectories of patients with sepsis using process mining," in *Proc. 18th Int. Work. Conf. Bus. Process Model., Develop. Support (BPMDS), 22nd Int. Work. Conf. Eval. Model. Methods Syst. Anal. Develop. (EMMSAD), 8th Int. Workshop Enterprise Model. Inf. Syst. Architectures (EMISA), 29th Int. Conf. Adv. Inf. Syst. Eng.*, vol. 1859, Essen, Germany, 2017, pp. 72–80. [Online]. Available: https://ceur-ws.org/Vol-1859/bpmds-08-paper.pdf
[33] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Balanced multi-perspective checking of process conformance," *Computing*, vol. 98, no. 4, pp. 407–437, Apr. 2016.
[34] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Decision mining revisited—Discovering overlapping rules," in *Advanced Information Systems Engineering* (Lecture Notes in Computer Science), vol. 9694, S. Nurcan, P. Soffer, M. Bajec, and J. Eder, Eds., Ljubljana, Slovenia. Cham, Switzerland: Springer, 2016, pp. 377–392.
[35] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Measuring the precision of multi-perspective process models," in *Business Process Managemen* (Lecture Notes in Computer Science), vol. 256, M. Reichert and H. A. Reijers, Eds. Cham, Switzerland: Springer, 2015, pp. 113–125.
[36] R. De Masellis, C. Di Francescomarino, C. Ghidini, and S. Tessaris, "Solving reachability problems on data-aware workflows," *Exp. Syst. Appl.*, vol. 189, Mar. 2022, Art. no. 116059.
[37] A. Rozinat and W. M. P. van der Aalst, "Decision mining in prom," in *Business Process Managemen* (Lecture Notes in Computer Science), vol. 4102, S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds. Berlin, Germany: Springer, 2006, pp. 420–425.
[38] B. Scheibel and S. Rinderle-Ma, "Decision mining with time series data based on automatic feature generation," in *Advanced Information Systems Engineering* (Lecture Notes in Computer Science), vol. 13295, X. Franch, G. Poels, F. Gailly, and M. Snoeck, Eds., Leuven, Belgium. Cham, Switzerland: Springer, 2022, pp. 3–18.
[39] D. Sommers, N. Sidorova, and B. F. van Dongen, "Exact and approximated log alignments for processes with inter-case dependencies," in *Application and Theory of Petri Nets and Concurrency* (Lecture Notes in Computer Science), vol. 13929, L. Gomes and R. Lorenz, Eds., Lisbon, Portugal. Cham, Switzerland: Springer, 2023, pp. 99–119.
[40] A. F. Syring, N. Tax, and W. M. P. van der Aalst, "Evaluating conformance measures in process mining using conformance propositions," in *Transactions on Petri Nets and Other Models of Concurrency XIV*, vol. 14. Berlin, Germany: Springer, 2019, pp. 192–221.
[41] N. Tax, X. Lu, N. Sidorova, D. Fahland, and W. M. P. van der Aalst, "The imprecisions of precision measures in process mining," *Inf. Process. Lett.*, vol. 135, pp. 1–8, Jul. 2018.

ADAM BANHAM received the bachelor's degree (Hons.) in information technology from Queensland University of Technology (QUT), Brisbane, Australia, in 2020, where he is currently pursuing the Ph.D. degree in process mining, under the supervision of Prof. Sander J. J. Leemans, Dr. Robert Andrews, and Prof. Moe T. Wynn.

From 2017 to 2021, he was a Research Assistant with QUT across several data science/process mining projects which investigated ambulance triage, car insurance claims, and clinical populations across in QLD, Australia. His research interests include business process management, modeling formalisms for studying behavior in processes, and decision-making in business.

ARTHUR H. M. TER HOFSTEDE received the Ph.D. degree from Katholieke Universiteit Nijmegen (since renamed to Radboud Universiteit), Nijmegen, The Netherlands, in 1993. Since 1997, he has been with Queensland University of Technology, Brisbane, QLD, Australia, where he is currently a Professor and a Principal Research Fellow of the School of Information Systems. He was involved in the well-known workflow patterns initiative. At QUT, he has managed the well-known YAWL initiative. He is the coauthor on over 290 publications, including over 100 journal publications. His research interests include business process automation, process mining, and data quality.

SANDER J. J. LEEMANS received the Ph.D. degree from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2017. He is currently a Professor (W2) with Rheinisch-Westfälische Technische Hochschule University (RWTH), Aachen, Germany. His research interests include process mining, process discovery, conformance checking, stochastic process mining, and robotic process automation. In particular, he specializes in making solid academic techniques available to end-users, analysts, and industry partners. He teaches business process management, business process modeling, and business process improvement. He is also affiliated with the Fraunhofer FIT Institute.



FELIX MANNHARDT received the Ph.D. degree from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2018. He is currently an Assistant Professor (Universiair Docent) with Eindhoven University of Technology. His research interests include process mining in various settings and in conjunction with other data science and machine learning methods, the development and application of process mining methods related to low-level event data (event abstraction, activity recognition, sensor data, and multi-perspective event logs), related to conformance checking (combination of control-flow with rules over data), and related to trust and privacy concerns (privacy-preserving methods, privacy, and trust models). He has published more than 50 peer-reviewed publications and also contributed significantly to the leading open-source process mining frameworks, ProM and bupaR.

ROBERT ANDREWS received the Ph.D. degree from Queensland University of Technology, Brisbane, QLD, Australia, in 2003. He is currently a Senior Research Fellow and has worked on applying process mining techniques in the healthcare and insurance sectors. He is also working on projects involving pre-hospital transport and retrieval, patient journeys and outcomes following major trauma resulting from road traffic crashes, and process-data quality. His research interests include data quality, process mining, data mining, and machine learning.

MOE T. WYNN (Member, IEEE) received the Ph.D. degree in workflow management from Queensland University of Technology (QUT), Brisbane, QLD, Australia, in 2007. She currently leads the Business Process Management Research Group, QUT. She is also the Vice-Chair and one of the steering committee members of the IEEE Taskforce on Process Mining. She has published more than 80 refereed articles, including more than 30 journal articles. Her research interests include process-oriented data mining (process mining), data quality, and robotic process automation for the digital transformation of processes.