Measuring the interestingness of temporal logic behavioral specifications in process mining

Alessio Cecconi a,∗,1, Giuseppe De Giacomo b,1, Claudio Di Ciccio b,1, Fabrizio Maria Maggi c,1, Jan Mendling d,a,1

a Vienna University of Economics and Business, Vienna, Austria
b Sapienza University of Rome, Rome, Italy
c Free University of Bozen-Bolzano, Bolzano, Italy
d Humboldt-Universität zu Berlin, Berlin, Germany

A R T I C L E   I N F O

Article history:
Received 14 March 2021
Received in revised form 21 September 2021
Accepted 13 October 2021
Available online 1 November 2021
Recommended by Gottfried Bossen

Keywords:
Declarative process mining
Specification mining
Association rule mining
Interestingness measures
Temporal rules

A B S T R A C T

The assessment of behavioral rules with respect to a given dataset is key in several research areas, including declarative process mining, association rule mining, and specification mining. An assessment is required to check how well a set of discovered rules describes the input data, and to determine to what extent data complies with predefined rules. Particularly in declarative process mining, Support and Confidence are used most often, yet they are reportedly unable to provide a sufficiently rich feedback to users and cause rules representing coincidental behavior to be deemed as representative for the event logs. In addition, these measures are designed to work on a predefined set of rules, thus lacking generality and extensibility. In this paper, we address this research gap by developing a measurement framework for temporal rules based on (LTLp). The framework is suitable for any temporal rules expressed in a reactive form and for custom measures based on the probabilistic interpretation of such rules. We show that our framework can seamlessly adapt well-known measures of the association rule mining field to declarative process mining. Also, we test our software prototype implementing the framework on synthetic and real-world data, and investigate the properties characterizing those measures in the context of process analysis.

© 2021 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Measuring the degree to which process traces comply with behavioral rules is key in process analysis branches such as conformance checking [1], compliance assessment [2], and discovery of process constraints [3]. To date, several measures have been defined to this end, yet there are two major problems with their application.

First, measures adopted for process mining are defined inconsistently for specific applications. For example, among the most frequently used measures there are Support and Confidence. However, their definition has been customized to the specification languages in use and even for the specific mining algorithms under analysis. For instance, there is a significant difference in the definition of Support used in [3] (percentage of traces fully compliant to a rule) and [4] (percentage of activations that lead to a fulfillment), in a way that the Support of rule “If a is executed, then b will be executed later” on a set of traces like \{⟨a, b, c, d⟩, ⟨a, b, c, a⟩, ⟨a, c⟩\} is equal to 0.33 for [3] and 0.5 according to [4]. Furthermore, the definition of those measures are defined ad hoc for specific sets of rules, like DECLARE [5] templates. Such issues hinder the fair comparison and eventually the advancement of rule-based process mining.

Second, the opportunity to adopt available measures from association rule mining has been largely missed so far. A plethora of measures that are reportedly superior in comparison to Support and Confidence [6] have been proposed in this field. Support measures only the satisfaction frequency of a rule and Confidence its validity. Although those are crucial aspects in the assessment of a rule, they do not suffice to avoid spurious result [7]. Markedly, various directions have been explored in prior research, among others by revising Support and Confidence [8–10], and by defining complementary measures [11–13]. However, all such measures do not account for the temporal perspective, which is a first-class citizen dimension in process mining.

In this paper, we address the research challenge of defining a general and comprehensive measurement system. More
specifically, we propose a framework based on formal semantics grounded in Linear-time Temporal Logic with Past on Finite Traces (LTLp) to express Reactive Constraints (RCons) [14] in a way that abstracts from specific rule-specification languages. Such constraints are rules in the form of “if \( A \) then \( B \)”, thus binding the satisfaction of an antecedent \( A \) to the occurrence of a consequent \( B \), wherein both \( A \) and \( B \) are temporal formulas. We show that a probabilistic interpretation of the fine-grained temporal logic evaluation of any such formulas allows us to employ all available association rule mining measures as-is for temporal rules. Markedly, the framework has linear time and space complexity with respect to the input size.

Our contribution extends concepts from association rule mining to temporal logic specifications. In this way, we define a foundation upon which the fitness between measures and data analysis scenarios can be discussed in future research. We conduct an extensive set of simulation experiments, the results of which demonstrate that, driven by known properties, the measures respond differently to changes in the behavior evidenced by event logs. This is an important finding that highlights the need to select measures according to the application context, confirming previous findings for association rules [15] in the realm of temporal logic specifications.

This paper is an extension of our previous conference paper [16] presented at the 2nd International Conference on Process Mining (ICPM 2020). We extend the contribution in the following aspects:

- We extend the framework to provide measures at the level of the event log, and not only descriptive statistics at the trace level (Section 4);
- We revise the experiments based on the new theoretical extension. In particular, we score the proposed measures and rank them in order to identify the best candidates to be used for rule discovery (Section 6);
- We analyze the memory consumption of the framework along with the time performance (Section 5);
- We extend the discussion about the interestingness measures used and exploit their known properties for a better understanding of log behavior (Sections 3 and 6).

Additionally, we prove the linear-time performance of the RCons verification in Appendix.

The remainder of this paper is structured as follows. Section 2 discusses prior research on measures for declarative process mining and specification mining. Section 3 defines preliminaries upon which we define our framework. Section 4 defines the measurement framework. Section 5 presents a computational study of the framework and Section 6 shows the results of an array of simulation experiments and discusses them. Finally, Section 7 summarizes the contribution of the paper and points to opportunities for future research.

2. Related work

Behavioral rules have been widely used to support application scenarios such as association rule mining in machine learning, process discovery and conformance checking in process mining, and specification mining in software engineering. The assessment of rules with respect to the available data is a key component of all these techniques.

In association rule mining, interestingness measures are used to discriminate candidate pairs of relevant co-occurring events. A common technique is to discover frequent rules above a certain Support threshold (frequency) and to prune the results below a certain Confidence threshold (validity). For example, [17] discovers associations between items through an Apriori algorithm based on the downward-closure property of the Support measure. Nevertheless, the use of Support and Confidence alone is reportedly not sufficient to avoid a large number of spurious results [7], i.e., the discovery of rules which are frequently satisfied by the data although merely by chance, thus threatening their statistical validity. A plethora of new measures have been proposed in the literature to overcome the limits of using only Support and Confidence [11], yet the employment of Support and Confidence as the main interestingness measures remains widespread. The main goal driving the development of better measures is indeed the exclusion of spurious rules, so as to let the more crucial ones stand out. Several measures are directly improving on or refining the results of Support and Confidence (e.g., Lift [8] scales Confidence with the Support of the consequent of a rule), others combine different measures (e.g., Added Value [12] subtracts the Prevalence to the Confidence of a rule), and further ones show complementary information (e.g., Specificity [11] measures to what extent the absence of the consequent is related to the absence of the antecedent of a rule).

In declarative process discovery, interestingness measures are used to prune candidate rules based on user-defined thresholds. This pruning approach is used for DECLARE discovery in [3,4] and for DCR graphs discovery in [18]. These techniques are mainly based on Support and Confidence, which lead to the aforementioned limits [7]. In addition, the definitions of these metrics also differ depending on the techniques that use them. For instance, the Support measure presented in [3] does not correspond to the Support measure of [4], although both are expressly defined for the sole DECLARE constraints.

In the area of conformance checking, interestingness measures are used to check the degree of conformance of a rule with respect to an execution trace. In [19], Linear Temporal Logic (LTL) rules are checked against each trace in a given event log. This is highly generic as it supports any custom LTL formula, but it reports only binary results, i.e., whether a rule holds in a trace or not. In [20], Burattin et al. use measures like fulfillment ratio and violation ratio, based on the evaluation of the number of activations of a rule (intuitively, the occurrences of its antecedent) that lead to a fulfillment and the number of activations that lead to a violation in an event log. However, these metrics are specifically bound to the set of DECLARE rules, thus not providing a general measurement framework that can be applied to general type of rule.

In specification mining, interestingness measures are also used to prune candidate temporal specifications based on user-defined thresholds. Interestingly, specification mining and declarative process discovery are two largely overlapping concepts from distinct fields. Yang et al. [21] discover 2-value temporal patterns using a trace measure that quantifies partial satisfactions of a rule. Yet, the technique lacks generality as it is limited only to alternation patterns (similar to ALTERNATEResponse and ALTERNATEPrecedence in Table 1) and the adopted computation heuristics are tailored to the software domain. Le et al. [6] emphasize the limits of using only Support and Confidence measures and investigate properties of other measures reviewed in [11]. Their results demonstrate that there are several measures outperforming Support and Confidence, and that the combination of different measures yields better results. However, they limit their study to 2-value temporal patterns (specifically, Response and Precedence in Table 1). Furthermore, their computation of the probability for a temporal specification is based on a sliding window technique [22]: traces are read in chunks of the size of a given window, then the probability of a rule is the percentage of windows in which it is satisfied. They test the effect of different window sizes, showing that their results depend not only on the input rules and the data, but also on this parameter selection.
Lemieux et al. [23] extend specification mining to arbitrary LTL specifications (implicitly on finite traces) beyond 2-value templates. However, they resort to the sole Support and Confidence measures to prune uninteresting results, thus incurring in the already mentioned statistical limits [7].

The aforementioned shortcomings of quality measures are also discussed in the field of sequence mining [24] when dealing with discovering patterns (specifically subsequences) to classify sequential data. Egho et al. [25] highlight how measures like Confidence and Lift alone lead to unstable classification results of subsequences, proposing a probabilistic Bayesian-based measure to overcome such an instability and avoiding the requirement of setting thresholds for measures. It falls under the family of techniques based on the minimum description length principle, like [26], where an encoding scheme is used to discover a minimal set of subsequences which can reproduce the original data. Notably, subsequence interleaving patterns are only a subset of patterns expressible with LTL formulae. Works adopting behavioral rules for classification like [27], on the other hand, fall back to the sole employment of Support.

In summary, despite the discussion in different fields on measures and the problem of spurious relations, there is no technique that supports at the same time a comprehensive and extensible multi-measurements assessment of rules and its applicability on general temporal logic specifications.

3. Preliminaries

To develop our framework, we build on the sound foundations of LTLp, in this section, we introduce the fundamentals of LTLp formulae (Section 3.1) and interestingness measures for association rules (Section 3.2).

3.1. Linear-time Temporal Logic with Past on Finite Traces (LTLp)

As the formal foundations of our framework, we consider the rules specified in Linear Temporal Logic on Finite Traces (LTL) [28] as used in DECLARE [5,29], LTLp has the same syntax as LTL [30]. Its semantics is interpreted on finite traces (here abstracted as finite sequences of symbols), and thus take into account that business processes are assumed to eventually terminate [31]. DECLARE focuses on a set of specific LTLp formulae. Table 1 illustrates some of the most important rules for business process specifications in DECLARE.

LTLp is an extension of LTL supporting the expression of properties of the past (hence the “p” suffix) [14]. Well-formed Linear-time Temporal Logic with Past on Finite Traces (LTLp) formulae are built from an alphabet \( \Sigma \supseteq \{a\} \) of propositional symbols and are closed under the boolean connectives, the unary temporal operators \( \diamond \) (next) and \( \ominus \) (previous), and the binary temporal operators \( \mathbf{U} \) (Until) and \( \mathbf{S} \) (Since):

\[
\varphi := a | \neg \varphi | (\varphi_1 \wedge \varphi_2) | (\mathbf{U} \varphi_1 \varphi_2) | (\mathbf{S} \varphi_1 \varphi_2).
\]

From these basic operators, the following can be derived: Classical boolean abbreviations \( \text{True}, \text{False}, \lor, \rightarrow, \); Constant \( t_{\text{end}} \equiv \text{True} \), denoting the last instant of a trace; Constant \( t_{\text{start}} \equiv \text{True} \), denoting the first instant of a trace; \( \diamond \varphi \equiv \text{True} \) \( \mathbf{U} \varphi \) indicating that \( \varphi \) holds true eventually before \( t_{\text{end}} \); \( \mathbf{U} \varphi_1 \varphi_2 \equiv (\varphi_1 \mathbf{U} \varphi_2) \lor \varphi_1 \), which relaxes \( \mathbf{U} \) as \( \varphi_2 \) may never hold true; \( \varphi_2 \equiv \text{False} \mathbf{S} \varphi \) indicating that \( \varphi \) holds true eventually in the past, after \( t_{\text{start}} \); \( \varphi \equiv \ominus \varphi \equiv \ominus \ominus \varphi \) indicating that \( \varphi \) holds true from the current instant till \( t_{\text{end}} \); \( \varphi \equiv \ominus \ominus \ominus \varphi \) indicating that \( \varphi \) holds true from \( t_{\text{start}} \) to the current instant.

Given a finite trace \( t \) of length \( n \in \mathbb{N} \), an LTLp formula \( \varphi \) is satisfied in a given instant \( t \) (\( 1 \leq i \leq n \)) by induction of the following:

\[
t, i \equiv \text{True}; \quad t, i \equiv \text{False};
\]

\[
t, i \equiv \text{a} \; (t(i) \text{ is assigned with } a);
\]

\[
t, i \equiv \neg \varphi \text{ iff } t, i \not= \varphi;
\]

\[
t, i \equiv \varphi_1 \wedge \varphi_2 \text{ iff } t, i \equiv \varphi_1 \; \text{and } \; t, i \equiv \varphi_2;
\]

\[
t, i \equiv \varphi_1 \lor \varphi_2 \text{ iff } t, i \equiv \varphi_1 \; \text{or} \; t, i \equiv \varphi_2;
\]

\[
t, i \equiv \mathbf{U} \varphi_1 \varphi_2 \text{ iff } t, j \equiv \varphi_2 \text{ with } j \leq i \leq n, \; \text{and } \; t, k \equiv \varphi_1 \text{ for all } k \leq j < k \leq l;
\]

\[
t, i \equiv \mathbf{S} \varphi_1 \varphi_2 \text{ iff } t, j \equiv \varphi_2 \text{ with } 1 \leq j \leq i, \; \text{and } \; t, k \equiv \varphi_1 \text{ for all } k < j < k < l;
\]

A formula \( \varphi \) is satisfied by a trace \( t \), written \( t \equiv \varphi \), \( t \equiv \varphi \) if \( t \equiv \varphi \). One of the central properties of LTLp and LTL is that a deterministic finite state automaton (DFA) \( A_\varphi \) can be computed such that for every trace \( t \) we have \( t \equiv \varphi \) iff \( t \) is in the language recognized by \( A_\varphi \), as illustrated in [14,32,33].

Without loss of generality, in this paper we abstract traces as finite strings of symbols representing events. We assume that every event reports on the execution of exactly one task and LTLp formulae use as finite tasks as their propositional symbols — the so-called DECLARE assumption [32]. A trace extracted from the real-world Sepsis event log [34] is, e.g., \( \varphi_{\text{sepsis}} = (\text{ER Registration}, \text{ER Triage}, \text{Sepsis Triage}, \text{CRP}, \text{Lactic Acid}, \text{Lactate}, \text{IV Liquid}, \text{IV Antibiotics}). \)

Notice that this trace complies with the constraints that Mannhardt et al. identified as normative for the Sepsis treatment process [35], including the following ones: (i) \text{IN}{\text{T}(ER Registration)}, i.e., every trace begins with the registration at the emergency department; (ii) \text{AI}{\text{R(MostON)}(ER Triage)}, i.e., the triage in the emergency room occurs at most once in a process run; (iii) \text{RESP}{\textONSE}{\text{(ER Registration)}, \text{ER Sepsis Triage}}, i.e., the ER Triage procedure should be eventually followed by the Sepsis-specific triage, (iv) \text{PRECEDENCE}{\text{(ER Sepsis Triage), IV Antibiotics}}, i.e., the intravenous injection of antibiotics must be preceded by the ER Sepsis Triage procedure.

Table 17 contains an extended set of DECLARE constraints that a correct execution of the Sepsis treatment process should fulfill. For the sake of succinctness, we may use single-letter identifiers in place of full-length task names whenever suitable in the following examples.

An event log is a multi-set of traces, i.e., traces can recur multiple times in an event log. The cardinality of the event log is the sum of the multiplicities of its traces. Considering the Sepsis event log, the multiplicity of \( \varphi_{\text{sepsis}} \) is 13 (i.e., \( \varphi_{\text{sepsis}} \) occurs 13 times in the event log). Other exemplary traces of that log are \( \varphi_{\text{sepsis}} = (\text{ER Registration}, \text{ER Triage}, \text{Sepsis Triage}, \text{CRP}) \) (occurring 35 times) and \( \varphi_{\text{sepsis}} = (\text{ER Registration}, \text{ER Triage}, \text{ER Sepsis Triage}, \text{Leucocytes}, \text{CRP}) \) (with a multiplicity of 24). The cardinality of the event sub-log consisting of the traces above is thus 72. The cardinality of the whole Sepsis event log is 1050.

3.2. Interestingness measures for association rules

In this section, we revisit key findings of research on quality measures in association rule mining. More specifically, we build on prior research that surveys measures in the area of software engineering and association rule mining, namely [11,15,36,37].

These works are specifically suited as a foundation due to the wide coverage of measures and their comparative study of both formal and user-perceived measure properties. The rules under study are in the form “If A Then B”, where A is called the antecedent of the rule, and B its consequent. We refer to A and B as variables of the rule.

Specifically, we consider only probability-based objective measures, i.e., measures depending only on the data as opposed to those requiring user-provided parameters. Objective measures are based on the probabilities derived from the contingency table of the occurrences of the variables, as depicted in Table 2. Table 3 presents the list of measures covered in this study. In the following, we provide a brief description of each measure.
Table 1
Some DECLARE constraints expressed as RCons.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>LTL expression [28]</th>
<th>RCon</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTICIPATION(a)</td>
<td>o a</td>
<td>$I_{\text{Init}} \implies o a$</td>
</tr>
<tr>
<td>INIT(a)</td>
<td>a</td>
<td>$I_{\text{Init}} \implies a$</td>
</tr>
<tr>
<td>END(a)</td>
<td>o a</td>
<td>$I_{\text{End}} \implies o a$</td>
</tr>
<tr>
<td>AtMostOne(a)</td>
<td>$o(A \rightarrow \neg o a))$</td>
<td>$a \implies (\neg o a)$</td>
</tr>
<tr>
<td>RESPONSEDExistence(a, b)</td>
<td>$o a \rightarrow o b$</td>
<td>$o b \implies (o b \vee o b)$</td>
</tr>
<tr>
<td>Response(a, b)</td>
<td>$o a \rightarrow o b$</td>
<td>$o b \implies o b$</td>
</tr>
<tr>
<td>AlternateResponse(a, b)</td>
<td>$o a \rightarrow o b$</td>
<td>$o b \implies o b$</td>
</tr>
<tr>
<td>CHAINResponse(a, b)</td>
<td>$o a \rightarrow o b$</td>
<td>$o b \implies o b$</td>
</tr>
<tr>
<td>PRECEDENCE(a, b)</td>
<td>$\neg o W a$</td>
<td>$b \implies \neg o a$</td>
</tr>
<tr>
<td>AlternatePRECEDENCE(a, b)</td>
<td>$(\neg o W a \land o b \implies \neg o W a)$</td>
<td>$b \implies o \land \neg o S a$</td>
</tr>
<tr>
<td>CHAINPRECEDENCE(a, b)</td>
<td>$(\neg o W a \land o b \implies \neg o W a)$</td>
<td>$(a \land b) \implies (a \implies o b) \lor (b \land o a)$</td>
</tr>
<tr>
<td>Succession(a, b)</td>
<td>$o a \rightarrow o b \land (\neg o W a)$</td>
<td>$(a \land b) \implies (a \implies o b) \lor (b \land o a)$</td>
</tr>
<tr>
<td>AlternateSuccession(a, b)</td>
<td>$o a \rightarrow o b \land (\neg o W a)$</td>
<td>$(a \land b) \implies (a \implies o b) \lor (b \land o a)$</td>
</tr>
<tr>
<td>CHAINSuccession(a, b)</td>
<td>$o a \rightarrow o b \land (\neg o W a)$</td>
<td>$(a \land b) \implies (a \implies o b) \lor (b \land o a)$</td>
</tr>
<tr>
<td>CoExistence(a, b)</td>
<td>$(o a \land o b) \lor (\neg o a \land \neg o a)$</td>
<td>$(a \land b) \lor (a \land b) \lor (b \land a)$</td>
</tr>
<tr>
<td>NotCoExistence(a, b)</td>
<td>$(o a \land \neg o b) \land (\neg o b \land \neg o a)$</td>
<td>$(a \lor b) \lor (a \lor b) \lor (b \lor a)$</td>
</tr>
</tbody>
</table>

Support [17] measures the frequency of the co-occurrence of the antecedent and the consequent. The Support of the sole antecedent of the rule is also called Coverage, while the Support of the consequent is called Prevalence.

Confidence [17] measures the co-occurrences of the antecedent and the consequent in the fraction of data containing the antecedent.

Recall [11] measures the co-occurrences of the antecedent and the consequent in the fraction of data containing the consequent.

Specificity [11] measures the co-absences of the antecedent and the consequent in the fraction of data not containing the antecedent.

Accuracy [11] measures the fraction of the data either containing both the consequent and the antecedent or neither of the two.

Lift [8] scales the Confidence by the probability of the consequent, to check if the co-occurrence of the antecedent and consequent is more likely than their independence.

Leverage [9] measures the difference between the Confidence of the rule and the independent occurrence of its variables.

Added Value [12] measures the difference between the Confidence of the rule and the probability of the consequent alone, to check if the conditioned occurrence of the consequent differ from its unconditioned occurrence.

Relative Risk [38] measures the ratio of the conditional probability of the consequent given the antecedent to the conditional probability of the consequent given the negation of the antecedent.

Jaccard’s Coefficient [39] measures the similarity between the variables using the ratio of their co-occurrence to the union of all their independent occurrences.

Certainty Factor [10] measures the ratio of the Added Value of the rule to the Added Value of the consequent alone, in order to see the variation of probability in the data containing the antecedent.

Odds Ratio [40] measures the ratio of the probability of having the consequent when the antecedent is present to the probability of having the consequent when the antecedent is not present.

Odds Multiplier [40] measures the ratio of the probability of having the antecedent when the consequent is present to the probability of having the antecedent when the consequent is not present.

Yule’s Q and Yule’s Y [41,42] are normalization of the Odds Ratio to have it centered around 0 and ranging between −1 and 1.

Klosgen’s Measure [43] weights the Support of the rule using its Added Value.

Conviction [13] measures the occurrences of the antecedent without the consequent in comparison to their independence.

Interestingness Weighting Dependency [44] combines Support and Lift of a rule and explicitly gives weights to each of them to let the user decide their relative importance.

Collective Strength [12] measures the ratio of the agreement ratio (number of non-violations per expected number of non-violations) to the violation ratio (number of violations per expected number of violations).
<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
<th>Range</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support</td>
<td>$P(AB)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confidence/Precision</td>
<td>$P(B</td>
<td>A)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverage</td>
<td>$P(A)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prevalence</td>
<td>$P(B)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recall</td>
<td>$P(A</td>
<td>B)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specificity</td>
<td>$P(\neg A</td>
<td>\neg B)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>$P(AB) + P(\neg A</td>
<td>\neg B)$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lift/Interest</td>
<td>$\frac{P(B</td>
<td>A)}{P(B)}$ or $\frac{P(AB)}{P(A)P(B)}$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leverage</td>
<td>$P(B</td>
<td>A) - P(A</td>
<td>B)$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Added Value/Change of Support/Centered Confidence</td>
<td>$P(B</td>
<td>A) - P(B)$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jaccard</td>
<td>$P(B</td>
<td>A) - P(B</td>
<td>\neg A)$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Certainty factor</td>
<td>$\frac{P(B</td>
<td>A) - P(B)}{1 - P(B)}$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Odds ratio/Bayes Factor</td>
<td>$\frac{P(A</td>
<td>\neg B)P(\neg A)-P(A\neg B)P(\neg A</td>
<td>\neg B)}{P(A</td>
<td>\neg B)P(\neg A)-P(A\neg B)P(\neg A</td>
<td>\neg B)}$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yule’s Q</td>
<td>$\frac{P(AB)(\neg A</td>
<td>B) - P(A</td>
<td>B)P(\neg AB)}{P(AB)(\neg A</td>
<td>B) + P(A</td>
<td>B)P(\neg AB)}$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yule’s Y</td>
<td>$\frac{\sqrt{P(AB)P(\neg A</td>
<td>B) - \sqrt{P(AB)P(\neg AB)}}}{\sqrt{P(AB)P(\neg A</td>
<td>B) + \sqrt{P(AB)P(\neg AB)}}}$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klosgen</td>
<td>$\frac{\sqrt{P(AB)}\times\max(P(B</td>
<td>A) - P(B), P(A</td>
<td>B) - P(A))}$</td>
<td>[−1, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conviction</td>
<td>$\frac{P(A</td>
<td>\neg B)P(\neg A)-P(A\neg B)P(\neg A</td>
<td>\neg B)}{P(A</td>
<td>\neg B)P(\neg A)}$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interestingness Weighting Dependency</td>
<td>$\left(\frac{P(AB)}{P(A</td>
<td>B)P(B)}\right)^n - 1 \times P(AB)^n$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collective Strength</td>
<td>$\frac{P(AB) + P(\neg A</td>
<td>\neg B)}{P(A</td>
<td>B)P(B) + P(\neg A</td>
<td>\neg B)} \times \left(\frac{P(A</td>
<td>B)P(B)}{P(A)P(B)}\right)$</td>
<td>[0, +∞)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laplace Correction</td>
<td>$\frac{N(AB)+1}{N(A)+2}$</td>
<td>[0.5, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gini Index</td>
<td>$P(A) \times (P(B</td>
<td>A)^2 + P(\neg B</td>
<td>A)^2)$ \quad + (P(\neg A) \times (P(B</td>
<td>\neg A)^2 + P(\neg B</td>
<td>\neg A)^2)$ \quad - P(B)^2 - P(\neg B)^2$</td>
<td>[0, 1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-Measure</td>
<td>$P(AB)\log_2 \frac{P(B</td>
<td>A)}{P(B)} + P(\neg A)\log_2 \frac{P(\neg B</td>
<td>A)}{P(\neg B)}$</td>
<td>(−∞, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One-Way Support</td>
<td>$P(B</td>
<td>A)\log_2 \frac{P(AB)}{P(A)P(B)}$</td>
<td>(−∞, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two-Way Support</td>
<td>$P(AB)\log_2 \frac{P(AB)}{P(A)P(B)}$</td>
<td>(−∞, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two-Way Support Variation</td>
<td>$P(AB)\log_2 \frac{P(AB)}{P(A)P(B)} + P(\neg A\neg B)\log_2 \frac{P(\neg A)}{P(\neg A)P(B)}$</td>
<td>(−∞, +∞)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)

**Laplace Correction** [45] is a variation of Confidence to take into account small data.

**Gini index** [46] measures if the entropy introduced by a rule brings a marked difference.

**J-measure** [47] is an entropy based measure for the information content of a rule.

**One-way Support and Two-way Support** [48] combine respectively Confidence and Support of a rule with the degree of independence between the variables.
Table 3 (continued).

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
<th>Range</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø–Coefficient</td>
<td>( \frac{P(AB) - P(A)P(B)}{\sqrt{P(A)P(B)P(\neg A)P(\neg B)}} )</td>
<td>((-\infty, +\infty))</td>
<td>sym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>var</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Platetsky–Shapiro</td>
<td>( P(AB) - P(A)P(B) )</td>
<td>([-1, 1])</td>
<td>sym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>var</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Cosine</td>
<td>( \frac{P(AB)}{\sqrt{P(A)P(B)}} )</td>
<td>([0, +\infty))</td>
<td>sym</td>
<td>(\downarrow)</td>
<td>var</td>
<td>var</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Loevinger</td>
<td>( \frac{P(A)P(\neg B)}{P(A-B)} )</td>
<td>((-\infty, 1])</td>
<td>asym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>const</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Information Gain</td>
<td>( \log \left( \frac{P(AB)}{P(A)P(B)} \right) )</td>
<td>((-\infty, +\infty))</td>
<td>sym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>var</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Sebag–Schoenauer</td>
<td>( \frac{P(AB)}{P(A-B)} )</td>
<td>([0, +\infty))</td>
<td>asym</td>
<td>(\rightarrow)</td>
<td>var</td>
<td>const</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Least Contradiction</td>
<td>( P(AB) - P(A-B) )</td>
<td>((-\infty, +\infty))</td>
<td>asym</td>
<td>(\downarrow)</td>
<td>var</td>
<td>var</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Odd Multiplier</td>
<td>( \frac{P(AB)}{P(B</td>
<td>A-B)} )</td>
<td>([0, +\infty))</td>
<td>asym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>const</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>Example and Counterexample Rate</td>
<td>( 1 - \frac{P(A-B)}{P(AB)} )</td>
<td>((-\infty, 1])</td>
<td>asym</td>
<td>(\rightarrow)</td>
<td>var</td>
<td>const</td>
<td>(\downarrow)</td>
<td></td>
</tr>
<tr>
<td>Zhang</td>
<td>( \frac{P(AB) - P(A)P(B)}{\max(P(AB)P(\neg B), P(B</td>
<td>A-B))} )</td>
<td>((-\infty, +\infty))</td>
<td>asym</td>
<td>(\downarrow)</td>
<td>const</td>
<td>const</td>
<td>(\downarrow)</td>
</tr>
</tbody>
</table>

**Two-way Support Variation** [48] measures the change in the Two-way-Support.

**Linear Correlation Coefficient** [49] measures the Pearson’s correlation between the variables.

**Platetsky–Shapiro** [9] measures the difference between the co-occurrences of antecedent and consequent and their independent frequency.

**Cosine** [37] measures the geometric mean between Lift and Support of a rule.

**Loevinger** [50] measures the homogeneity between antecedent and consequent.

**Information Gain** [51] is the logarithm of the Lift.

**Sebag–Schoenauer** [52] measures the proportion of positive and negative occurrences of the antecedent.

**Least Contradiction** [53] measures the difference between positive and negative occurrences of the antecedent weighted by its frequency.

**Example and Counterexample Rate** [11] measures the proportion of the antecedent occurrences with and without consequent.

**Zhang** [54] measures the positive or negative association between the antecedent and the consequent.

Different studies have been dedicated to the analysis of general properties for measures [9,11,15,37]. Properties show the response of measures under certain conditions. Therefore, properties can be used to group similar measures and decide the proper ones to be employed depending on the context. For example, we will analyze the sensitivity of measures to the increase of noise in the data as an important selection criterion for rule monitoring or discovery. We will delve deeper into this aspect in Section 6. In this paper, we focus specifically on a subset of the properties proposed in [37] and [15], as their meaning and effects are reportedly recognizable in a clear manner by the final user. The selected properties are explained below and associated to each measure \(\mathcal{M}\) in Table 3.

P1. **Null invariance** [37]. The measure is unaffected by traces not containing neither \(A\) or \(B\). Therefore, it assesses whether the traces not related to the rule affect the measurement or not. To satisfy this property, the measure should not vary when \(\neg A \neg B\) increases in the contingency table, while the other values remain fixed. In Table 3, we use the ‘\(\text{sym}\)’ or ‘\(\text{asym}\)’ symbols to indicate whether the property holds or not, respectively. For example, for Confidence and Recall this property holds, whereas for Support, Leverage and Collective Strength it does not.

P2. **Asymmetric processing of variables** [15]. The measure is asymmetric under variable swap, i.e., the measure of if \(A\) then \(B\) differs from that of if \(B\) then \(A\). The measure enjoys this property if it does not vary upon the swapping of the values of \(A\) and \(B\) in the contingency table. In Table 3, every measure is marked with “\(\text{asym}\)” or “\(\text{sym}\)” to indicate whether the property is enjoyed or not, respectively. For instance, Support and Leverage are symmetric under variable swap, whereas Confidence, Recall and Collective Strength provide an asymmetric processing of the variables.

P3. **Variation with occurrences of \(B\) in the absence of \(A\)** [15]. The value of the measure varies when the occurrences of \(B\) in the absence of \(A\) increase. In other words, this property focuses on whether the independent occurrence of \(B\) influences the measure. Given an if \(A\) then \(B\) rule, if \(B\) is very likely to occur regardless of \(A\), the influence of \(A\) on \(B\) may be questioned. To verify this, the value of the measure varies when \(\neg A \neg B\) increases in the contingency values (and the other values remain fixed). In Table 3, measures are marked with “\(\downarrow\)” if the variation is a decrease (as in the case of Support), “\(\uparrow\)” if it is an increase (e.g., Leverage), “\(\pm\)” if the variation can be either a decrease or an increase depending on the values of \(B\) (e.g., Collective Strength), “\(\approx\)” if the value does not vary at all (e.g., Confidence).

P4. **Reference situations: independence** [15]. If the variables are independent, then the measure exhibits a known value. The variables are considered independent when their joint probability is equal to the product of their respective probabilities, i.e., \(P(AB) = P(A)P(B)\). The measure should have a constant and known value in that case.
are labeled as “const” (e.g., Lift) if this property holds, and “var” otherwise (e.g., Support).

P5. Reference situations: logical rule [15]. If the rule is always satisfied, then the measure exhibits a known value. An if A then B rule is always satisfied if P(A→B) = 0. In other words, if there are no counterexamples in the data, the value of the measure that enjoys this property is a known constant (let it be a number or tendency to infinite).

Table 3 provides a list of measurements by building on LTLp

In the following section, therefore, we extend the aforementioned measures to temporal process rules.

4. Temporal-extended measurement framework

Our framework addresses the limits of Support and Confidence measurements by building on LTLp formal semantics and the spectrum of measures defined in different areas of computer science. Furthermore, it is generic as it allows for the usage of any probabilistic measure (including those of Table 3) on any temporal-logic-based rules specification. To this end, Section 4.1 formalizes the reactive temporal specification of rules, Section 4.2 discusses their probabilistic interpretation, and Section 4.4 defines the overall framework.

4.1. Reactive temporal specification

Our first building block is the concept of Reactive Constraint (RCon), originally introduced in [14], the paper which we extend here. A rule typically expresses that the occurrence of given preconditions (activator) implies certain consequences (target). The reactive nature of this kind of rule lies in the fact that the condition on the target is exerted only if the activator is verified. We codify this intuition in RCons, whose semantics is based on LTLp.

Definition 4.1 (Reactive Constraint (RCon)). Given an alphabet of propositional symbols Σ ∪ {start, end, True, False}, let ψa and ψt be LTLp formulae over Σ. A Reactive Constraint (RCon) is a pair (ψa, ψt) hereafter denoted as ψ ≜ ψa ⊢ ψt.

An RCon is interpreted as follows: Each time the activator is true, the target should be true at that point of the trace. For example, a ⊢ ψt is an RCon stating that every time a (the activator, ψa) is True, then also ψt (the target, ψt) must evaluate to True. That RCon corresponds to Response(a, c) in DECLARE as it requires that if a occurs in a trace, it must be eventually followed by c. e.g., a ⊢ a corresponds to Precedence(d, c) in DECLARE because it requires that every time c (the activator) occurs in a trace, then it has to be preceded by a (the target). Table 1 provides a list of standard DECLARE constraints expressed in the form of RCons.

Table 4 Evaluation (0 is False and 1 is True) and probabilistic interpretation of RCon a ⊢ ψt.

<table>
<thead>
<tr>
<th>Trace t1</th>
<th>(a, b, c, d, t1, s1, c, b, h)</th>
<th>ψa: a</th>
<th>1 0 0 0 0 0 0 0 0</th>
<th>ψt: c</th>
<th>1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(ψa, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td>P(¬ψa ∧ ¬ψt, t) = 1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(ψa, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td>P(ψa ∧ ¬ψt, t) = 1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(ψa, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td>P(¬ψa ∧ ¬ψt, t) = 1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(ψa, t) = 1/2</td>
<td>P(ψa ∧ ¬ψt, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P(ψa, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td>P(ψa ∧ ψt, t) = 1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An RCon that goes beyond the standard repertoire of DECLARE is (ϕb ∧ ◦e) ⊢ (¬e ∨ 0); its activator is the formula ϕb = b ∧ e, satisfied between the occurrence of b and the occurrence of e in a trace; its target is the formula ϕt = ¬e ∨ 0, which evaluates to True if either c is False, or c occurs and is eventually followed by t. Because at every event of the trace (i.e., any point in time) both the activator and target can be either True or False, the possible evaluation of an RCon can result in either of the following four combinations.

Definition 4.2 (RCon Evaluation). Given an RCon Ψ ≜ ψa ⊢ ψt, and a trace t of length n ∈ N, let i denote the ith event in the trace (1 ≤ i ≤ n). For each ti ∈ t the possible evaluations of Ψ are:

ψa = False, ψt = False if i, i ≠ ψa and t, i ≠ ψt;
ψa = False, ψt = True if i, i ≠ ψa and t, i = ψt;
ψa = True, ψt = False if i, i = ψa and t, i ≠ ψt;
ψa = True, ψt = True if i, i = ψa and t, i = ψt.

For example, the second and third rows of Tables 4–6 show the evaluation of RCons a ⊢ ψt (i.e., Response(a, c) in Table 4), c ⊢ a (i.e., Precedence(d, c) in Table 5) and (ϕb ∧ ◦e) ⊢ (¬e ∨ 0) (Table 6) on trace (a, b, c, d, t, c, e, c, h). Notice that ψa and ψt are evaluated separately at every event of a trace.

The RCon evaluation can be performed efficiently based on the automaton-based techniques defined in [14], adapting it for offline verification. The full discussion on this aspect can be found in Appendix, but we briefly outline the rationale here. Intuitively, we resort to [14, Theorem 4]: An RCon can be separated in pure-past, pure-present and pure-future components. The respective sub-formulae contain only past temporal operators, none,
or only future ones, respectively. As they are LTLp formulae, all components correspond to finite state automata (FSAs). The key point is that, by mirroring pure-past formulae and reversing their automata, a single replay of the sub-trace from the beginning to the activator event keeps track of the truth value of the pure-past formula till that point. As we have knowledge of the whole trace, and thus of the suffix too a fortiori, we can apply the same principle to pure-future formulae too: A single replay from the end of the trace to the activator event keeps track of the truth value of the pure-future formula from that point onwards.

From this optimization, it follows that any LTLp formula can be evaluated at each event reading the trace only twice (as in [4, 55] though for any RCon and not just DECLARE constraints): Once from $t_{\text{start}}$ to $t_{\text{end}}$ (past components) and once from $t_{\text{end}}$ to $t_{\text{start}}$ (future components). This result implies that the computational cost depends linearly on the number of events in the event log and in the number of rules to verify. Specifically, given an event log $L$ of cardinality $|L|$, assuming that (i) every trace $t \in L$ has a length of up to $n$, and (ii) $|R|$ rules are under analysis, the cost to evaluate all rules on $L$ is: $O(|L| \times n \times |R|)$.

### 4.2. Probabilistic interpretation on a trace

The evaluation of RCons indicates whether a rule holds true or false within a trace. In real life, traces often contain noise or partially deviate from desired process specifications. In those occasions wherein the trace may contain also events that do not satisfy the rule, we are interested in understanding to what degree a rule is satisfied. As we have previously defined the notion of satisfaction for $\varphi_a$ and $\varphi_t$ on single events (Definition 4.2), we can devise a probabilistic interpretation for RCons over traces.

**Definition 4.3 (Probability of an LTLp Formula in a Trace).** Given an LTLp formula $\varphi$ and a trace $t$ of length $|t| = n$, we define the probability of $\varphi$ in $t$ as the proportion of the events in $t$ that satisfy $\varphi$:

$$P(\varphi, t) = \frac{|\{i \in [1, n] : t[i] = \varphi\}|}{n}.$$

**Definition 4.4 (Joint Probability of LTLp Formulae in a Trace).** Given two LTLp formulae $\varphi_1$ and $\varphi_2$ and a trace $t$ of length $n$, we define the probability of the intersection of $\varphi_1$ and $\varphi_2$ in $t$ (joint probability) as the proportion of the events in $t$ that satisfy both $\varphi_1$ and $\varphi_2$:

$$P(\varphi_1 \land \varphi_2, t) = \frac{|\{i \in [1, n] : t[i] = \varphi_1 \land t[i] = \varphi_2\}|}{n}.$$

The probabilities of the evaluations of activator and target of an RCon follow from the above definitions (Table 7 shows the resulting contingency table):

<table>
<thead>
<tr>
<th>$\neg \varphi_t$</th>
<th>$\varphi_t$</th>
<th>$\neg \varphi_a$</th>
<th>$\varphi_a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\neg \varphi_a \land \neg \varphi_t, t)$</td>
<td>$P(\neg \varphi_a \land \varphi_t, t)$</td>
<td>$P(\varphi_a \land \neg \varphi_t, t)$</td>
<td>$P(\varphi_a \land \varphi_t, t)$</td>
</tr>
</tbody>
</table>

Table 7 Contingency table of the probabilities of an RCon $\varphi_a \Rightarrow \varphi_t$, in a trace.

2 Notice that we use the comma in $P(\varphi, t)$ and similar following expressions to separate the parameters, namely the formula to be evaluated (here, $\varphi$) and the structure on which the formula is analyzed (here, $t$).

### 4.3. Probabilistic interpretation on an event log

Following the definition for LTLp formulae over traces, it is of interest to define similar probabilities over event logs. Intuitively, if the trace probabilities assess the likelihood of the rule correctness in events within a trace, event log probabilities should question the likelihood of the rule correctness in the traces of an event log. As previously mentioned, the descriptive statistics of trace measures across an event log are suitable for this purpose because they preserve the focus on the events. In order to achieve this goal, we have to first derive the conditional probability of the target given the activator in a trace, i.e., $P(\varphi_t|\varphi_a, t)$. Intuitively, this is the probability of the target to hold true when the activator holds true. Notice that this viewpoint is conceptually closer to the notion of Reactive Constraint than the joint probability of activator and target. Furthermore, the conditional interpretation of rules is also more in line with their
human interpretation [56]. This makes the conditional probability a suitable means for the probabilistic analysis of a constraint in a trace as a whole.

**Definition 4.5 (Conditional Probability of LTLp, Formulas in a Trace).** Given two LTLp formulas \(\varphi_1\) and \(\varphi_2\) and a trace \(t\) of length \(n\), we define the conditional probability of \(\varphi_2\) given \(\varphi_1\) over \(t\) as the proportion of events satisfying \(\varphi_2\) among those that satisfy \(\varphi_1\):

\[
P(\varphi_2|\varphi_1, t) = \frac{|\{i \in [1, n] : t_i = \varphi_1 \text{ and } t_i = \varphi_2\}|}{|\{i \in [1, n] : t_i = \varphi_1\}|}.
\]

From the above definition, it follows that:

\[
P(\varphi_1, t) = \frac{|\{i \in [1, n] : t_i = \varphi_1 \text{ and } t_i = \varphi_1\}|}{|\{i \in [1, n] : t_i = \varphi_1\}|},
\]

\[
P(\neg \varphi_2, t) = \frac{|\{i \in [1, n] : t_i = \varphi_1 \text{ and } t_i = \varphi_2\}|}{|\{i \in [1, n] : t_i = \varphi_1\}|},
\]

\[
P(\varphi_1 \land \neg \varphi_2, t) = \frac{|\{i \in [1, n] : t_i = \varphi_1 \text{ and } t_i = \varphi_2\}|}{|\{i \in [1, n] : t_i = \varphi_1\}|},
\]

\[
P(\neg \varphi_1 \land \neg \varphi_2, t) = \frac{|\{i \in [1, n] : t_i = \varphi_1 \text{ and } t_i = \varphi_2\}|}{|\{i \in [1, n] : t_i = \varphi_1\}|}.
\]

Tables 4–6 show the conditional probabilities of RCon (\(\varphi \land \neg \varphi \rightarrow (\neg \varphi \lor \varphi)\) on trace \((a, b, c, d, e, f, e, c, e, h)\). Notably, the conditional probability is not influenced by the total amount of events in the trace, but only by the events of interest.

To devise the probability of an RCon in an event log \(L\) (henceforth, event log probability), we have to detect the portion of the event log satisfying an LTLp formula. To this end, we split the event log into a sub-log that has only the traces in which the activator occurs at least once (i.e., every \(t \in L\) such that \(P(\varphi_1, t) > 0\)), and the complementary sub-log consisting of the traces in which the activator does not occur (i.e., every \(t \in L\) such that \(P(\varphi_1, t) = 0\)). Given the above considerations and the definition of conditional probability for RCons in single traces (Definition 4.5.), we devise a probabilistic interpretation for RCons over event logs as follows.

**Definition 4.6 (Conditional Probability of LTLp, Formulas in an Event Log).** Let \(\varphi_1\) and \(\varphi_2\) be two LTLp formulae and \(L\) an event log of cardinality \(|L|\). We say that \(\varphi_1\) is non-null in a trace \(t \in L\) if and only if \(P(\varphi_1, t) > 0\). If \(P(\varphi_1, t) = 0\), we say that \(\varphi_1\) is null in \(t\). The conditional probability of \(\varphi_2\) given \(\varphi_1\) in \(L\) is the portion of the event log that consists of traces for which \(\varphi_1\) is non-null and satisfies \(\varphi_2\), given the satisfaction of \(\varphi_1\):

\[
P(\varphi_2|\varphi_1, L) = \frac{\sum_{t \in \text{L} \cap \varphi_1} P(\varphi_2|\varphi_1, t)}{|L|}.
\]

The conditional probability of \(\varphi_2\) given \(\neg \varphi_1\) in \(L\) is the portion of the event log that consists of traces for which \(\varphi_1\) is null and satisfies \(\varphi_2\), given the satisfaction of \(\neg \varphi_1\):

\[
P(\varphi_2|\neg \varphi_1, L) = \frac{\sum_{t \in \text{L} \cap \neg \varphi_1} P(\varphi_2|\neg \varphi_1, t)}{|L|}.
\]

Table 10 shows the resulting contingency table. In the following, we provide the proof of the correctness of our approach.

**Theorem 4.1 (Contingency of Event Log Conditionals).** Given two LTLp, formulae \(\varphi_1\) and \(\varphi_2\) and an event log \(L\) of cardinality \(|L|\), let \(|L|_{\varphi_1} > 0\) be the number of traces in which \(\varphi_1\) is non-null and \(|L|_{\neg \varphi_1} = 0\) the number of traces in which \(\varphi_1\) is null. It follows that

\[
P(\varphi_2|\varphi_1, L) + P(\neg \varphi_2|\varphi_1, L) + P(\varphi_2|\neg \varphi_1, L) + P(\neg \varphi_2|\neg \varphi_1, L) = 1.\]

**Proof.** In light of the fact that there cannot be a trace where \(P(\varphi_1)\) is both 0 and not 0 at the same time, the proof of Theorem 4.1 proceeds as follows.

\[
P(\varphi_2|\varphi_1, L) = \frac{\sum_{t \in \text{L} \cap \varphi_1 > 0} P(\varphi_2|\varphi_1, t)}{|L|} + \frac{\sum_{t \in \text{L} \cap \varphi_1 = 0} P(\varphi_2|\varphi_1, t)}{|L|} + \frac{\sum_{t \in \text{L} \cap \varphi_1 > 0} P(\varphi_2|\neg \varphi_1, t)}{|L|} = 1.
\]

We remark that the non-trivial mapping from \(P(AB)\) to \(P(\varphi_2|\varphi_1, L)\) is intuitively rooted into the inherent nature of "if \(A\) then \(B\)" rules such as the RCons, as evidenced in [56], and its soundness is evidenced by Theorem 4.1. For example, Table 11 shows a few measures computed from the probabilities of the RCon \((\varphi \land \neg \varphi \rightarrow (\neg \varphi \lor \varphi))\) over a log composed of five traces. We remark that this result is distinct from the mere aggregation of trace measures. For example, comparing the average of the Support values in Table 9 (0.44) and the Support value presented in Table 11 (0.67), we observe that the former is the average...
proportion of events in a trace satisfying both the target and the activator, while the latter represents the proportion of traces of the event log satisfying the RCon.

### 4.4. Measurement system

Given an event log \( L \), a set of RCons \( R \), and a set of probabilistic measures \( M \) as input, our framework returns the measurement of every measure in \( M \) for each constraint in \( R \) both over every single trace \( t \in L \) and over the entire event log \( L \). More specifically, the output can be reported at three different levels of detail:

- **Event level**: distinct evaluation of \( \varphi_a \) and \( \varphi_e \) of each constraint in \( R \) on every event of every trace in \( L \).
- **Trace level**: measurement of each measure in \( M \) for each constraint in \( R \) for every trace in \( L \);
- **Aggregated view**: descriptive statistics over the event log of all the trace-level measures.

### Event log level

Measurement of every measure in \( M \) for every constraint in \( R \) for the entire event log \( L \).

For example, Table 9 shows some trace level measures together with their descriptive statistics and Table 11 shows the corresponding event log level measures for the RCon \( (\varphi_b \land \varphi_a) \rightarrow (\neg \varphi_c \lor \varphi_t) \). Since being able to perceive the overall status of a constraint is as important as the possibility to analyze its details in single traces, we report the entire statistical distribution of a measure across the event log to provide a complete information spectrum.

Fig. 1 depicts the pipeline of the framework from the input to the output. In the first stage, an RCon is evaluated on each trace of the event log. Then, the evaluation result is used to compute the probabilities of the rule. On top of them, the measures of the rule in each trace and in the entire event log are computed. Also, descriptive statistics over the event log are reported for each trace measure.

We remark that the design of the RCons is crucial for the evaluation and the computation of the measures especially in terms of definition of their activator. Let us take as an example the RespondedExistence\((a, b)\) constraint from the repertoir of DECLARE (see Table 1). The classical LTL \( \varphi \) formula underlying RespondedExistence\((a, b)\) for whole-trace evaluations is

\[ \neg \varphi_a \lor \varphi_b \] [28]. However, the formulation of the rule as an RCon can lead to different interpretations:

- **\( a \rightarrow (\varphi_b \lor \varphi_a) \)**: If an occurs, \( b \) is expected to occur somewhere in the trace;
- \((\varphi_a \lor \varphi_b) \rightarrow (\varphi_b \lor \varphi_a)\): For every event in the trace such that \( a \) occurs either in the past or in the future, also \( b \) should occur somewhere in the trace;
- **\( True \rightarrow \neg (\varphi_a \lor \varphi_b) \lor (\varphi_b \lor \varphi_a) \)**: For every event in the trace, if \( a \) occurs in the trace, also \( b \) is expected to occur;
- **\( \tau_{\text{Start}} \rightarrow \neg (\neg \varphi_a \lor \varphi_b) \)**: At the beginning of the trace, if \( a \) occurs in the trace, also \( b \) should occur.

All the formulations above are legitimate as they entail that the occurrence of \( a \) in the trace demands the occurrence of \( b \). However, the difference in the way the activator is represented turns out to be crucial. The activator, indeed, encodes when the rule is of interest. For example: Are we interested in each occurrence of task \( a \) or only in its eventual occurrence in the trace? Do we want the rule to be satisfied in every point of the trace or just at the beginning of the trace? These choices have a clear impact on the measures. Table 12 presents the evaluation of a trace with the different formulations seen above and their trace measurements for Confidence and Support. Confidence is equal to 1 as each time the activator holds true, also the target holds true. Support (i.e., the frequency of \( \varphi_a \land \varphi_e \)) varies considerably instead. Notice that this phenomenon comes with neither a good nor with a bad connotation, but stresses the idea that a full control over the formula implies a mindful decision about its design and subsequently on picking the right measures for it. In Table 1, we devised the RCon formulae based on the activators of DECLARE templates described in [3,20] — hence, e.g., the choice for RespondedExistence of the first option presented in Table 12. While DECLARE templates are reasonably simple and well-known standard cases, encoding the right activator is crucial for the design of custom RCons and their measures. Lastly, we would like to remark that all the variants in Table 12 take exactly the same amount of computational time to be checked as any formula requires a trace to be read only twice, as described in Section 4.1.

### Table 10

<table>
<thead>
<tr>
<th>( \varphi_e )</th>
<th>( P(\varphi_e, t) &gt; 0 )</th>
<th>( P(\varphi_e, t) = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi_e )</td>
<td>( \sum_{t \in L} P(\varphi_e, t) )</td>
<td>( \sum_{t \in L} P(\neg \varphi_e, t) )</td>
</tr>
<tr>
<td>( \neg \varphi_e )</td>
<td>( \sum_{t \in L} P(\neg \varphi_e, t) )</td>
<td>( \sum_{t \in L} P(\varphi_e, t) )</td>
</tr>
<tr>
<td>( \sum_{t \in L} P(\neg \varphi_e, t) )</td>
<td>( \sum_{t \in L} P(\neg \varphi_e, t) )</td>
<td>( \sum_{t \in L} P(\varphi_e, t) )</td>
</tr>
</tbody>
</table>

### Table 11

<table>
<thead>
<tr>
<th>Event log</th>
<th>( P(\varphi, t) )</th>
<th>( P(\varphi_e, t) )</th>
<th>( P(\neg \varphi, t) )</th>
<th>( P(\neg \varphi_e, t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 = (a, b, c, d, e, f, g, h) )</td>
<td>( &gt; 0 )</td>
<td>0.83</td>
<td>0.17</td>
<td>0.67</td>
</tr>
<tr>
<td>( t_2 = (b, a, f, e, g, d, a, d) )</td>
<td>( &gt; 0 )</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>( t_3 = (a, b, c, d, e, f, g) )</td>
<td>( &gt; 0 )</td>
<td>1.00</td>
<td>0.00</td>
<td>0.80</td>
</tr>
<tr>
<td>( t_4 = (a, b, c, e, a) )</td>
<td>( &gt; 0 )</td>
<td>0.50</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>( t_5 = (a, b, c, e, d) )</td>
<td>( = 0 )</td>
<td>NaN</td>
<td>NaN</td>
<td>0.75</td>
</tr>
</tbody>
</table>

| \( P(\varphi, L) \) | 0.80 | 0.67 | 0.15 | 0.05 |
| \( P(\varphi_e, L) \) | 0.82 | 0.67 | 0.13 | 0.05 |
| \( P(\neg \varphi, L) \) | 0.15 | 0.13 | 0.15 | 0.15 |
| \( P(\neg \varphi_e, L) \) | 0.15 | 0.13 | 0.15 | 0.15 |

Support: 0.67 Confidence: 0.83 Specificity: 0.25 Lift: 1.02
both trace and event log levels. The framework is designed to be suitable for any custom formula in the form of a Reactive Constraint, and any measure that is based on the probability of the activator and target of the constraints. Therefore, it supports template sets like DECLARE and all the interestingness measures from association rule mining seen in Table 3, though not being limited to them. Next, we evaluate our approach through tests conducted with our implemented prototype.

5. Implementation and performance analysis

We have implemented our measurement framework as a proof-of-concept software prototype built upon the existing declarative process specification processor tool Janus [14,16]. The Java source-code can be found at https://oneiroe.github.io/DeclarativeMeasurements-static. The core component of the software is the RCons verification engine, upon which are built independently a declarative process discovery module and the present declarative rules measurement module. All the process specifications used in the following experiments are discovered with this discovery module implementing the technique presented in [14]. In the remainder of this section, we first report on the results of a time and space analysis with simulated data. Then, we investigate the computational performance on real-world event log datasets. The results demonstrate the practical feasibility and applicability of our approach.

5.1. Time analysis

To assess the efficiency of our implemented technique, we measure its time performance against an increase in the data size (i.e., the cardinality of the event log and the length of its traces) and the model size (i.e., the number of rules) with synthetic event logs. We repeated every experiment 10 times to smooth random

Factors. The reported results average over the ones of the single repetitions. The machine used for the experiments was equipped with an Intel Core i5-7300U CPU at 2.60 GHz, quad-core, 16 Gb of RAM and an Ubuntu 18.04 LTS operating system.

To test the response of our implemented framework against the input data size, we set up a controlled experiment in which we first generate logs of varying sizes that are compliant with a fixed set of rules, resorting to the simulation engine of MINERFUL [57]. Thereupon, we compute the measures listed in Table 3 against all the rules of a larger test specification (not fully compliant with the event log). For every run, we recorded the wall-clock time of our prototype.

The starting set of rules stems from the DECLARE repertoire of templates [5] and is provided in Table 13. Notice that the set contains all the rule templates seen in Table 1 and is designed in a way that every constraint insists on different tasks.

The test model consists of 649 constraints extracted by the discovery algorithm of Janus (setting the Support and Confidence threshold parameters to 0.05 and 0.8) from a synthetic event log of 834 963 events, 500 traces and tasks in \{a, ..., z, 0, 1\} that is compliant with the initial model.\(^3\) Given the test specification obtained as described above, we performed two tests of 65 iterations (1) increasing the length of the traces (with a step of 100 events per iteration, keeping the number of traces per event log equal to 500), and (2) increasing the number of traces in the event log (with a step of 50 new traces per iteration, keeping the trace lengths between 900 and 1000 events). Fig. 2 illustrates the results of both experiments. We observe that the factor actually influencing the wall-clock time is the total amount of events rather than the trace length: indeed, Fig. 2 shows that the recorded timings of both experiments tend to lie on the same line. This experimental result confirms the linear relation between the total number of events in the log and the computational performance illustrated in Section 4.1.

Next, we investigate the response of the framework to an increase in the model size. To do so, we first generate an event log containing 1000 traces with a trace length between 100 and 500

\(^3\) Available at https://oneiroe.github.io/DeclarativeMeasurements-static.

Table 12

Measurements of a constraint expressed with different formulations on trace (d, a, b, c, a).

<table>
<thead>
<tr>
<th>RCon formulation</th>
<th>Evaluation</th>
<th>Support (P(w, t))</th>
<th>Confidence (P(x, y, t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a \leftarrow (ob \lor qb))</td>
<td>(w_0: (0, 1, 0, 0, 1))</td>
<td>(\frac{2}{5} = 0.4)</td>
<td>(\frac{2}{5} = 1)</td>
</tr>
<tr>
<td>((oa \lor qa) \leftarrow (ob \lor qb))</td>
<td>(w_0: (1, 1, 1, 1, 1))</td>
<td>(\frac{2}{5} = 1)</td>
<td>(\frac{2}{5} = 1)</td>
</tr>
<tr>
<td>(True \leftarrow \neg(oa \lor qa) \lor (ob \lor qb))</td>
<td>(w_0: (1, 1, 1, 1, 1))</td>
<td>(\frac{2}{5} = 1)</td>
<td>(\frac{2}{5} = 1)</td>
</tr>
<tr>
<td>(End\leftarrow (oa \lor qa) \lor (ob \lor qb))</td>
<td>(w_0: (0, 1, 0, 0, 0))</td>
<td>(\frac{2}{5} = 0.2)</td>
<td>(\frac{2}{5} = 1)</td>
</tr>
</tbody>
</table>

Table 13

The set of DECLARE rules used in the experiments.

<table>
<thead>
<tr>
<th>Rule</th>
<th>(INF(a))</th>
<th>(RESPONSE(a, t))</th>
<th>(CHAINRESPONSE(a, p))</th>
<th>(END(x))</th>
<th>(PRECEDENCE(q, h))</th>
<th>(CHAINPRECEDENCE(q, p))</th>
<th>(ATMOSTONE(x))</th>
<th>(ALTRESPONSE(l, t))</th>
<th>(RESPONDBARRED(x, t))</th>
<th>(SATISFIES(t, r))</th>
<th>(ATMOSFAILURE(t))</th>
<th>(NOEXISTS(t))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\neg(c))</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
<td>(\neg)</td>
</tr>
<tr>
<td>(</td>
<td>)</td>
<td>(</td>
<td>)</td>
<td>(</td>
<td>)</td>
<td>(</td>
<td>)</td>
<td>(</td>
<td>)</td>
<td>(</td>
<td>)</td>
<td>(</td>
</tr>
<tr>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
<td>()</td>
</tr>
</tbody>
</table>

Fig. 1. Measurement framework architecture.
events from the simulation of the rules in Table 13. Thereupon, we use the discovery algorithm of Janus to automatically retrieve different test models with varying levels of compliance. To that extent, we make the Confidence threshold range from 1.0 (full model compliance), down to 0.0 with a step of 0.05. The rationale is, the lower the Confidence threshold, the higher the number of constraints in the test model. Then, we calculate all the measures in Table 3 for every constraint of each test model. The time taken for the measurements are shown in Fig. 3. Notice that the computation time is linearly dependent on the number of rules to check, thus in line with the theoretical computational cost exposed in Section 4.1.

5.2. Space analysis

The space consumption of our technique depends on the data structures required to store the multi-level results. More specifically, four multidimensional matrices are used, containing respectively (1) the evaluation at the event level, (2) the measures at the trace level, (3) their descriptive statistics over the log, and finally (4) the measures at the event log level.

Considering \( |L| \) the number of traces in the log, \( |E| \) the total number of events in a log, \( |R| \) the number of constraints, \( |M| \) the number of measures, the sizes of the matrices are respectively the following:

1. Events evaluation: \( |E| \times |R| \), wherein each cell contains two boolean values (i.e., the evaluation of the activator and target of the constraint);
2. Trace measures: \( |L| \times |R| \times |M| \), having a real number in every cell;
3. Trace measures statistics: \( |R| \times |M| \), containing seven real numbers per cell (for the mean, geometric mean, variance, population variance, standard deviation, maximum value, and minimum value, respectively);
4. Event log measures: \( |R| \times |M| \), with a real number each.

The events matrix is optimized as a bit matrix, where two bits are sufficient to store the boolean results of the evaluation of both the activator and target for one event. We implemented our framework in Java, so we employ 1-byte Byte objects and 4-byte Float numbers (6 decimal digits are sufficiently accurate for our purpose). Taking these indicators into account, we can estimate the space consumption. For example, assuming that \( |L| = 1000 \), the maximum number of events in a trace \( n \) is 50, \( |R| = 100 \), \( |M| = 30 \), the expectation for the space demands are distributed as follows:

1. Events evaluation: \( 1000 \times 20 \times 260 \times 1 = 5200000 \) bit = 5.2 Mb;
2. Trace measures: \( 1000 \times 260 \times 37 \times 4 = 38480000 \) bit = 38.48 Mb;
3. Trace measures statistics: \( 260 \times 37 \times 7 \times 4 = 269360 \) bit = 269.36 Kb;
4. Event log measures: \( 260 \times 37 \times 4 = 384800 \) bit = 38.48 Kb.

Therefore, the most memory-demanding data structures are those that pertain to the events evaluation and the trace measurements matrices. The former is bigger than the latter only if the average number of events per trace is greater than 4 times the number of measures used, i.e., \( \frac{|E|}{|M|} > 4 \times |M| \). In our experiments, even using all the 37 measures of Table 3, this has not occurred.

As with the experiments for the evaluation of time, we analyze empirically the space consumption through simulations, controlling the number of events \( n \) per trace, the number of traces \( |L| \), and the number of constraints under analysis \( |M| \). To measure the memory consumed by the data structures, we perform a Bit serialization of the matrix objects listed above. This allows us to have a precise measure of the space consumed by every object, though it unavoidably requires that the available memory is twice as much as the strictly necessary amount.

Fig. 4 illustrates the results of our experiments. As it can be seen, the resulting linear trends are in line with the expectations, modulo the constant factors introduced by the Java Virtual Machine objects. It can be noticed that the number of constraints to check, being a common factor among all the objects, increases the overall required memory quicker than the other parameters. We remark that depending on the desired outcome, not all the measures nor all the matrices are necessary. For example, if the log measures are desired, the trace measures and their statistics can be ignored and vice-versa. The events evaluation is the only mandatory object upon which all other computations are based.

At present, our implementation works in-memory, thus it is assumed that all the objects fit in main memory (which proved to be sufficient for all the real life log under analysis). However, we would remark that every measure calculation (both at log and trace levels) is independent from the other one, thus it is possible to either (i) compute one measure or constraint at the time in pipeline, in order to reduce the memory load, or (ii) to distribute the workload, by making each system compute independently one measure per constraint. Both are interesting directions for the future upgrades of our implementation.
5.3. Application on real-world event logs

To test the performance also in real settings, we compute the rule measures on 13 event logs, whose characteristics are exposed in Table 14. Twelve of those event logs are openly available\(^4\) and belong to the Business Process Intelligence Challenge (BPIC) collection, a Road-Traffic Fines Management Process (RTFMP) and the aforementioned Sepsis event log. In addition, we analyze the performance of our prototype on an event log stemming from a partner of a smart-city project in which the authors are involved (labeled as “Smart city” in Table 14). We included the Smart city event log due to its considerable size: as it can be noticed from the table, it is the one bearing the largest amount of events in this experiment.

For each log, we ran the discovery algorithm of Janus [14] in order to extract a test model to check the event log against. We tuned the parameters of the discovery algorithm to obtain a set of rules to which the event log complies for the most part (Confidence threshold of 0.8), even though the constrained tasks are possibly infrequently co-occurring (Support threshold of 0.05). Table 14 illustrates the results. For each event log we report, along with the number of traces, the occurring tasks, the events, the number of constraints in the test model, the total time from the launch to the termination of the software (“Time”), the time to evaluate the rules on events (“Checks”), the time to compute the measures both at the trace and log levels (“Measures”), and the total space consumed by the data structures of our tool (“Space”) against their expected value (“Expectation”). We remark that the space consumption is consistent with our theoretical expectation and that the wall-clock time remains within acceptable ranges as the slowest run takes around 4.5 min to check about 600 constraints in a considerably big event log such as BPIC17 [58] handling around 2.5 Gb of data.

5.4. Analysis of custom rules

In order to demonstrate that our framework can handle any Reactive Constraints, beyond the standard DECLARE repertoire, we applied our approach to compute the discussed measures of a custom rule on the Sepsis event log. We name the custom rule BidirectionalTimeConsequent(\(a, b, c\)) as its RCon formulation is \(a \rightarrow \diamond b \land \Diamond c\). It states that if \(a\) occurs, it is expected that either \(c\) occurred before it or \(b\) will occur afterwards. Table 15 reports the measures at log and trace level for BidirectionalTimeConsequent(Admission NC, CRP, IV Liquid) calculated on the Sepsis real-life log [34]. As it can be noticed, Confidence and Recall are relatively high (0.82 and 0.79, respectively) and the values of Coverage and Prevalence (0.76 and 0.79, respectively) suggest a frequent occurrence of activator and target. The value of Lift is greater than 1, which denotes dependency between activator and target (especially at a trace level). The detailed results of the evaluation on each trace can be found at oneiroe.github.io/DeclarativeMeasurements-static.

The capability of our framework to handle non-standard rules opens up new possibilities for the claimed extensibility of DECLARE as a declarative specification language, claimed from its very inception to be open to customization through the definition of new rules according to the process analyst needs [65].

6. Evaluation

In this section, we report on experiments that show interesting implications of having a vast availability of measures with customization options. Specifically, Section 6.1 investigates over which measures can be of interest in the scope of declarative specification discovery, and Section 6.2 shows how the properties of measures can be exploited to characterize the alterations of constraints when the underlying process changes.

All the experimental data (code, input data, results) can be found at https://oneiroe.github.io/DeclarativeMeasurements-static. In the following experiments, we resort to the following tools: (i) The Janus discovery algorithm [14] for the discovery of declarative models from event logs; (ii) The simulation engine of MINERful [57] for the generation of event logs complying with given declarative specifications; (iii) The error injection engine of MINERful [66] for the controlled insertion of noise into event logs; (iv) The declarative model simplification technique of MINERful [33] for the removal of redundancies from declarative specifications.

6.1. Ranking experiment

The objectives of this experiment are the following: (1) Empirically showing that relying on more measures than the sole Support and Confidence measures is effective to characterize process rules in an event log; (2) Highlighting insightful measures in a declarative process mining context.

To achieve both objectives we rank the measures according to how many correct and interesting rules they are able to recognize. We take inspiration from the correct rules at \(N\) experiment introduced by the seminal work of Le and Lo [6]. Given an event log \(L\), a ground truth set of rules \(R_G\) satisfied in \(L\), a set of rules \(R_D \supset R_C\) containing also loosely satisfied rules in \(L\), the set of interestingness measures \(M\) of Table 3, and a predefined threshold \(N\), we compute the value of each measure \(m \in M\) for all the rules \(r \in R_D\) on \(L\). Then, for every measure \(m \in M\) we create sets of rules that are associated to a common value and

\(^4\) https://data.4tu.nl/.
and confidence thresholds set to 0.05 and 0.5, respectively) in discovering a new process specification with loose bounds (Support with the rules in Table 13. From the simulated event log, we notice that the rules are designed to not interfere with the rules in Table 13. Together with the ranking, for each measure we report also the number of correct rules over the total number of rules in the top-1, 5, 10, 25, 50, 100, 200, 500, 1000, and 1500, i.e., ranging from considering only the best-scoring rules to considering all the rules inRp. The final ranking of a measure is computed as the average of its ranking for each N. Table 16(a) shows the final rankings for this experiment. Together with the ranking, for each measure we report also the number of correct rules ("Correct" column) and the average ratio of correct rules over the total number ("Ratio" column) in the top-N sets, averaged over the 10 repetitions of the experiment.

We run our experiments with three different event logs: (i) a simulation of a synthetic process specification (Section 6.1.1); (ii) a simulation of a synthetic process specification with random changes in the event log so as to mimic partial non-compliance (Section 6.1.2); (iii) a real-life event log (Section 6.1.3). At the end of this section, we draw some conclusions from the obtained results.

6.1.1. Process simulation

More specifically, we simulate the specification described in Table 13. Notice that the rules are designed to not interfere with one another and each of them contains different tasks. The simulation produces an event log that is fully compliant with the rules in Table 13. From the simulated event log, we discover a new process specification with loose bounds (Support and confidence thresholds set to 0.05 and 0.5, respectively) in order to discover also infrequent and seldom violated rules. We simplify the resulting set of rules by removing those that do not match yet strictly subsume or are entailed by the ground-truth rules, in order to avoid misleading results — for example, if \( \text{CHAINRESPONSE}(o, p) \) and \( \text{RESPONSE}(o, p) \) both belong to the returned set of rules, the former is retained and the latter is removed because \( \text{CHAINRESPONSE}(o, p) \) is part of the ground-truth specification (see Table 13) and is subsumed by \( \text{RESPONSE}(o, p) \).

The full detail of the technique that deals with the removal of redundant rules is out of scope for this paper. The interested reader can find a detailed description of the problem and the approach in [33]. The simplified discovered model consists of 1310 rules on average. Thereupon, we apply our measurement framework to compute the measures in Table 3 at the event log level for all the discovered rules. Finally, we sort the rules according to each measure, and rank the measures according to how many of the original rules are among in top N sets.

Notice that Support ranks only sixteenth. Confidence, by contrast, is at the top of the ranking. It should be observed, however, that the experiment considers by design never violated rules (i.e., those with maximal Confidence), hence the top position of this measure. Nevertheless, there are two measures that match Confidence, namely (i) Example and Counterexample Rate and (ii) the Sebag–Schoenauer measure. This is in accordance with PS, as the absence of counterexamples for the rules makes measures with known maximal values highly rank the original, correct rules. The "Ratio" column reported in Table 16(a) helps to distinguish the accuracy of the results. For example, Odd Multiplier and Odd Ratio have the same rank (i.e., the same amount of correct rules identified), although the Odd Ratio returns 3 times more rules than the Odd Multiplier within the top-N sets.

6.1.2. Process simulation with noise

The previous experiment tests a perfectly compliant setup where the reference rules are never violated. For this reason, we conduct a modified version of the experiment, this time injecting noise in the event log in order to check if the ranking is preserved.
in non-optimal situations. Specifically, we randomly delete or duplicate 5% of the occurrences of every task. The results can be found in Table 16(b). It can be seen that also in the presence of partial non-compliance, the measures of Confidence and Example and Counterexample Rate continue to be on top of the ranking, while Sebag–Schoenauer measure drops in the second half of the list, together with Ylue Q, Ylue Y and Odds Ratio. This sudden change due to noise is motivated by the fact that these measures are convex, as we discussed in regard with P6, so they rapidly decrease in presence of counterexamples. Support persists in the middle of the ranking, with 17 measures scoring better than it. Lift and Information Gain perform by far better in presence of noise. Understanding how measures react to changes in the data (noise in this case), is key for process drift analysis [67], where the evolution of the process is the main objective. We study more in details the influence of noise on the measurements in Section 6.2.

6.1.3. Real-life event log

We replicate our ranking experiment based on a real-life event log, specifically the Sepsis data-set [34]. In [35], Mannhardt and Blinde illustrate a procedural model discovered with the help of domain experts representing the sepsis treatment process at a hospital. We manually translated the model into Declare rules that we use as a ground-truth specification to replicate the ranking experiment. The model consists of 100 rules, listed in Table 17. Because it is a real life event log, we do not inject any noise. Table 16(c) illustrates the results. It can be seen that the measures of Confidence and Example and Counterexample Rate remain at the top of the ranking. Also, while Recall conquers the first position for the amount of correct rules reported, it also returns a high number of other incorrect rules as signaled by its low average ratio score. In this case, Support drops at the bottom of the ranking.

These experiments suggest possible candidate measures to be used for declarative process discovery. We tested different scenarios, thereby showing how the measures’ scores vary in each of them and how the evaluation of rules can benefit from the perspectives of multiple measures that were not previously available for temporal specifications. However, we remark that some measures should be handled in a specific manner. For example, Coverage is a descriptive measure reporting on the portion of the event log in which the rule activator occurs: this characteristic determines its low ranking in all the previous experiments. A low score in these experiments does not imply that a measure is of scarce use in general, though. Indeed, we can only highlight the average top ranking measures of these experiments as excellent options, as we do in the next section while depicting the current
results. The informed proposal of best practices for the selection of measures and combinations thereof depending on the analysis purposes (e.g., discovery) constitutes a highly interesting outlook for research.

6.2. Sensitivity and resilience to noise

In this section, we study in details the effect of noise injection in the event log on the constraints log measurements. We experimentally observed in Section 6.1.2 that measures exhibit different changes upon the presence of alterations from the expected behavior in an event log. A measure may “sense” the alteration in the data with respect to a rule or remain unaffected. Also, if the alteration is perceived, the magnitude of the measures reaction may be different, in light of the different properties that the measures enjoy, as discussed in Section 3.2. Markedly, we will empirically demonstrate that the properties originally for the measures at varying noise levels targetting one constraint at a time, thus limiting the effect of cross-interference.

Thereupon, we injected noise in the event log and calculated all the measures in Table 3 for the reference model. In particular, we made use of the following types of noise [66]:

- **Events insertion**: Spurious events are included in the traces (mimicking, e.g., double records, alien events, etc.);

- **Events deletion**: Events are expunged from the event log (mimicking, e.g., missing records, uncommitted transactions, etc.);

- **White noise**: Events are randomly inserted and deleted.

Addressing one rule at a time, we studied (1) the direct effect of noise on that constraint by altering the occurrences of its activator and target via insertions and deletions, and (2) the indirect effect, by altering the occurrences of the other tasks in the event log with white noise. We made the noise spread all over the event log according to a controlled probability variable. For instance, setting the noise injection as the deletion of occurrences of task a with a probability of 20% results in the removal of 20% of the occurrences of task a from the event log, picked at random.

More specifically, for every rule in the set of Table 13, we ran a separate experiment for (i) event insertion noise affecting the activator or (ii) the target, (iii) event deletion noise affecting the activator or (iv) the target, (v) white noise affecting neither the activator nor the target. For each of the combinations above, we let the error-injection probability range from 0 to 100% with a step of 10%. Because of the random factor, we repeated each experiment 10 times and recorded the average results.

Fig. 5 shows the results of our experiments on constraint Response(a, t). The Response constraint imposes that the target
Fig. 5. Effect of error injection on constraint $\text{RESPONSE}(e, f)$ for the best 10 scoring measures from the ranking experiment in Section 6.1.

The deletion of events that satisfy the target leads to more violations of the rule, i.e., lower $P(\psi_\tau | \psi_\alpha, L)$ and higher $P(\neg \psi_\tau | \psi_\alpha, L)$. The negative effect is thus reflected in the constant decrease of all measures. The rapidity at which
this phenomenon occurs is particularly interesting. In accordance with property P6, the decrease of concave measures (e.g., Example and Counterexample Rate) is slower than that of linear ones (e.g., Confidence), and convex measures decrease faster than all the others (e.g., Sebag-Schoenauer).

- The deletion of events that satisfy the activator, instead, does not bring further violations, but a higher $P(\neg \psi, L)$. As a consequence, the measures are mostly stable until the error rate is close to 100%. As per P5, measures with a decreasing trend (e.g., Laplace Correction or Accuracy) are susceptible to the frequency of the rule activators in the event logs, while measures depending only on rule satisfaction (e.g., Conviction and Odd Multiplier) are constant at all the error rates except 100% (in which case, they are no longer defined due to a zero-by-zero division in their formula).

- The insertion of more events that satisfy the target lead to a higher $P(\psi, L)$ but does not influence the constraint satisfactions, as $P(\psi, |\psi, L)$ remains constant. We remark that the target $\psi$ of RESPONSE(e, t) is $\preceq$, thus injecting more occurrences of $t$ does imply an increase of $P(\psi, L)$ proportional to the error rate. Following from this consideration and in light of P3, all the selected measures remain constant.

- The insertion of more events that satisfy the activator is less definite, as the new tasks may bring either to more rule satisfactions, i.e., higher $P(\psi|\psi, L)$, or violations, i.e., higher $P(\neg \psi|\psi, L)$. As RESPONSE(e, t) requires the occurrence of $t$ eventually after $e$ at any distance in the trace, in this experiment the injections mostly leads to satisfactions. That is why most of the measures show only a slightly decreasing trend. Notably, Conviction and Odd Multiplier measures sense this alteration in a marked way with respect to all the other rules.

- Lastly, the insertion of events not related to both the target and the activation mostly do not alter the measures. The satisfactions and violations remain constant, whilst the only increase is in $P(\neg \psi, L)$. Nevertheless, considering P1 and P5, we can distinguish the fluctuating measures due to the frequencies change (e.g., Cosine and Jaccard) from the unaffected ones (e.g., Confidence and Conviction).

Because of space limits, we cannot illustrate the outcome for the other rules of Table 13. The interested reader can find the full set of experimental data and results at https://oneiroe.github.io/DeclarativeMeasurements-static in a digital interactive format, which is more suitable for data exploration and browsing.

6.3 Discussion

The applications of our framework presented in Sections 6.1 and 6.2 allows for some reflections on the employment and availability of the measures. First, the measures evolution reflects whether the alteration in the data affects either the target or the activator of the rule (or both). This gives more insights into the analyzed phenomena than knowing whether a rule is satisfied or not at a given moment.

Nevertheless, given a constraint, not all the data alterations are perceived by all measures. This implies that depending on the requirements of the analysis the choice of the measure is crucial. We reported, for example, an experiment to identify measures suitable for discovery. Furthermore, given a certain alteration, it is possible to identify groups of measures with similar trends that focus on the same aspects of a rule, as it can be easily noticed in Fig. 5. Within such groups, it is then possible to select one representative measure.

To this extent, the properties discussed in Section 3.2 are clearly a guiding criterion for the selection of measures, markedly reflected in our experiments in which measures with similar trends could be distinctly identified by the properties they enjoy. Ultimately, this choice is strictly related to the requirements and goals of the event log analysis.

For instance, P6 turned out to be particularly relevant. While measures may have similar trends, the magnitude of such trends, reflected in the steepness of the curves with which the measure evolves (i.e., concave, convex, or linear), indicates how quickly the measure react to changes in the data. Furthermore, it shows the range of tolerance before the alteration in the data becomes too large to recognize the specific constraint behavior. A resilient measure with a slower decrease is desirable to sense if the fundamental characteristics of a log are still visible despite the deviations (e.g., to implement discovery algorithms that are robust to noise). On the other hand, a sensitive measure with a faster decrease is desirable when exceptions to the rules are only accepted in a very limited manner (e.g., to monitor normative processes).

An approach enabling such a vast number of measures for temporal specifications is presented in the seminal work of Le and Lo [6]. We extend their investigation along two main lines: first, our approach can handle arbitrary temporal formulae as activator and target, as opposed to single-task variables only (which also restrict the analysis to the sole Response and Precedence patterns). Second, our computation of the rule probabilities processes the whole trace at once, while the calculation scheme in [6] relies on a sliding window, whose size has to be manually set up thereby influencing the results.

7. Conclusion

In this paper, we presented a comprehensive measurement framework for declarative specifications modeled as Reactive Constraints. Given an event log and a set of custom probabilistic measures, the framework accepts in input any RCon and returns as output the evaluation of the rule for each event of the log, the computed measures for all the traces and their statistics over the entire event log, and the computed measures over the entire log. The framework goes beyond the current state of the art as it is not limited to a specific set of measures or rules. The experiments conducted reveal the possibility to characterize the behavior of a given constraint through the combination of different measures, which sense differently the behavior recorded in the log. Also, while the choice of the measures to employ is highly context-dependent, we showed how the measures properties can be used to guide the selection, as their effects are clearly visible in the results.

Future work Different possibilities are now open upon the foundations of this measurement framework. It is possible to exploit the possibility to characterize a phenomenon by studying the evolution of different measures for, e.g., the dynamic recognition of exceptions in process monitoring [69], the identification of process drifts [67] or the analysis of process variants [70]. Also, it can be employed as a post-processing tool for multi-measure filtering of the results of declarative process discovery techniques like [3,4,14].

As measures react differently to diverse stimuli for distinct types of rules, a method to find the best combination of measures depending on the analysis context turns out to be key. To this extent, future research could resort to existing techniques like [71] or develop novel multi-measure heuristics. Also, the measures can be integrated for the assessment of multi-constraint specifications as a whole as in [1,20].
While the analysis of multiple measures at once may be overwhelming for a human, machine learning techniques could benefit from the availability of the great amount of information returned by the proposed framework, as they are designed to deal with large sets of multidimensional data. Therefore, it seems to be also promisingly exploitable for feature selection tasks in sequence classification [24].

Finally, we observe that the implementation of this framework can largely benefit from run-time optimization for the verification of the rules’ automata, particularly for as far as the recognition of permanent violations and satisfactions is concerned [72]. The design and integration of such dedicated techniques serves as an impulse for future research.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors want to thank Arik Senderovich for the fruitful discussion and useful observations on this research. The work of Claudio Di Ciccio was partly supported by the Italian Ministry of University and Research (MUR) under grant “Dipartimenti di eccellenza 2018-2022” of the Department of Computer Science at Sapienza University of Rome and by the Sapienza SPECTRA research project, Italy.

The research by Jan Mendling was supported by the Einstein Foundation Berlin.

Appendix. Proof of linearity

The RCon evaluation can be performed efficiently based on the automaton-based technique defined in [14]. The verification is there defined for online settings (run-time analysis). In contrast, we can exploit the offline setting of the present contribution (post-mortem analysis) to enhance further the performances.

Before proceeding, we need to extend the LTLp notation presented in Section 3. In the following, we will classify ⊔, ⊓, ⊓, U as future operators, ⊔, ⊓, ⊑, S as past operators, and the following pairs of operators as mirror images: (i) ⊔ and ⊓, (ii) ⊓ and ⊔, (iii) and ⊑, (iv) U and S. A LTLp formula ϕ is named: pure past (ϕ*) if it contains only past operators; pure present (ϕ?) if it contains no temporal operators at all; pure future (ϕ?) if it contains only future operators [73]. For example, ϕ* = ϕ(a S g), ϕ? = a ∧ b ∨ c, and ϕ? = ⊓(≡ (∥ ∨ ⊔)) W p) are pure past, pure present, and pure future formulae, respectively. Finally, the mirror image ϕM of formula ϕ is the temporal formula obtained by replacing all its operators with their mirror images [74].

In [14] it has been proven that

- any LTLp formula ϕ can be decomposed in ϕ = ∨m ϕ*, where ϕ?, ϕ*, and ϕ? are respectively pure past, present, and future formulae [14, Definition 2];
- each sub-formula of the decomposed formula can be evaluated through distinct automata [14, Theorem 3];
- the past-only automata can be reversed into future-only automata valuating the reverse of a trace [14, Theorem 4].

The advantage of the reverse is that it allows for the verification of past components in only one read of the trace. This was not possible for future components because the end of the trace is unknown in an online setting. In an offline setting, as the entire trace is given, it is possible to apply the same reasoning to reverse future-only automata into past-only automata valuating a trace in one backward read from the end to the start of the trace.

In the following, we provide proof sketches of the intuitions above. We replicate the steps taken for past reversion [14] applying them for the future reversion. To this extent, we rely on mirror images and reversed automata.

Lemma A.1. Let t ∈ Σ* be a trace of length n and t̄ k its reverse.

Let t ∈ Σ* be a trace of length n and t̄ k its reverse.

Given a pure future formula ϕ*, and its mirror image ϕM, then t, 1 |= ϕ* iff t, n |= ϕM.

The proof follows from the semantics of future and past operators of LTLp provided in Section 3. For instance, verifying ϕ* = ∨ a on t = {t, e, d, c, b, a, g, h, i} at instant i = 1 is equivalent to verifying ϕ̄ M = a on t̄ k = {t, h, g, a, b, c, d, e, i} at i = 9. Notice that this holds for sub-traces too, thus verifying ϕ* on t at instant i = 5 is equivalent to verifying ϕM over t5:9GR = {h, g, a, b, i} at i = 9.

It follows from Lemma A.1 that any pure future formula can be seen as a pure past one on a reversed trace. Therefore the automaton verifying the mirror image of ϕ* can be used for verification on the reversed trace, as stated in the following.

Corollary A.1. Let Aϕϕ be the automaton verifying ϕM. Then t, n |= ϕ* iff t̄ k ∈ C (Aϕϕ).

Notice that ϕM is a pure past formula, therefore Aϕϕ can be built by applying the technique of [75]. Furthermore, it is possible to transform the obtained automaton in order to read directly the original trace t thanks to the property of closure under reversion of regular languages [76].

From Lemmas A.1 and A.1 we derive the following.

Theorem A.1 (Valuation through Aϕϕ). Let ϕ* be a pure future formula and ϕM its mirror image. Let Aϕϕ ∈ A be the automaton verifying ϕM. Given a trace t ∈ Σ* of length n, we have that:

t, 1 |= ϕ* iff t, n ∈ C (Aϕϕ).

Consider t = {t, e, d, c, b, a, g, h, i} and the RCon Ψ = a → (⊣ ∨ ⊔ ∨ ⊓) and the pure future formula of its separated automata set (sep.aut.set) ϕ* = ⊓. The RCon is activated by t(6). Its mirror image is ϕM = ⊓. It is possible to verify ϕM over trace t6:9GR = {h, g, a, b, i} as per Lemma A.1. Thanks to Theorem A.1, ϕ* can be verified on t6:9 with the reversed automaton Aϕϕ

In conclusion, we proved that any pure future formula can be verified by parsing sub-traces from the end of the trace till the activator event. As the verification of future formulae is the only potentially exponential time element in [14], using the reversed verification provided in this section guarantees the optimization to linear time, which has been also empirically proved in Section 5.

References


