A Temporal Logic-Based Measurement Framework for Process Mining

Alessio Cecconi  Giuseppe De Giacomo, Claudio Di Ciccio  Fabrizio Maria Maggi  Jan Mendling
WU Vienna  Sapienza University of Rome  Free University of Bolzano  WU Vienna
Vienna, Austria  Rome, Italy  Bolzano, Italy  Vienna, Austria
alessio.cecconi@wu.ac.at  {name.surname}@uniroma1.it  maggi@inf.unibz.it  jan.mendling@wu.ac.at

Abstract—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. The assessment is required to check how well a set of discovered rules describes the input data, as well as to determine to what extent data complies with predefined rules. In declarative process mining, in particular, some measures have been taken from association rule mining and adapted to support the assessment of temporal rules on event logs. Among them, support and confidence are used most often, yet they are reportedly unable to provide a sufficiently rich feedback to users and often cause spurious rules to be discovered from 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 general measurement framework for temporal rules based on Linear-time Temporal Logic with Past on Finite Traces (LTLp). The framework is independent from the rule-specification language of choice and allows users to define new measures. 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.

Index Terms—Declarative Process Mining, Specification Mining, Association Rule Mining, Quality Measures, Temporal Rules

I. 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. 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 fulfilled triggers of the rule over the entire log), 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.

A plethora of other measures are available in the context of association rule mining that are reportedly superior in comparison to support and confidence [6]. However, these measures do not take into consideration the temporal dimension, which is a first-class citizen dimension in process mining.

In this paper, we address this research problem by proposing a general measurement framework rooted in formal semantics, specifically in Linear-time Temporal Logic with Past on Finite Traces (LTLp), to express process rules in a reactive form [7] and abstract from specific rule-specification languages. More specifically, we show that a fine grained temporal logic interpretation of any formula in the form “if A then B” allows us to assess all the available association rule mining measures as-is for temporal rules.

Towards investigating the implications of using a large set of available measures on log analysis, we conduct an extensive set of simulation experiments. Through them, we observe that the measures respond differently to changes in the behavior evidenced by event logs, thus suggesting that different measures can be used to highlight different aspects of a process as per its recorded executions. Also, we test our implementation of the framework on both synthetic and real-world data.

The remainder of this paper is structured as follows. Section II discusses prior research on measures for declarative process mining and specification mining. Section III defines preliminaries upon which we define our framework. Section IV defines the measurement framework. Section V presents computational studies of our implementation and Section VI the results of its application to a series of simulation experiments. Finally, Section VII summarizes the contribution of this paper and points to opportunities for future research.

II. RELATED WORK

Temporal logic and declarative specifications have been widely used to support process discovery and conformance checking. The assessment of rules with respect to traces is a key component of all these techniques.

In declarative process discovery, quality measures are used to prune candidate rules based on user-defined thresholds. This pruning is used for DECLARE discovery in [3], [4] and for DCRgraphs discovery in [8]. These techniques are mainly based on confidence and support measures. Nevertheless, the use of support and confidence is not sufficient to avoid a large
number of spurious results, thus threatening their statistical validity [9]. In addition, the definitions of these metrics are also different for different techniques. For instance, the support measure presented in [3] is different from the support of [4]. Also, both are defined for the sole DECLARE template set.

In the area of conformance checking, de Leoni et al. [1] use alignments and fitness to measure the degree of conformance of a DECLARE model with respect to an execution trace. Polyvyanyy et al. [10] rely on entropy to measure precision and recall of both procedural and declarative models. Finally, Burattin et al. [11] use specific measures like fulfillment ratio and violation ratio based on the evaluation of the number of activations of a rule that lead to a fulfillment and the number of activations that lead to a violation in a log. Quality measures are critical for all these techniques. However, the metrics provided in these works are bound to the specific techniques and the precise modeling languages used. Also, those techniques do not provide a general measurement framework that can be applied for any type of temporal-logic rule.

In software engineering, a number of works employ techniques for the discovery and quality assessment of temporal patterns. This stream of research is called specification mining. Yang et al. [12] 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 \( R_{\text{ائن}} \) and \( R_{\text{حان}} \) in Table I) 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 [13]. 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 I). Furthermore, their computation of the probability for a temporal specification is based on a sliding window technique [14]: the 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 valid. 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. [15] extend specification mining to arbitrary Linear Temporal Logic (LTL) specifications (implicitly on finite traces) beyond 2-value templates. Yet, they resort only on support and confidence measures to prune uninteresting results, thus incurring in the already mentioned statistical limits [9].

In summary, we observe that measures are largely used as a tool to support research, but they are hardly made subject to evaluative research themselves. Next, we define preliminaries for a framework to assess various measures.

### III. Preliminaries

As the formal foundations of our framework, we consider rules specified in Linear Temporal Logic on Finite Traces (LTL\(_f\)) [16], as used in DECLARE [5], [17]. LTL\(_f\) has exactly the same syntax as LTL [18]. Its semantics is interpreted on finite traces, and hence takes into account that business processes are assumed to terminate sooner or later [19]. DECLARE focuses on some specific LTL\(_f\) formulas. Table I illustrates some of the most important rules for business process modeling.

Specifically useful for our purposes is LTL\(_p\), which is an extension of LTL\(_f\) supporting the expression of properties of the past (hence the “p” suffix) [7]. Well-formed LTL\(_p\) formulae are built from an alphabet \( \Sigma \supseteq \{ a \} \) of propositional symbols and are closed under the boolean connectives, the unary temporal operators \( \circ \) (next) and \( \ominus \) (previous), the binary temporal operators \( \mathbf{U} \) (Until) and \( \mathbf{S} \) (Since):

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

From these basic operators, it is possible to derive: classical boolean abbreviations True, False, \( \lor, \land \); constant \( t_{\text{find}} \), verified as \( \neg \circ \text{True} \), denoting the last instant of a trace; constant \( t_{\text{Start}} \), verified as \( \ominus \text{True} \), denoting the first instant of a trace; \( \diamond \varphi \) as True \( \mathbf{U} \) \( \varphi \) indicating that \( \varphi \) holds true eventually before \( t_{\text{find}} \); \( \varphi_1 \mathbf{W} \varphi_2 \) as \( (\varphi_1 \mathbf{U} \varphi_2) \lor \boxdot \varphi_1 \), which relaxes \( \mathbf{U} \) as \( \varphi_2 \) may never hold true; \( \hat{\varphi} \) as True \( \mathbf{S} \) \( \varphi \) indicating that \( \varphi \) holds true eventually in the past after \( t_{\text{Start}} \); \( \lozenge \varphi \) as \( \neg \hat{\varphi} \) indicating that \( \varphi \) holds true from the current instant till \( t_{\text{find}} \); \( \boxdot \varphi \) as \( \neg \lozenge \neg \varphi \) 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 LTL\(_p\) formula \( \varphi \) is satisfied in a given instant \( i \) (\( 1 \leq i \leq n \)) by induction of the following:

\[
t, i \models \text{True}; t, i \not\models \text{False};
\]

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

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

\[
t, i \models \varphi_1 \land \varphi_2 \text{ iff } t, i \models \varphi_1 \text{ and } t, i \models \varphi_2;
\]

\[
t, i \models \varphi_1 \lor \varphi_2 \text{ iff } t, i \not\models \varphi_1 \text{ or } t, i \not\models \varphi_2;
\]

\[
t, i \models \varphi_1 \mathbf{U} \varphi_2 \text{ iff } (t, i \not\models \varphi_1 \land t, i + 1 \models \varphi_1);
\]

\[
t, i \models \varphi_1 \mathbf{S} \varphi_2 \text{ iff } (t, i \models \varphi_1 \land t, i + 1 \models \varphi_2);\]

\[
t, i \models (\neg \varphi) \text{ iff } i < n \text{ and } t, i + 1 \models \varphi;
\]

\[
t, i \models \diamond \varphi \text{ iff } i > 1 \text{ and } t, i - 1 \not\models \varphi;
\]

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

\[
t, i \models \varphi_1 \mathbf{S} \varphi_2 \text{ iff } t, j \models \varphi_2 \text{ with } 1 \leq j < i, \text{ and } t, k \models \varphi_1 \text{ for all } k \text{ s.t. } j < k \leq i.
\]

A formula \( \varphi \) is satisfied by a trace \( t \), written \( t \models \varphi \) iff \( t, 1 \models \varphi \). One of the central properties of LTL\(_p\), shared with LTL\(_f\), is that one can compute a deterministic finite state automaton (DFS) \( A_\varphi \) such that for every trace \( t \) we have \( t \models \varphi \) iff \( t \) is in the language recognized by \( A_\varphi \).

<table>
<thead>
<tr>
<th>Constraint</th>
<th>LTL(_f) expression [16]</th>
<th>RCon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participation(a)</td>
<td>( a )</td>
<td>( \forall a \text{ s.t. } a \models a )</td>
</tr>
<tr>
<td>Init(a)</td>
<td>( a )</td>
<td>( \exists a \text{ s.t. } \neg a \models a )</td>
</tr>
<tr>
<td>End(a)</td>
<td>( \neg a )</td>
<td>( \exists a \text{ s.t. } \neg a \models a )</td>
</tr>
<tr>
<td>AtMostOne(a)</td>
<td>( \exists x \models (\neg a) )</td>
<td>( \forall a \text{ s.t. } \neg a \models a )</td>
</tr>
<tr>
<td>RespondedExistence(a,b)</td>
<td>( a \models b )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
<tr>
<td>Response(a,b)</td>
<td>( \exists x \models (a \rightarrow b) )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
<tr>
<td>AlternateResponse(a,b)</td>
<td>( \neg (a \rightarrow b) \times (a \rightarrow \neg (\neg b \rightarrow \neg b)) )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
<tr>
<td>ChainResponse(a,b)</td>
<td>( \neg (a \rightarrow b) \times (a \rightarrow \neg (b \rightarrow b)) )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
<tr>
<td>Precedence(a,b)</td>
<td>( b \models a )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
<tr>
<td>AlternatePrecedence(a,b)</td>
<td>( \neg (b \models a) \times (b \models \neg (b \models b) \models a) )</td>
<td>( \forall a, b \text{ s.t. } \neg a \models b )</td>
</tr>
</tbody>
</table>
IV. Temporal-Extended Measurement Framework

In this section, we build on LTLp_j to develop our measurement framework. Above, we discussed that relying only on confidence and support measures can produce spurious results [9] and that various measures beyond them have been proposed in association rule mining for time-unaware rules [13]. Table II presents a comprehensive list of those measures. Some of them, like confidence and support, are used for the temporal specification of processes, though interpreted in different, ad-hoc ways [3], [4], [7]. The framework we propose next is generic as it allows for the usage of any probabilistic measure (including those of Table II) on any rule for the temporal specification of processes. To this end, Section IV-A formalizes the reactive temporal specification of rules, Section IV-B discusses their probabilistic interpretation, and Section IV-C defines the overall framework.

A. Reactive Temporal Specification

Our first building block is the concept of Reactive Constraint (RCon), originally introduced in [7], which we extend here. A rule typically expresses that the occurrence of certain preconditions (activation) implies certain consequences (target). We codify such intuition in RCons based on LTLp_j.

Definition 4.1 (Reactive Constraint (RCon)): Given an alphabet Σ ∪ {f_start, f_end, True, False}, let ϕ_α and ϕ_τ be LTLp_j formulae over Σ. A Reactive Constraint (RCon) Ψ is a pair (ϕ_α, ϕ_τ) hereafter denoted as Ψ = ϕ_α ⇒ ϕ_τ.

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 → ◦b is an RCon describing that every time a (the activator, ϕ_α) is True, then also ◦b (the target, ϕ_τ) must evaluate to True. 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 = ϕ_α ⇒ ϕ_τ and a trace t of length n ∈ N, let t_i be the i-th event in the trace (1 ≤ i ≤ n). For each t_i in t the possible evaluations of Ψ are:

ϕ_α = False, ϕ_τ = False if {t, i} ∪ ϕ_α ∧ t_i ∪ ϕ_τ;  
ϕ_α = False, ϕ_τ = True if {t, i} ∪ ϕ_α ∧ t_i ∪ ϕ_τ;  
ϕ_α = True, ϕ_τ = False if {t, i} ∪ ϕ_α ∧ t_i ∪ ϕ_τ;  
ϕ_α = True, ϕ_τ = True if {t, i} ∪ ϕ_α ∧ t_i ∪ ϕ_τ.

For example, the first two rows of Table III show the evaluation of RCon (ϕ b ∧ ◦e) ⇒ (¬c ∨ ◦f) in each event of trace ⟨a, b, c, d, f, e, c, e, c, h⟩. The constraint states that, between the execution of tasks b and e, no occurrence of c is expected unless it is eventually followed by f. Notice that ϕ_α and ϕ_τ are evaluated separately at every event of a trace.

The RCon evaluation can be performed efficiently based on the automaton-based techniques defined in [7], adapting it for offline verification. The full description of this aspect goes beyond the scope of the paper, but we briefly outline the rationale here. Intuitively, we resort on [7, Theorem 4]: an RCon can be separated in pure-past, pure-present and pure-future components. The respective sub-formulas contain only past temporal operators, none, or only future ones. Therefore, by mirroring pure-past formulas 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 know the suffix of the trace, we can apply the same principle to pure-future formulas 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. In this way, we evaluate any formula at each event reading the trace only twice: once from t_start to t_end (past components) and once from t_end to t_start (future components). This implies that the computational cost depends linearly on the number of events in the event log and the number of rules to verify. Specifically, given an event log L with |L| traces, assuming every trace t ∈ L having length up to n events, and M rules, the cost to verify all rules on L is: O(|L| × n × M).

B. Probabilistic interpretation

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 such occasions wherein the trace contains 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 ϕ_α and ϕ_τ on single events (Def. 4.2), we can devise a probabilistic interpretation for RCons over traces.

Definition 4.3 (Probability of an LTLp_j formula): Given an LTLp_j formula ϕ and a trace t of length |t| = n, we define the probability of ϕ over t = ⟨t_1, . . . , t_n⟩ as the proportion of events t_i (i ∈ [1, n], n ∈ N) satisfying ϕ.

\[ P(ϕ, t) = \frac{|\{i \in [1, n] : t_i \models ϕ\}|}{n} \]

Definition 4.4 (Probability of LTLp_j formulae intersection): Given two LTLp_j formulae ϕ_1 and ϕ_2 and a trace t of length n, we define the probability of the intersection of ϕ_1 and ϕ_2 over t as the proportion of events t_i ∈ t satisfying both ϕ_1 and ϕ_2.

\[ P(ϕ_1 \cap ϕ_2, t) = \frac{|\{i \in [1, n] : t_i \models ϕ_1 \land t_i \models ϕ_2\}|}{n} \]

The probability of all the RCon evaluations follows from the above definitions.

\[
\begin{align*}
P(¬ϕ_α \cap ¬ϕ_τ, t) &= \frac{|\{i \in [1, n] : t_i \not\models ϕ_α \land t_i \not\models ϕ_τ\}|}{n}
\end{align*}
\]

\[
\begin{align*}
P(¬ϕ_α \cap ϕ_τ, t) &= \frac{|\{i \in [1, n] : t_i \not\models ϕ_α \land t_i \models ϕ_τ\}|}{n}
\end{align*}
\]

\[
\begin{align*}
P(ϕ_α \cap ¬ϕ_τ, t) &= \frac{|\{i \in [1, n] : t_i \models ϕ_α \land t_i \not\models ϕ_τ\}|}{n}
\end{align*}
\]

\[
\begin{align*}
P(ϕ_α \cap ϕ_τ, t) &= \frac{|\{i \in [1, n] : t_i \models ϕ_α \land t_i \models ϕ_τ\}|}{n}
\end{align*}
\]

For example, Table III shows the probabilities resulting form the evaluation of RCon (ϕ b ∧ ◦e) ⇒ (¬c ∨ ◦f) on trace ⟨a, b, c, d, f, e, c, e, c, h⟩. Probabilities defined as above permit the application, in the context of temporal logic specifications, of measures defined for association rule mining [13], where an
Table II: Probabilistic measures from [13] on association rules in the form “if A then B”.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
<th>Range</th>
<th>Measure</th>
<th>Formula</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support</td>
<td>$P(A \land B)$</td>
<td>[0, 1]</td>
<td>Shrink vertex</td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\alpha$</td>
<td>[0, +\infty]</td>
</tr>
<tr>
<td>Coverage</td>
<td>$P(A)$</td>
<td>[0, 1]</td>
<td>Weighting</td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\beta$</td>
<td>[0, +\infty]</td>
</tr>
<tr>
<td>Precision</td>
<td>$P(B</td>
<td>A)$</td>
<td>[0, 1]</td>
<td>Yule’s Q</td>
<td>$\frac{P(A \land B)}{P(A)}$</td>
</tr>
<tr>
<td>Recall</td>
<td>$P(B</td>
<td>A)$</td>
<td>[0, 1]</td>
<td>Klosgen</td>
<td>$\sqrt{P(A</td>
</tr>
<tr>
<td>Specificity</td>
<td>$P(A</td>
<td>B) = \frac{P(A \land B)}{P(B)}$</td>
<td>[0, 1]</td>
<td>Gain Index</td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\alpha + P(A \rightarrow B)^\alpha\cdot P(B)^\beta + P(A \rightarrow B)^\alpha\cdot P(B)^\gamma$</td>
</tr>
<tr>
<td>Accuracy</td>
<td>$P(B</td>
<td>A)$</td>
<td>[0, 1]</td>
<td>Collective Benefit</td>
<td>$\frac{P(A \land B)}{P(A)} - P(A</td>
</tr>
<tr>
<td>LiftFactor</td>
<td>$P(A</td>
<td>B) \cdot P(B</td>
<td>A)$</td>
<td>[0, +\infty]</td>
<td>Laplace Correction</td>
</tr>
<tr>
<td>Event level:</td>
<td>$P(A</td>
<td>B)$</td>
<td>[0, 1]</td>
<td>$\alpha$ Measure</td>
<td>$P(A</td>
</tr>
<tr>
<td></td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\beta$</td>
<td>[0, +\infty]</td>
<td>Two-Way Support</td>
<td>$P(A</td>
<td>B) \cdot \log \frac{P(A</td>
</tr>
<tr>
<td></td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\beta$</td>
<td>[0, +\infty]</td>
<td>Variation</td>
<td>$\frac{P(A</td>
<td>B) \cdot \log \frac{P(A</td>
</tr>
<tr>
<td></td>
<td>$P(A \rightarrow B)^\alpha\cdot P(B)^\beta$</td>
<td>[0, +\infty]</td>
<td>Zheng</td>
<td>$\frac{P(A</td>
<td>B) \cdot \log \frac{P(A</td>
</tr>
</tbody>
</table>

Table III: Evaluation (0 is False and 1 is True), probabilistic interpretation, and statistics computation of a sample of measures for RCon ($\hat{\Phi}b \land \hat{\Phi}e \Rightarrow (\neg c \lor \hat{\Phi}f)$).

![Figure 1: Measurement framework pipeline.](image)

**Log level**: statistical distribution of every measure in M for every constraint in R.

For example, Table III shows the aggregation of trace-level measures for RCon ($\hat{\Phi}b \land \hat{\Phi}e \Rightarrow (\neg c \lor \hat{\Phi}f)$) in a log that consists of 4 traces. Being able to perceive the overall status of a constraint is as important as the possibility to analyze its details in single traces. Therefore, we report the entire statistical distribution of a measure across the log to provide a complete information spectrum. Figure 1 depicts the pipeline of the framework from the input to the output. In the first stage, an RCon is evaluated in each trace of the log. Then, the evaluation result is used to compute probabilities and measures of the rule in each trace. In the final stage, the statistical distribution of each measure over the log is drawn.

The design of the RCons, in particular the choice of the activators, is crucial for the evaluation and the computation of the measures. Let us take as an example the constraint RESPONDEDEXISTENCE(a,b) from the repertoire of DECLARE (see Table I). The classical LTL formula underlying RESPONDEDEXISTENCE(a,b) for whole-trace evaluations is $\neg \langle a \land \langle b \rangle \rangle$ [16]. However, the formulation of the rule as an RCon can lead to different interpretations:

- $a \Rightarrow (\langle b \lor \hat{\Phi}b \rangle)$, i.e., when a occurs, b is expected to occur somewhere in the trace;
- $\langle a \lor \hat{\Phi}a \rangle \Rightarrow (\langle b \lor \hat{\Phi}b \rangle)$, i.e., for every event of the trace.

**C. Measurement system**

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

- **Event level**: distinct evaluation of $\varphi_A$ and $\varphi_B$ for every constraint in R on each event of each trace in L;
- **Trace level**: measurement of every measure in M for every trace in L for every constraint in R;
- **Event log level**: statistical distribution of every measure in M for every constraint in R.

antecedent A and a consequent B are defined for every rule in the form “if A then B”. To that extent, it suffices to map $\varphi_A$ to A and $\varphi_B$ to B, thus having $P(A)$ as $P(\varphi_A)$, $P(B)$ as $P(\varphi_B)$, and $P(AB)$ as $P(\varphi_A \land \varphi_B)$. As a result, we can extend any measure defined for association rules to temporal rules “if A then B”.

Table II shows few measures computed from the probabilistic measures of RCon ($\hat{\Phi}b \land \hat{\Phi}e \Rightarrow (\neg c \lor \hat{\Phi}f)$).
such that a occurs either in the past or in the future, also b should occur somewhere in the trace;

• True $\Rightarrow \neg (\Diamond a \lor \Diamond b) \lor (\Box b \lor \Box \Diamond b)$, i.e., at any event, if a occurs in the trace, also b is expected to occur;

• $t_{\text{Start}} \Rightarrow \neg (\Diamond \Diamond a \lor \Diamond b)$, i.e., 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 single 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?

In summary, we have defined a measurement framework for declarative specifications defined as Reactive Constraints, capable of reporting customizable measures at both trace and event log levels.

V. IMPLEMENTATION AND PERFORMANCE ANALYSIS

We have implemented our measurement framework as a proof-of-concept software prototype on top of an existing declarative process specification discovery tool [7]. The Java source-code can be found at github.com/Oneiroe/Janus. The core software architecture for the verification of RCons is shared with the discovery tool. That is why the new and the old software components are contained in the same repository despite the independence of the two modules (discovery and measurement). All the models used in the following experiments are discovered with [7].

In the remainder of this section, we report on the results of an experimental investigation on the computational performance of our implemented framework. In particular, we assess the performance of the implemented technique against an increase in the data size (i.e., the size of the event log) and the model size (i.e., the number of rules) with synthetic event logs. Finally, we test the performance on a set of real life 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 2.60GHz, quad-core, 16Gb of RAM and an Ubuntu 18.04.4 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 generated logs of varying sizes that are compliant with a fixed set of rules, resorting on the simulation engine of MINERful [20]. Thereupon, we computed the measures listed in Table II against all the rules of a larger test model (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 V. Notice that the set contains all the rule templates seen in Table I 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] that is compliant with the initial model.\footnote{Available at github.com/Oneiroe/Janus/blob/master/tests-SJ2T/model.zip} Given the test model obtained as described above, we performed two tests of 65 iterations each, based on synthetic event logs that comply with the rules of Table V, by: (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); (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). Figure 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 IV-A.

Next, we investigate the response of the framework to an increase in the model size. To do so, we first generated an event log containing 1000 traces with a trace length between 100 and 500 events from the simulation of the rules in Table V. Thereupon, we used the discovery algorithm of Janus to automatically retrieve different test models with varying levels of compliance. To that extent, we made the confidence threshold range from 1.0 (full model compliance), down to 0.0 with a step

---

| RCon formulation | Evaluation | Support $P(\varphi_a \land \varphi_r)$ | Confidence $P(\varphi_r | \varphi_a)$ |
|------------------|------------|------------------------------------|----------------------------------|
| $a \Rightarrow (\Diamond b \lor \Box b)$ | $\varphi_a: (0, 1, 0, 0, 1)$ | $\varphi_r: (1, 1, 1, 1, 1)$ | $\gamma = 0.4$ | $\gamma = 1$ |
| $(\Diamond a \lor \Box a) \Rightarrow (\Box b \lor \Diamond b)$ | $\varphi_a: (1, 1, 1, 1, 1)$ | $\varphi_r: (1, 1, 1, 1, 1)$ | $\gamma = 1$ | $\gamma = 1$ |

| True $\Rightarrow (\Diamond a \lor \Box b)$ | $\varphi_a: (1, 1, 1, 1, 1)$ | $\varphi_r: (1, 1, 1, 1, 1)$ | $\gamma = 1$ | $\gamma = 1$ |

| $t_{\text{Start}} \Rightarrow (\neg \Diamond a \lor \Diamond b)$ | $\varphi_a: (1, 0, 0, 0, 0)$ | $\varphi_r: (1, 1, 0, 0, 0)$ | $\gamma = 0.2$ | $\gamma = 1$ |

---

<table>
<thead>
<tr>
<th>Table V: The set of DECLARE rules used in the experiments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT(a)</td>
</tr>
<tr>
<td>End(b)</td>
</tr>
<tr>
<td>ATMOSTONE(c)</td>
</tr>
<tr>
<td>PARTICIPATION(d)</td>
</tr>
</tbody>
</table>

---

Table IV: Measurements of a constraint expressed with different formulations on trace $\langle d, a, b, c, a \rangle$. |
We tuned the parameters of the discovery algorithm to obtain a set of rules that are highly compliant (confidence threshold of 0.05). The rationale is, the lower the confidence threshold, the higher the number of constraints in the test model. Then, we calculated all the measures in Table II 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 total number of rules to check, thus in line with the theoretical computational cost exposed in Section IV-A.

To test the performance also in real settings, we calculated the measurements on 11 openly available BPICs event logs\(^2\) plus one event log stemming from a partner of a smart-city project in which the authors are involved (labelled as “Smart city” in Table VI). We included the latter 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 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 that are highly compliant (confidence threshold of 0.8), even if not frequent (support threshold of 0.05). Table VI illustrates the results. For each log we report, along with the number of traces, tasks and events contained in the log and the number of rules in the test model, the total time from the launch to the termination of the software (“Total”), the time to evaluate the rules on every event (“Checks”), and the time to compute the measures aggregated by trace and log (“Measures”). We remark that the wall-clock time remains within acceptable ranges as the slowest run takes around 1.5 minutes to check about 500 constraints.

### VI. Evaluation

In this section, we report on experiments that show interesting implications of having a vast availability of measures with customization options. We argue that having different measures yields a more precise characterization of the behavior at hand. Indeed, different measures respond differently to different stimuli in the data. To support this claim, we study the variation in the measures at the injection of specific noise types in the event log that directly or indirectly affect a constraint, thereby assessing their sensitivity or resilience to changes. In fact, while an observable reaction of measures could be desirable when, e.g., noise reveals a change in the process execution, resilience could be preferable, for example, when noise stems from a mere interference. Thereupon, we injected noise in the event log, one constraint at a time, thus diminishing the effect of cross-interference. Indeed, different measures respond differently to different stimuli in the data. To support this claim, we study the variation in the measures at the injection of specific noise types in the event log, even if not frequent (support threshold of 0.05).

#### Table VI: Performance records on real-life datasets.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BPIC12</td>
<td>13087</td>
<td>56</td>
<td>262200</td>
<td>519</td>
<td>94.7</td>
<td>20 595.40</td>
<td>70 414.90</td>
</tr>
<tr>
<td>BPIC13_x</td>
<td>1487</td>
<td>7</td>
<td>6660</td>
<td>20</td>
<td>1.3</td>
<td>129.3</td>
<td>274.9</td>
</tr>
<tr>
<td>BPIC13_y</td>
<td>7554</td>
<td>13</td>
<td>65533</td>
<td>14</td>
<td>2.6</td>
<td>389.5</td>
<td>666.5</td>
</tr>
<tr>
<td>BPIC14_x</td>
<td>41533</td>
<td>9</td>
<td>369485</td>
<td>51</td>
<td>20.6</td>
<td>3871.70</td>
<td>13 477.10</td>
</tr>
<tr>
<td>BPIC14_y</td>
<td>902</td>
<td>70</td>
<td>21656</td>
<td>3856</td>
<td>37.9</td>
<td>13 197.70</td>
<td>22 796.00</td>
</tr>
<tr>
<td>BPIC15_x1</td>
<td>681</td>
<td>82</td>
<td>24678</td>
<td>5809</td>
<td>19 139.90</td>
<td>35 130.60</td>
<td>45 503.70</td>
</tr>
<tr>
<td>BPIC15_x2</td>
<td>1369</td>
<td>62</td>
<td>47786</td>
<td>4098</td>
<td>72.3</td>
<td>24 449.00</td>
<td>45 503.70</td>
</tr>
<tr>
<td>BPIC15_x</td>
<td>860</td>
<td>65</td>
<td>29403</td>
<td>4690</td>
<td>55</td>
<td>18 605.10</td>
<td>44 459.80</td>
</tr>
<tr>
<td>BPIC15_y1</td>
<td>975</td>
<td>74</td>
<td>30300</td>
<td>5164</td>
<td>62.9</td>
<td>21 975.60</td>
<td>39 039.20</td>
</tr>
<tr>
<td>RTFMP</td>
<td>15070</td>
<td>11</td>
<td>561470</td>
<td>49</td>
<td>62</td>
<td>12 666.30</td>
<td>43 145.40</td>
</tr>
<tr>
<td>SEPSIS</td>
<td>1050</td>
<td>16</td>
<td>15214</td>
<td>260</td>
<td>3.1</td>
<td>710.70</td>
<td>1500.70</td>
</tr>
<tr>
<td>Smart city</td>
<td>4347</td>
<td>20</td>
<td>692333</td>
<td>292</td>
<td>67</td>
<td>22 226.60</td>
<td>39 295.50</td>
</tr>
</tbody>
</table>

\(^2\)https://data.4tu.nl/
activator and target via insertions and deletions, and (2) the indirect effect, by altering the occurrences of the other tasks in the log with white noise. We made the noise spread all over the 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 log, picked at random.

More specifically, for every rule in the set of Table V, 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.

Figure 4 shows the results of such experiments on constraint \( \text{RESPONSE}(a,t) \). Every line corresponds to a measure. The \( \text{RESPONSE} \) constraint imposes that the target occurs eventually after each occurrence of the activator. The plots reveal through the starting point and the steepness of the slopes in the curves whether, and to what extent, the corresponding measures consider the frequency of the events satisfying the activator or the target in the traces. As it can be seen, the measures show different trends for each stimulus.

At large, the \( \text{RESPONSE} \) constraint appears to be particularly sensitive to the deletions of the target and influenced by both the deletions and insertions of the activator, while mostly insensitive to spurious insertions of the target and white noise. More specifically, we can derive the following observations.

- The deletion of events satisfying the target leads to more violations of the rule (higher \( P(\varphi_a \neg \varphi_t) \)), thus the negative effect is reflected in the drop of many measures based on \( P(\varphi_a \neg \varphi_t) \) (e.g., confidence, lift, certainty factor). In contrast, the curves of specificity and accuracy grow because of their definition bound to \( P(\neg \varphi_a \varphi_t) \).

- The deletion of events satisfying the activator, instead, does not bring more rule violations, but only less satisfactions (lower \( P(\varphi_a \varphi_t) \)). Therefore, measures such as relative risk, lift, Zhang, or leverage, decrease. Measures focusing on the target (e.g., prevalence) are basically unaffected.

- The insertion of more events satisfying the target (higher \( P(\varphi_t) \)) does not influence the frequency with which the activator is satisfied. Most of the measures are stable, with a slightly decreasing trend for, e.g., lift or accuracy.

- The insertion of more events satisfying the activator is less characterizing, as the new elements may bring both new rule satisfactions (higher \( P(\varphi_a \varphi_t) \)) and violations (higher \( P(\varphi_a \neg \varphi_t) \)). That is why many measures show an increasing trend (e.g., cosine or Laplace), while others oscillate around the initial value (e.g., confidence). Even the most visible downward trends (see certainty factor, Zhang and Yule’s Y and Q) do not show a totally smooth trend.

- Lastly, the constraint is mostly stable against random alterations that affect neither the activator nor the target. The satisfactions and violations remain constant, whilst the only increase is in \( P(\neg \varphi_a \varphi_t) \) and \( P(\neg \varphi_a \neg \varphi_t) \). The slight fluctuations of the measures are due to the variation in the number of events in the traces.

Because of space limits, we cannot illustrate the results for the other rules of Table V. The interested reader can find them at github.com/Oneiroe/Janus/blob/master/tests-SJ2T/NOISE-INJECTION-PlOTS.zip. Overall, generalizing from this specific case, we observe the following: (1) The reaction of the measures to noise depends on the type of stimulus and on whether it affects the target or the activator of the rule. (2) Not all measures sense certain alterations. If a measure is mostly stable with and without an error, it means that it cannot sense that particular stimulus. (3) Some measures have similar trends, but different magnitude. This means that there are “classes” of measures that focus on the same aspects of a rule. (4) The steepness of the curves with which the measure evolves indicates how much the measure is resilient to the change in presence of noise. In particular, it shows the range of tolerance before the error becomes too large to recognize the specific constraint behavior. Notice that sometimes it 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).

VII. 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. 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.

**Future work.** Different possibilities are now open upon the foundations of the measurement framework. It is possible to exploit the possibility to characterize a phenomenon by studying the evolution of different measures for, e.g., dynamic recognition of exceptions in process monitoring [22] or the identification of process drifts [23].

As different measures react differently to different stimuli for different types of rules, a method to select and combine the most appropriate measures depending on the context turns out to be key. To this extent, future research could resort to existing techniques like [24] 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], [11].

Many specifications can be considered equivalently valid for static measurements, yet their evolution can show which of them is the most error-tolerant. In this sense, the framework can support the identification of resilient specifications.

While the analysis of multiple measures at once may be overwhelming for a human, machine learning techniques, dealing naturally with multidimensional data, could benefit
from the availability of the great amount of information returned by the proposed framework. Therefore, it seems to be also promisingly exploitable for feature selection tasks in sequence classification [25].

ACKNOWLEDGMENT

The work of C. Di Ciccio was partly supported by MIUR under grant “Dipartimenti di eccellenza 2018-2022” of the Department of Computer Science at Sapienza University of Rome.

REFERENCES