A Tree-Based Definition of Business Process Conformance

Sylvain Hallé

Laboratoire d'informatique formelle
Université du Québec à Chicoutimi, Canada

Abstract. Conformance checking is a process that typically produces a binary pass/fail verdict. Yet, there exist situations where it is desirable to qualify the extent to which the execution of a process satisfies or violates a given condition. To this end, the paper proposes a relation on event traces that compares the tree resulting from the evaluation of a conformance condition on each of them. This relation neither requires Boolean conditions to be rewritten or adapted, nor expects additional information (such as weights) from the user.

1 Introduction

Conformance checking refers to the task of assessing whether the execution of an information system satisfies a set of conditions stipulating its expected behavior [17]; the process has at times also been labelled as compliance [12,22]. Under different names, this concept can be found in a variety of situations; for instance, it can be used to determine if a business process log follows business rules [3, 7, 29, 30], to check that the execution of a computer program is exempt from bugs or security policy violations [5], or to verify that a web service is used according to a specified protocol [10, 20, 33]. To a large extent, even the symbolic pattern matching used by some types of intrusion detection systems can be seen as a form of (anti-) conformance checking [36].

In its simplest form, the result produced by a conformance checking procedure is a Boolean verdict. This all-or-nothing behavior has long been identified as a limitation to the usefulness of such techniques in practice, due to the scarce information provided to a user as to the cause of a violation, its location in the execution of the system, and possibly the severity or impact of this violation on the global operation of the system. Consequently, various approaches have been proposed to complement or replace a true/false verdict with additional elements aimed at facilitating diagnostics: aligning an execution to a process model to identify points of discrepancy [37], identifying subsets of an execution or a property that explain a violation [14, 18], or replacing a conformance condition with more general queries on a process log [34].

However, there exist situations where a user may not be interested in locating the source of an error, but rather to assess to what extent a condition has been satisfied or violated. For instance, a process that inverts the expected order of a few operations a few times should be given lower troubleshooting priority than another that regularly performs operations out of order. In Section 2, we shall see that a few approaches have been put forward to replace a two-valued verdict with a finer-grained scale to which executions of

a system can be mapped, and thus ordered; for most of them, that scale is a closed interval of real numbers. However, these techniques require one to explicitly define counters and fine-tune user-defined thresholds, and the numerical result can be likened to a form of "percentage of violation", but its exact semantics is typically difficult to grasp.

After Section 3 provides a formal model for expressing conditions on event sequences, Section 4 proposes an alternate approach to conformance based on a different premise. Instead of attempting to assign a (seldom meaningful) number to each run of a system, it suggests the use of a relation that simply compares two executions, stating whether one of the two is preferable to the other. It does so by comparing tree structures resulting from the evaluation of a condition written in LTL.

This evaluation procedure has been implemented into a tool that is described in Section 5. Experiments show that the proposed relation can efficiently compare executions in logs and on constraints extracted from real-world scenarios. This opens the way to multiple applications and extensions of this principle, which are discussed in Section 6.

2 The Need for Finer-Grained Conformance Verdicts

The execution of a system or process can be symbolically represented by a sequence (also called a trace) of data elements representing different operations or actions occurring at different times. A conformance property is a condition on such an execution that is answered with true or false: either the sequence is conforming, or it is not. For example, a statement such as "every received message must be handled with a reply within 24 hours" admits only two verdicts, since a message is either dealt with on time or not, even if only by one minute. Such conditions, formulated in this way, do not allow for a "gray area". In this section, we first illustrate situations where a notion of "partial satisfaction" would be appropriate, and then survey the various works that attempted to tackle this problem.

2.1 Motivation

It is important from the outset to clarify what is meant by the notion of "degrees" of satisfaction or violation of a property, which can be confused with other situations. For example, one might think that a condition like "every received message must be replied to within 24 hours, with a maximum of one exception" admits multiple degrees of satisfaction. However, this is not the case: it is indeed a property that adds one exception to the rule stated above, but we remain in the Boolean domain —either at least one message is replied to late, or not. There are no more degrees of satisfaction of this condition than in the previous case.

The same applies to Service Level Agreements (SLAs), whose terms are often expressed in the form of a series of levels describing the quality of service promised to a consumer. Thus, one could define Tier 1 of a web hosting service as promising 99.9% uptime, while Tiers 2 and 3 would provide 99% and 90%, respectively. However, what appears to be degrees of satisfaction of a condition is in reality simulated by the overlap of a chain of conditions, each being Boolean in nature (either the service reaches the announced threshold or not) and logically implying the next. What we seek to do is

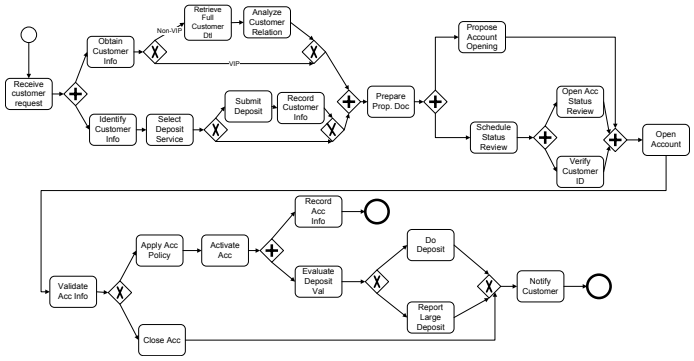


Fig. 1: A BPMN diagram for a banking process; figure reproduced from [2].

the opposite: state a *single* condition, which at any moment is either true or false, but somehow manage to distinguish multiple degrees of satisfaction or violation.

As a running example, we consider the business process model taken from [26], illustrated in Figure 1. It describes the process of creating a new account for a customer and making an initial deposit into it, following the banking regulations imposed by the People's Bank of China. The process has been used in past works on business process conformance (e.g. [2, 22]) and is subject to a wide array of potential constraints; however for the purpose of this paper, we shall focus on four such constraints, listed below:

1. No account must be open without first obtaining and verifying customer information.
2. For VIP customers, an account should be open at most one week after the request.
3. A manager must perform either the preparation of the documentation, the opening of the account or the validation of the account.
4. A deposit must be immediately preceded by an evaluation of its amount.

An execution of the process violates Condition 1 either because it is missing the *obtain* action or the *verify* action. However, one could argue that an execution where both of these activities are absent is an even greater violation of the condition. Thus, if evaluating Condition 1 on the former produces the "false" outcome, evaluating it on the the latter should produce an even "falser" value. This highlights the need for more than one verdict indicating the violation of a condition.

On its side, Condition 2 is violated when a VIP customer is not given an account in the expected delay. Yet, missing the delay by one hour is probably less detrimental than exceeding it by one week. Here again, multiple levels of violation would allow making such a distinction. Conversely, the condition is satisfied if the account if created on time; however, one could argue that the quicker the account is created, the better; a client getting their account in one day shows a higher quality of service than if their request is being answered at the last possible moment.

Therefore, in addition to multiple degrees of violation, it is also desirable to distinguish between multiple executions that satisfy a condition. The same idea is conveyed by Condition 3, which ensures that at least one of three key activities is conducted by a senior staff member. An execution of the process where more than one of these activities is

handled by a manager could be seen as exceeding expectations, and thus be ranked higher than an instance where only the bare minimum is done to comply with the regulation.

Finally, Figure 1 reveals that Condition 4 can be satisfied in two different ways: either *substantially* by evaluating and then allowing a deposit, or *vacuously* if no deposit ever takes place. However, it can be argued that without additional context, neither is a preferable way of satisfying the condition, although they are essentially distinct. Thus, each should be associated with positive verdicts, but without these verdicts necessarily being comparable. The same can be said, at a higher level, of two non-conformant instances of the process violating a different condition from the list. It would be ill-advised to arbitrarily declare one violation to be worse than another —which highlights the need for uncomparable negative verdicts as well.

2.2 Related Work

Early work has focused on the study of *similarity metrics* between event sequences [13,31]. In this context, two sequences of events are more or less "similar" depending on the amount of edit operations that need to be applied in order to turn one sequence into the other. However, similarity is blind to any notion of correctness with respect to a condition; it is only possible to tell how far apart are two execution sequences. Moreover, events are considered as atomic symbols, so differences in parameters are not taken into account (except for the case of timestamps in [31]).

The degree to which a trace satisfies a specific process description can first be quantified by finding an *alignment*, which can be summarized as a mapping between contiguous events of a trace and contiguous transitions in a process model [37]. Discontinuities in an alignment (e.g. swapping, inserting or deleting an event) indicate a discrepancy between the trace and the model, and such discontinuities can be counted and used as numerical measure of deviation. A similar approach consists of calculating the distance between a target event trace with a set traces used as a reference [27].

On their side, *grey security policies* are conditions on a sequence of events that produce a real value in the unit interval [35]. For example, a condition that states that every file that is open must eventually be closed would assign a score to a trace depending on the number of open files that have not been closed. However, conditions are expressed in an *ad hoc* manner using user-defined counters or other devices providing a numerical outcome, which are specific to each situation. TK-LTL attempts to resolve the issue by providing an extension of Linear Temporal Logic that allows users to write assertions on the number of times a given condition is satisfied [23].

Similarly, fuzzy LTL is an extension of LTL where the truth value of a proposition is replaced by a real number between 0 and 1 [25]. However, non-Boolean verdict is only obtained if the ground terms of an expression take a fractional value; therefore, its measure represents imprecise observations, but not partial satisfaction. Another real-valued semantics of LTL is proposed [4], in which, for example, a temporal operator stating that a condition is always true is relaxed so that it can be false "a few times" without compromising satisfaction. One can also consider the conformance of an execution across multiple dimensions (e.g. time, ordering, cost) and evaluate a user-defined metric on the unit interval on each dimension; those values can then be aggregated to obtain a genral conformance score [24].

When considering properties admitting compensation (such as the case of deontic logic) one can distinguish between executions that are *ideal* (the main condition is fulfilled), *sub-ideal* (the condition is violated, but the stipulated compensation was duly executed) and *non-ideal* (a mandatory compensation was not executed) [28]. The count of possible executions in a business process model belonging to each category can form the basis of a numerical compliance metric, again in $[0, 1]$.

Finally, *informativeness* is a measure that is associated to a trace depending on the number of "non-essential events" it contains, i.e. events that can be removed from a trace without compromising conformance [6]. It was introduced in the context of process mining, in order to quantify the extent to which an execution trace reveals information about optional paths in a process model. However, it presents the downside of only applying to conforming traces, and requires the user to manually assign weights to each event.

2.3 Limitations of Existing Approaches

All these approaches provide a way to quantify, or at the very least, allow some degree of comparison between multiple event traces, and therefore offer a verdict that is arguably more detailed than a simple pass/fail response. That said, they come with several limitations, which we detail below.

With the exception of informativeness, the aforementioned approaches do not distinguish between successful executions. Indeed, a conforming trace has no discontinuity when searching for an alignment with a process model, and logic-based formalisms producing a numerical truth value map all satisfying traces to 1. Yet, we have seen in the examples above that there is value in distinguishing a process that barely conforms with a regulation from another one that largely fulfills expectations. Moreover, many of these approaches consider events as atomic symbols, or otherwise only focus on the order in which the events are observed. This does not allow for measuring deviations from other types of conditions, such as those related to the values of parameters that these events might contain.

In some cases, a condition that was originally expressed as a true/false assertion must be rewritten by the user so that a more detailed verdict can be returned. For example, in the context of TK-LTL, it is indeed possible to relax the condition "every opened file must be closed" to "there are no more than 1% of files that are not closed," but this is only achieved by replacing the original specification with a completely different expression where the count of files and the expected fraction are explicitly mentioned.

More surprisingly, it can also be considered a problem that these approaches associate traces with numerical values, for several reasons. First, calculating this value is generally complex and its precise meaning often eludes intuition. Furthermore, mapping traces to numbers can lead to improper interpretations, such as the fact that an execution obtaining a score of $1/4$ is "half as true" as another with a verdict of $1/2$.

Finally, the last problem lies in the fact that the set of real numbers is subject to a total order, which implies that for any two distinct traces, one of them is always ranked lower than the other —even in situations where this comparison is not appropriate. Consider for example the simple case of the statement $a \wedge b$. According to the semantics of existing works, an execution that satisfies a but not b is equivalent (i.e. is scored identically) to an

execution that does the opposite, since both fail for "half" of their arguments. Manually defined weights can give higher precedence to one of the terms, but the eventuality that these two types of failures are simply distinct and incomparable cannot be accounted for.

3 Conditions on Event Sequences

We start by briefly describing the formal foundations of conformance used in this paper. With the exception of evaluation trees (§3.3), these notions have already been presented and used in past works about conformance, and thus this section should be seen as a quick refresher.

3.1 Event Model

Let P be an arbitrary set of parameter *names*, and V be a set of *values*. An event is modeled as a total function $e : P \to V$, which maps every parameter to a value. We reserve a special symbol # to represent the fact that no value is assigned to a parameter (i.e. that it is absent from an event). Suppose for example that $P = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ is the set of parameter names and their values are natural numbers (i.e. $V = \mathbb{N}$). The fact that an event assigns the value 3 to a and the value 1 to c, leaving b undefined, shall be denoted by $\{\mathsf{a} \mapsto 3, \mathsf{c} \mapsto 1\}$.

We suppose that the execution of a process produces a finite sequence of events $\overline{e} = e_0, \ldots, e_k$ called an event *trace*. We denote by $\mathcal{E}^*$ the set of all finite traces of events. Following conventions, the notation $\overline{e}[i]$ will designate the i-th event of $\overline{e}$ (indices starting at 0), $\overline{e}[i..]$ will stand for the suffix of $\overline{e}$ that starts at index i, while $\overline{e}[..i]$ will stand for the prefix of $\overline{e}$ that ends at index i. The length of $\overline{e}$ is noted $|\overline{e}|$, and ϵ designates the unique trace of length 0.

3.2 Linear Temporal Logic

In order to express conditions on what constitute valid traces, we take up an existing logical formalism called Linear Temporal Logic (LTL). A sequence of events $\overline{e}$ is said to satisfy an expression φ, noted $\overline{e} \models \varphi$, if it follows the semantic rules shown in Table 1. In this table, p denotes an arbitrary predicate evaluated on concrete arguments $\pi_1, \ldots, \pi_n$.

$$
\begin{aligned}
\overline{e} \models p(\pi_1, \ldots, \pi_n) &\Leftrightarrow p(\pi_1, \ldots, \pi_n) \text{ holds in } \overline{e}[0] \\
\overline{e} \models \neg\varphi &\Leftrightarrow \overline{e} \not\models \varphi \\
\overline{e} \models \varphi \wedge \psi &\Leftrightarrow \overline{e} \models \varphi \text{ and } \overline{e} \models \psi \\
\overline{e} \models \mathbf{X}\,\varphi &\Leftrightarrow \overline{e}[1..] \models \varphi \\
\overline{e} \models \mathbf{G}\,\varphi &\Leftrightarrow \overline{e}[i..] \models \varphi \text{ for every } i \in [0, |\overline{e}| - 1] \\
\overline{e} \models \mathbf{Y}\,\varphi &\Leftrightarrow \overline{e}[..|\overline{e}| - 2] \models \varphi \\
\overline{e} \models \mathbf{H}\,\varphi &\Leftrightarrow \overline{e}[..i] \models \varphi \text{ for every } i \in [0, |\overline{e}| - 1]
\end{aligned}
$$

Table 1: The formal semantics of LTL with past operators.

Boolean connectives have their usual meaning. The temporal operator $\mathbf{G}$ means "globally". For example, the formula $\mathbf{G}\,\varphi$ means that formula φ is true in every suffix of

the trace, starting from the current event. The operator **X** means "next"; it is true whenever φ holds in the suffix starting at the next event of the trace. Operators **H** ("historically") and **Y** ("yesterday") are the past duals of **G** and **X**: $\mathbf{H}\,\varphi$ holds for some trace $\overline{e}$ if φ holds in every prefix of $\overline{e}$, while $\mathbf{Y}\,\varphi$ holds if φ holds for the prefix of $\overline{e}$ that omits the last event. The definition of the remaining connectives and operators is obtained through the classical identities.[1] The presence of past-time modalities does not increase the expressiveness of the language, but are sufficient to express the "until" modality using a mix of unary future and past operators, since $\varphi\,\mathbf{U}\,\psi$ can be rewritten as $\mathbf{F}(\psi \wedge \mathbf{H}\,\varphi)$. A specific semantics needs to be specified for these operators in the case of an empty trace; we follow conventional definitions assuming that $\epsilon \models \mathbf{G}\,\varphi$, $\epsilon \not\models \mathbf{X}\,\varphi$, and dually for their past equivalents. This setup it is expressive enough for a wide range of constraints, including temporal patterns for finite-state specifications [11], as well as specification languages whose semantics is grounded in LTL, such as ConDec [32] or DECLARE [1].

We can revisit the conditions of the bank process of Section 2.1 and express them as LTL expressions. We assume that each event in the process has attributes respectively representing the name of the activity being executed (a), the status of the client in this process instance (s), the seniority level of the employee performing the activity (ℓ), and the time elapsed since the reception of the original request (τ).

1. $\mathbf{H}\,(\mathsf{a} = \text{"open"} \rightarrow (\mathbf{O}\,(\mathsf{a} = \text{"obtain"}) \wedge \mathbf{O}\,(\mathsf{a} = \text{"verify"})))$
2. $\mathsf{s} = \text{"VIP"} \rightarrow (\mathbf{G}\,(\mathsf{a} = \text{"open"} \rightarrow \tau < 2))$
3. $\mathbf{F}\,(\ell = \text{"manager"} \wedge \mathsf{a} = \text{"prepare"}) \vee \mathbf{F}\,(\ell = \text{"manager"} \wedge \mathsf{a} = \text{"open"})$ $\vee\, \mathbf{F}\,(\ell = \text{"manager"} \wedge \mathsf{a} = \text{"validate"})$
4. $\mathbf{H}\,(\mathsf{a} = \text{"deposit"} \rightarrow \mathbf{Y}\,\mathsf{a} = \text{"evaluate"})$

3.3 Evaluation Trees

The recursive evaluation of an LTL expression on a sequence of events induces a tree structure that can be leveraged to compare two executions of a given process or system, which we shall call the *evaluation tree*. In this context, an evaluation tree node can be represented as a vector of the form $t = \langle \ell, [t_1, \ldots, t_n] \rangle$, where ℓ is an arbitrary textual label, and the t_i are themselves tree nodes corresponding to the children of t (for a leaf node, the list is simply empty). Given a trace $\overline{e}$ and an LTL expression φ, the evaluation tree of φ on $\overline{e}$, noted $\tau(\varphi, \overline{e})$, is the tree structure resulting from the recursive application of the rules specified in Table 2.

The label of each node represents either the predicate, Boolean connective or temporal operator. The children of the nodes correspond to the recursive evaluation of the arguments. For example, in the case of a conjunction $\varphi_1 \wedge \cdots \wedge \varphi_n$, children of the "$\wedge$" parent are the evaluation trees resulting from the evaluation of each φ_i. Temporal operators warrant further discussion; the case of a formula of the form $\mathbf{G}\,\varphi$ conveys the general principle followed by the remaining temporal modalities. The top-level node is labeled with **G**, and the children of this node correspond to the evaluation tree of φ for all non-empty suffixes of the input trace $\overline{e}$.

Nodes can also be associated with a "color" that represents the truth value of the LTL expression they stand for. The root node of $\tau(\overline{e}, \varphi)$ will be colored either green

[1] Namely: $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$, $\mathbf{F}\,\varphi \equiv \neg\,\mathbf{G}\,\neg\varphi$, and $\mathbf{O}\,\varphi \equiv \neg\,\mathbf{H}\,\neg\varphi$.

$$
\begin{aligned}
\tau(\overline{e}, p(\pi_1, \ldots, \pi_n)) &= \langle p, [\pi_1, \ldots, \pi_n] \rangle \\
\tau(\overline{e}, \neg\varphi) &= \langle \neg, [\tau(\overline{e}, \varphi)] \rangle \\
\tau(\overline{e}, \varphi_1 \wedge \cdots \wedge \varphi_n) &= \langle \wedge, [\tau(\overline{e}, \varphi_1), \ldots, \tau(\overline{e}, \varphi_n)] \rangle \\
\tau(\overline{e}, \varphi_1 \vee \cdots \wedge \varphi_n) &= \langle \vee, [\tau(\overline{e}, \varphi_1), \ldots, \tau(\overline{e}, \varphi_n)] \rangle \\
\tau(\overline{e}, \mathbf{X}\,\varphi) &= \langle \mathbf{X}, [\tau(\overline{e}[1..], \varphi)] \rangle \\
\tau(\overline{e}, \mathbf{G}\,\varphi) &= \langle \mathbf{G}, [\tau(\overline{e}[0..], \varphi), \tau(\overline{e}[1..], \varphi), \ldots, \tau(\overline{e}[-1], \varphi)] \rangle \\
\tau(\overline{e}, \mathbf{F}\,\varphi) &= \langle \mathbf{F}, [\tau(\overline{e}[..|\overline{e}| - 2], \varphi), \tau(\overline{e}[1..], \varphi), \ldots, \tau(\overline{e}[-1], \varphi)] \rangle \\
\tau(\overline{e}, \mathbf{Y}\,\varphi) &= \langle \mathbf{Y}, [\tau(\overline{e}[1..], \varphi)] \rangle \\
\tau(\overline{e}, \mathbf{H}\,\varphi) &= \langle \mathbf{H}, [\tau(\overline{e}[..|\overline{e}| - 1], \varphi), \tau(\overline{e}[..|\overline{e}| - 2], \varphi), \ldots, \tau(\overline{e}[0], \varphi)] \rangle \\
\tau(\overline{e}, \mathbf{O}\,\varphi) &= \langle \mathbf{O}, [\tau(\overline{e}[..|\overline{e}| - 1], \varphi), \tau(\overline{e}[..|\overline{e}| - 2], \varphi), \ldots, \tau(\overline{e}[0], \varphi)] \rangle
\end{aligned}
$$

Table 2: Definition of the evaluation tree for an LTL expression evaluated on a trace.

or red, depending on whether $\overline{e} \models \varphi$ or not. Since by construction, the leaf nodes of an evaluation tree are concrete values given as arguments to a predicate, and can be constants of any type, these nodes will be left uncolored.

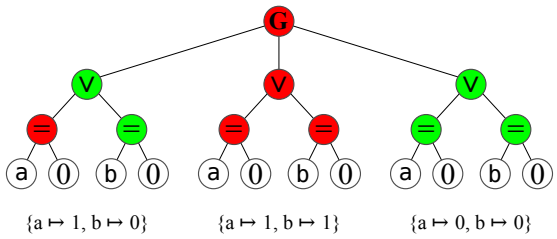


Fig. 2: An example of an evaluation tree.

Figure 2 shows an example of such a colored evaluation tree, for the expression $\mathbf{G}(\mathsf{a} = 0 \vee \mathsf{b} = 0)$. It is evaluated on a trace $\overline{e}$ of three events, the value of a and b in each of them being shown at the bottom of the figure. As stipulated in Table 2, each suffix of the trace spawns a distinct subtree of the root node labeled "**G**", where the condition $\mathsf{a} = 0 \vee \mathsf{b} = 0$ is evaluated on the first event of the suffix. The color of each node is assigned according to the satisfaction of the corresponding sub-expression.

4 A Tree-Based Definition of Conformance

As with any other logic-based formalization of valid executions, the semantics of Table 1 only outputs a pass/fail verdict. Based on the observations of the existing multi-valued verdict definitions surveyed in Section 2.2, we set on to propose an alternative notion of conformance with respect to a condition that addresses the issues raised in Section 2.3.

Since the numerical value that these approaches associate with a trace generally carries little meaning in itself, its only legitimate use lies in the fact that it can be used to compare two traces. Yet, to reach this goal, one can simply establish a *relation* (in the mathematical sense of the term) that allows one to determine, given two traces $\overline{e}$ and $\overline{e}'$, which of the two (if any) comes before the other. Therefore, we propose an alternative notion of conformance that compares the trees resulting from the evaluation of a condition. We shall first formally define this relation, and highlight its key properties through simple examples.

4.1 Definition of a Comparison Relation

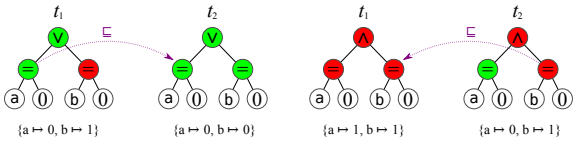
In the following, without loss of generality, we assume that an LTL condition φ is expressed in Negated Normal Form (NNF). For two evaluation trees, we then define *subsumption* as follows.

Definition 1. Let $\overline{e}_1$ and $\overline{e}_2$ be two event traces, φ be an LTL formula in NNF, and t_1 and t_2 be the corresponding evaluation trees resulting from the evaluation of φ through τ. We say that t_1 is *subsumed* by t_2, noted $t_1 \sqsubseteq t_2$, if the roots of both trees have the same label, and if the following rules are satisfied:

(a) if the root of t_1 is green, then 1) the root of t_2 is green and 2) for every green child t'_1 of the root of t_1, there exists a distinct child t'_2 of the root of t_2 such that $t'_1 \sqsubseteq t'_2$;
(b) if the root of t_1 is red, then for every red child t'_2 of the root of t_2, there exists a distinct child t'_1 of the root of t_1 such that $t'_1 \sqsubseteq t'_2$.

Intuitively, an evaluation tree t_1 is subsumed by another tree t_2 if the latter represents an execution of a process that is "more favorable" than the former with respect to φ. If none of these conditions are satisfied, then t_1 is not subsumed by t_2; note however that, contrary to numbers, this does not imply the reverse statement (i.e. that t_2 is subsumed by t_1). We shall note $t_1 \sqsubset t_2$ when $t_1 \sqsubseteq t_2$ but $t_2 \not\sqsubseteq t_1$. We also note $t_1 \approx t_2$ when both $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq t_1$; this is possible even when $t_1 \neq t_2$. Let us now illustrate the consequences of this definition on a few simple examples.

Boolean Connectives First, let us consider the property φ defined as $\mathsf{a} = 0 \vee \mathsf{b} = 0$. It imposes a condition on two attributes of the first event of an execution. Figure 3a shows the evaluation tree for two traces made of a single event: the first is such that $\mathsf{a} = 0$ and $\mathsf{b} = 1$, and the second has $\mathsf{a} = 0$ and $\mathsf{b} = 0$. Both of these traces satisfy φ, and thus produce trees with a green root.



(a) $\vee$ satisfied (b) $\wedge$ violated

Fig. 3: Two examples of subsumption for the $\wedge$ and $\vee$ connectives.

We can apply Definition 1 and conclude that $t_1 \sqsubseteq t_2$. First, condition a.1 is obviously satisfied. For condition a.2, one must exhibit a mapping between the (single) green child t'_1 of t_1, and some child t'_2 of t_2, such that $t'_1 \sqsubseteq t'_2$. The purple arrow shows the only possible mapping. Recursively, one can apply Definition 1 again and observe that t'_1 is indeed subsumed by t'_2 (the conclusion is direct in this case as the two trees are identical). However, it is not possible to conclude that $t_2 \sqsubseteq t_1$. Since t_2 has two green nodes, by Definition 1, it is impossible to subsume each of its green subtrees by a *distinct* subtree of t. These conclusions match the intuition: φ stipulates that an execution is

valid whenever a or b is null in the first event; t represents the case where one of these conditions is fulfilled, while t' represents the case where both are fulfilled. In a way $t \sqsubseteq t'$ illustrates the fact that, while the two executions satisfy the property, the second exceeds the expectations compared to the first.

The reverse argument can be made for two executions that violate a property, as is shown in Figure 3b. This time, $t_1 \sqsubseteq t_2$ holds because every *red* subtree of t_2 can be associated to a subtree of t_1 that is subsumed by it (purple arrow). By a symmetrical argument, reversing the order of the trees does not satisfy the subsumption relation. The fact that $t_1 \sqsubseteq t_2$, in this case, illustrates the fact that while both executions violate the condition, the first violates it by a larger margin.

Note however that subsumption is not merely a matter of counting how many terms of a connective are satisfied or violated. As an example, consider the trees of Figure 4, for the property φ defined as $\mathsf{a} = 0 \vee (\mathsf{b} = 0 \wedge \mathsf{c} = 0)$. Although $t_1 \sqsubseteq t_2$ and $t_1 \sqsubseteq t_3$, we have that neither t_2 nor t_3 subsumes the other. The conjunction fails in t_2 because $\mathsf{b} \neq 0$, while it fails in t_3 because $\mathsf{c} \neq 0$; by condition (c) of Definition 1, no subsumption relation exists between these subtrees. This is in line with the observation made previously that violating a property for two distinct "reasons", even though each represents a single failure, should not immediately be deemed equivalent.

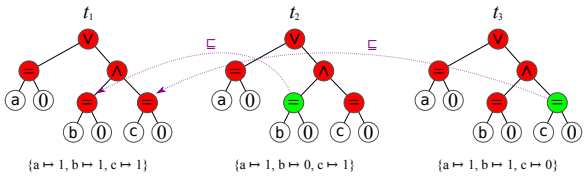
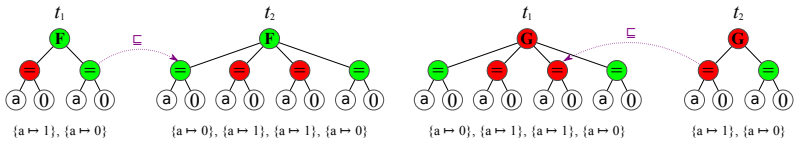


Fig. 4: Three evaluation trees for the property $\mathsf{a} = 0 \vee (\mathsf{b} = 0 \wedge \mathsf{c} = 0)$.

Temporal Operators So far, the mapping involved in conditions (a.2) and (b) of Definition 1 has amounted to a direct association between children at matching positions in both trees. However, this is not always the case, as can be seen in the handling of conformance requirements involving temporal operators. Figure 5a shows two evaluation trees for the property $\mathbf{F}(\mathsf{a} = 0)$. The first tree, t_1, corresponds to a trace of length 2 where $\mathsf{a} = 0$ in the second event, while the second tree, t_2, corresponds to a trace of length 4 where $\mathsf{a} = 0$ in the first and fourth events. One can observe that $t_1 \sqsubseteq t_2$; however, this time the mapping between the second child of t_1 associates it to the first child of t_2.



(a) **F** satisfied (b) **G** violated

Fig. 5: Two examples of subsumption for the **F** and **G** temporal modalities.

This example highlights a few characteristics of Definition 1. First, note that the mapping between children of both trees is not unique; another possibility would have

been to match the green child of t_1 with the fourth child of t_2; Definition 1 only requires the existence of a mapping. Second, the position in the trace where trees are associated is irrelevant –provided that these trees satisfy the subsumption relationship. In other words, two traces that satisfy $\mathsf{a} = 0$ exactly once will be deemed equivalent regardless of the actual index of the event where $\mathsf{a} = 0$. As with Boolean connectives, the dual reasoning can be made on executions that violate a temporal property. Figure 5b shows an example for the condition $\mathbf{G}(\mathsf{a} = 0)$ and the same two traces as before.

4.2 Properties of $\sqsubseteq$

The previous examples have shown that the subsumption relation addresses some of the issues leveled at related works on the topic. First, it works directly on a condition producing a Boolean pass/fail verdict, and does not require it to be somehow rewritten in order to allow a finer-grained verdict. Rather, it extracts additional information from the evaluation of the condition to determine if one trace ranks higher than the other, if any. In the same way, no additional information (in the form of user-defined weights, aggregation functions, external counters, etc.), is required to calculate this ranking. The subsumption relation can thus be retro-fitted on any pre-existing set of conditions expressed in a notation that is covered by LTL.

In addition, its relatively simple expression makes it possible to formally establish properties of the relation. For instance, we can show that this comparison relation is "well-behaved", in the sense that it forms a *preorder* over the set of evaluation trees (the proof is relatively simple and is omitted due to space limitations).

Theorem 1 (Preorder)**.** Let φ be a condition, $\overline{e}_1, \overline{e}_2, \overline{e}_3$ be three traces and t_1, t_2, t_3 be their respective evaluation trees. Then: 1) $t_1 \sqsubseteq t_1$; 2) if $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq t_3$, then $t_1 \sqsubseteq t_3$.

Finally, one can remark that in the case of a green root, only *green* children need to be subsumed by some child of the other tree. This entails that, in the case of Figure 5a, an execution $\overline{e}$ where a is null in 2 out of 4 events will be ranked higher than another $\overline{e}'$ where a is null once out of 2 events. The reasoning behind this behavior is that in the case of t_2, changing an event where $a = 0$ for another value still results in a conforming execution, as a remaining occurrence of $a = 0$ still ensures that the property is satisfied. In contrast, in t_1, conformance hinges on the single occurrence where $a = 0$. There is, therefore, a stronger support for the satisfaction of φ in t_2. Note that this notion is not appropriately conveyed if one were to express conformance through a ratio of true to false terms (which, in that case, would rank $\overline{e}'$ equal to $\overline{e}$). A reverse argument can be made in the case of red nodes for trees that violate a specification.

4.3 Handling Numerical Conditions

The conditions given as examples so far were limited to asserting the equality of certain parameters to constants; the fact that these values were numbers was not considered relevant. However, as seen in Section 2, there are scenarios where events can contain numerical values from measurements (timestamps, prices, etc.), and the satisfaction of a constraint may be modulated by the distance from the observed value to a certain reference value.

Although our model based on evaluation trees is entirely discrete and involves no arithmetic calculations, it is still possible, to a certain extent, to model this notion of numerical distance. Consider, for example, the property stating that $\mathsf{a} = 4$. One might wish to view values close to 4, although they represent a violation of the condition, as still preferable to more distant values, as shown at the top of Figure 6.

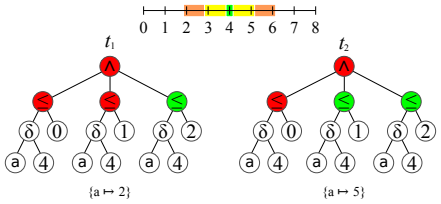


Fig. 6: Subsumption for a condition involving a numerical value.

It is possible, without changing the definition of subsumption, to achieve such behavior by replacing the original condition with an expression like $\delta(\mathsf{a}, 4) \leq 0 \wedge \delta(\mathsf{a}, 4) \leq 1 \wedge \delta(\mathsf{a}, 4) \leq 2$, where δ is an auxiliary function defined as $\delta(x, y) = |x - y|$. The trees t_1 and t_2 in Figure 6 show the result of evaluating this condition for two events, the first where $\mathsf{a} = 2$ and the second where $\mathsf{a} = 5$. In the first case, only the last inequality is satisfied, while the last two inequalities are satisfied in the second; as a result, $t_1 \sqsubseteq t_2$.

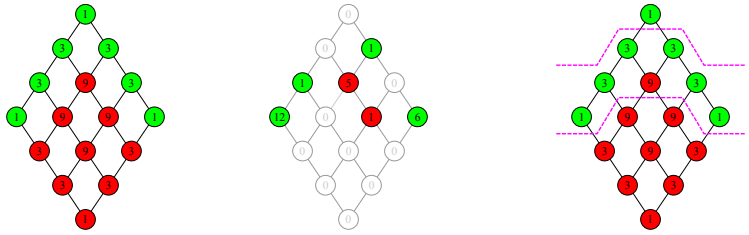
Note that it is not necessary for the property provided by the user to be directly written in this way. Instead, one could consider that a numerical equality be accompanied by the definition of one or more distance intervals, and that the transformation into a series of inequalities be performed automatically in the background. Also observe that the same "trick" can be used in reverse for satisfaction: one could express the condition that a lies between 2 and 6, and rank traces higher as the value gets close to 4.

4.4 Subsumption for a Set of Traces

The fact that $\sqsubseteq$ is a preorder entails that, given a LTL property φ and set of event traces $E = \{\overline{e}_1, \ldots, \overline{e}_n\}$, one can define the equivalence class of $\overline{e}_i$, noted $[\overline{e}_i]$, as the set $\{\overline{e} \in E : \overline{e} \approx \overline{e}_i\}$. The ordering relation on traces can be lifted to equivalence classes; we have that $[\overline{e}_i] \sqsubseteq [\overline{e}_j]$ if and only if $\overline{e}_i \sqsubseteq \overline{e}_j$. Given a LTL property φ and a set of traces E, it is possible to characterize the structure of E by considering its Hasse diagram [9]. This diagram is defined as a graph whose vertices are the equivalence classes of E, and for every pair of classes $[\overline{e}]$, $[\overline{e}']$, a directed edge from the former to the latter exists whenever $[\overline{e}]$ *covers* $[\overline{e}']$ —that is, $[\overline{e}] \sqsubseteq [\overline{e}']$ and there does not exist a distinct $[\overline{e}'']$ such that $[\overline{e}] \sqsubseteq [\overline{e}''] \sqsubseteq [\overline{e}']$.

As an example, consider the property φ defined as $\mathbf{G}(\mathsf{a} = 0) \vee \mathbf{G}(\mathsf{b} = 0)$ against the set of all traces of length 3 where, in each event, a and b take either the value 0 or 1. This corresponds to a total of 64 distinct traces. Figure 7a shows the Hasse diagram of this set with respect to φ. Each node in the diagram is labeled with the number of distinct traces in the equivalence class it corresponds to. Following convention, the direction of edges is from bottom to top; thus the bottom node labeled "1" corresponds to the single trace where $\mathsf{a} = 0$ and $\mathsf{b} = 0$ in all three events. This trace is directly subsumed by two disjoint

set of traces, namely those where a is never null and b = 0 exactly once (to its top left), and the reverse (to its top right).



(a) Original (b) Sample log (c) Service Level Agreement

Fig. 7: The Hasse diagram for a log with respect to the property $\mathbf{G}(\mathsf{a} = 0) \vee \mathbf{G}(\mathsf{b} = 0)$.

The diagram presents a very regular structure, due to the simple and symmetric nature of the underlying condition; more complex formulas induce diagrams with less regularity. One can observe that, due to Definition 1, there exists a clear segregation between conforming (green) and non-conforming (red) traces. In other words, while the subsumption relation admits a form of comparison between execution traces, it never blurs the border between satisfaction and violation.

Figure 7a shows a diagram for a complete set of traces of given length, considering all possible assignments of parameters in all events. In reality, a log of a given process is very unlikely to have this characteristic. Thus, it may be instructive to compute the Hasse diagram of an actual set of logs for a given property, and to compare it to the "abstract" version covering all possible behaviors. Figure 7b shows what such a diagram could look like for a hypothetical log with respect to property φ. Most executions are conforming; however almost all of them do so by satisfying only one of the two temporal conditions: either a is always null but b never is (node labeled "12"), or the reverse (node labeled "6"). Only two traces (the two green nodes "1") satisfy one condition and the other partially. Moreover, the violations observed (red nodes) are still "not too far" from a correct execution: in both cases they have an immediate neighbor that is green. Larger violations would be represented by nodes lying further down in the diagram.

As one can see, the analysis of this diagram can provide insight in the execution of a process with respect to a conformance condition φ. It allows one to qualitatively assess the degree to which the condition is satisfied or violated by each trace in a log, but also to identify common reasons for a violation —since all traces in the same equivalence class can be seen as sharing similar behavior with respect to φ. Approaches that associate a numerical value to each trace have the result of flattening the whole set on a linear scale that cannot reveal such structures.

5 Evaluating Tree-Based Conformance

The formal definition of conformance introduced in Section 4 has been implemented in the form of a stand-alone tool. In this section, we discuss this implementation and present preliminary results of the application of the subsumption relation on sample traces.

The implementation takes the form of a stand-alone Java-based library that is available under an open source license[2]. In addition to the objects it defines and which can be manipulated directly in a Java program, another possible use is as a command-line tool, where one can read traces from local files and compare their evaluation tree with respect to a given LTL conformance property. For example, to compare traces from two XML files against an LTL property contained in text file `phi.ltl`, one would write:

```
$ java -jar tc.jar compare --property phi.ltl file1.xml file2.xml
```

If n trace filenames are provided, the output of the tool is an $n \times n$ matrix, where entry (i, j) is `1` if the trace in file i is subsumed by the trace in file j, and `0` otherwise.

If `compare` is replaced by the action `draw-trees`, an image of the evaluation tree is produced for each of the traces provided, and saved to a local file in the same folder. Finally, the last supported action is `draw-hasse`, which generates the Hasse diagram of the set of traces given as arguments. A command line option allows each evaluation tree to be drawn to a file, with an indication of the node in the Hasse diagram it belongs to.

Definition 1 hides implicit complexity in that for two evaluation trees t_1 and t_2, one must find an injection that maps each (either red or green) child of t_1 to a unique (red or green) child of the other. Suppose that t_1 has m such children, and t_2 has $n \geq m$ children; these children can be represented as the discrete sets $S_1 = \{1, \ldots, m\}$ and $S_2 = \{1, \ldots, n\}$. The number of possible injective functions $S_1 \to S_2$ is $m! \cdot \binom{n}{m}$. This could represent a barrier to the use of the subsumption relation in practice. Therefore, we proceeded to an experimental evaluation of the performance overhead induced by the evaluation of this relation, for sample LTL conformance properties on sample logs. In addition to the banking example discussed in Section 2.1, the scenarios and properties considered are:

- *Beep Store*: a set of logs from a web service implementing shopping cart manipulations similar to Amazon's ECS [19]. Property *Search item once* looks for the occurrence of an `ItemSearch` message, while *Max shopping carts* checks that the number of `CartCreate` events does not exceed a certain threshold.
- *CVC Procedure*: operations recorded from multiple instances of a medical procedure, from the Conformance Checking Challenge 2019 [15]. Property *Max duration* ensures that the delay between two successive operations does not exceed a predefined time, while *Procedure lifecycle* checks several ordering relationships between operations.

Table 3 summarizes the results for each scenario. For each, the length of the traces considered, the size of the corresponding evaluation trees, and the time required to compare two trees is reported; this time includes the generation of the evaluation trees and the evaluation of the subsumption relation. The T/O (timeout) column indicates the number of tree pairs for which the evaluation of the relation exceeded the predefined timeout of 3 seconds. In addition, we report on the number of pairs of traces that have been compared, and the number of times the subsumption relation held between two traces.

We can observe that the fears regarding the combinatorial explosion related to searching for an injection between two trees did not materialize in these examples.

[2] `https://github.com/liflab/shaded-conformance`

Table 3: Summary of experimental results.

Scenario	Property	Trace size		Tree size		Pairs	Time (ms)	T/O	Subsumed	Refin.
		Min.	Max.	Min.	Max.					
Beep Store	Search item once	10	100	4	91	1770	2.27	0	802	1712
	Max shopping carts	10	100	8	183	1770	2.96	0	1023	1657
CVC Procedure	Max. duration	26	59	16	66	190	13.2	0	93	140
	Procedure lifecycle	26	59	565	1291	190	106	0	113	73
Bank	Condition 1	13	21	679	1599	45	585	33	13	11
	Condition 2	13	21	16	66	45	53.7	0	45	0
	Condition 3	13	21	277	445	45	133	0	29	39
	Condition 4	13	21	102	150	45	52.2	0	38	15

Although we are dealing with traces of several dozen events and trees sometimes exceeding a thousand nodes, the average evaluation time required to compare two trees remains on the order of milliseconds.

More interestingly, the last column indicates the number of times the subsumption relation provides more refined information than the simple Boolean evaluation of the condition on each of the two trees t and t' considered. This occurs when both executions either violate or satisfy the conformance condition, but the subsumption relation still manages to distinguish between them (i.e., $t_1 \sqsubset t_2$, or vice versa). We can see that in most cases, the subsumption relation provides added value for a significant proportion of the tree pairs.

6 Conclusion

This article presented a new relationship that allows the comparison of two execution traces relative to a conformance property, inducing a form of gradation in the level with which a trace respects or violates the said property. Unlike existing approaches that typically associate each trace with a numerical value, we saw that the so-called subsumption relationship relies on the tree structure resulting from the evaluation of a temporal logic formula for a given trace. This relationship can accommodate the comparison of numerical values, and the examples illustrate that the ordering of the traces it produces can be explained intuitively. In this sense, it is an effective and versatile technique for analyzing the level of conformance of a set of traces.

This basic idea opens the way for several applications and extensions. Firstly, the Hasse diagrams produced by a log and discussed in Section 4.4 can form the core of a new diagnostic method to identify the reasons why a process meets or violates a requirement. This diagram could also be used in the context of Service Level Agreements (SLA). An SLA could be stated in the form of a Boolean condition, but be equipped with different levels of service corresponding to strata in the Hasse diagram corresponding to the property, as is shown in Figure 7c.

Although LTL is sufficient to model a large number of conformance constraints, it would be desirable to extend the subsumption relationship itself to more expressive specification languages, such as logics including the possibility of compensation for a violation [16]. Finally, to promote the use of this comparison measure, in the short term, we aim to integrate the stand-alone library presented in this article with existing platforms, for example in the form of a plugin for the ProM tool [8], or an extension to the BeepBeep library [21].

References

1. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Comput. Sci. Res. Dev. **23**(2), 99–113 (2009). https://doi.org/10.1007/S00450-009-0057-9, https://doi.org/10.1007/s00450-009-0057-9
2. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. In: Dumas, M., Reichert, M., Shan, M. (eds.) Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5240, pp. 326–341. Springer (2008). https://doi.org/10.1007/978-3-540-85758-7_24, https://doi.org/10.1007/978-3-540-85758-7_24
3. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. J. Vis. Lang. Comput. **22**(1), 30–55 (2011)
4. Baresi, L., Pasquale, L.: A Temporal Semantics for Fuzzy Linear Temporal Logic. Tech. rep., https://www.academia.edu/2935585/A_Temporal_Semantics_for_Fuzzy_Linear_Temporal_Logic
5. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457, pp. 1–33. Springer (2018)
6. Burattin, A., Guizzardi, G., Maggi, F.M., Montali, M.: Fifty shades of green: How informative is a compliant process trace? In: Giorgini, P., Weber, B. (eds.) Advanced Information Systems Engineering, vol. 11483, pp. 611–626. Springer International Publishing, Cham (2019)
7. Chesani, F., Mello, P., Montali, M., Riguzzi, F., Sebastianis, M., Storari, S.: Checking compliance of execution traces to business rules. In: BPM Workshops. pp. 134–145 (2008)
8. Claes, J., Poels, G.: Process mining and the prom framework: An exploratory survey. In: Rosa, M.L., Soffer, P. (eds.) Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers. Lecture Notes in Business Information Processing, vol. 132, pp. 187–198. Springer (2012). https://doi.org/10.1007/978-3-642-36285-9_19, https://doi.org/10.1007/978-3-642-36285-9_19
9. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press (2002)
10. Díaz, G., Llana, L.: Contract compliance monitoring of web services. In: Lau, K., Lamersdorf, W., Pimentel, E. (eds.) Service-Oriented and Cloud Computing - Second European Conference, ESOCC 2013, Málaga, Spain, September 11-13, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8135, pp. 119–133. Springer (2013)
11. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999. pp. 411–420. ACM (1999)
12. Fdhila, W., Knuplesch, D., Rinderle-Ma, S., Reichert, M.: Verifying compliance in process choreographies: Foundations, algorithms, and implementation. Inf. Syst. **108**, 101983 (2022)
13. Feijs, L.M.G., Goga, N., Mauw, S., Tretmans, J.: Test Selection, Trace Distance and Heuristics, pp. 267–282. Springer US, Boston, MA (2002)
14. Francalanza, A., Cini, C.: Computer says no: Verdict explainability for runtime monitors using a local proof system. J. Log. Algebraic Methods Program. **119**, 100636 (2021)
15. de la Fuente, R., Sepúlveda, M., Fuentes, R.: Central veinous catheter, compliance checking challenge 2019 (2019), https://data.4tu.nl/repository/uuid:c923af09-ce93-44c3-ace0-c5508cf103ad
16. Governatori, G.: The regorous approach to process compliance. In: Kolb, J., Weber, B., Hallé, S., Mayer, W., Ghose, A.K., Grossmann, G. (eds.) 19th IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2015, Adelaide, Australia,

September 21-25, 2015. pp. 33–40. IEEE Computer Society (2015). https://doi.org/10.1109/EDOCW.2015.28, https://doi.org/10.1109/EDOCW.2015.28
17. Groefsema, H., van Beest, N.R.T.P., Governatori, G.: On the use of the conformance and compliance keywords during verification of business processes. In: Ciccio, C.D., Dijkman, R.M., del-Río-Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management Forum - BPM 2022 Forum, Münster, Germany, September 11-16, 2022, Proceedings. Lecture Notes in Business Information Processing, vol. 458, pp. 21–37. Springer (2022). https://doi.org/10.1007/978-3-031-16171-1_2, https://doi.org/10.1007/978-3-031-16171-1_2
18. Hallé, S.: Explainable queries over event logs. In: 24th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2020, Eindhoven, The Netherlands, October 5-8, 2020. pp. 171–180. IEEE (2020)
19. Hallé, S., Villemaire, R.: Constraint-based invocation of stateful web services: The Beep Store (case study). In: Lago, P., Lewis, G.A., Metzger, A., Tosic, V. (eds.) 4th International ICSE Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2012, June 4, 2012, Zurich, Switzerland. pp. 61–62. IEEE (2012)
20. Hallé, S., Villemaire, R.: Runtime enforcement of web service message contracts with data. IEEE Trans. Serv. Comput. **5**(2), 192–206 (2012)
21. Hallé, S.: Event Stream Processing With BeepBeep 3: Log Crunching and Analysis Made Easy. Presses de l'Université du Québec (2018)
22. Hashmi, M., Governatori, G., Lam, H., Wynn, M.T.: Are we done with business process compliance: state of the art and challenges ahead. Knowl. Inf. Syst. **57**(1), 79–133 (2018)
23. Khoury, R., Hallé, S.: Tally keeping-ltl: An LTL semantics for quantitative evaluation of LTL specifications. In: 2018 IEEE International Conference on Information Reuse and Integration, IRI 2018, Salt Lake City, UT, USA, July 6-9, 2018. pp. 495–502. IEEE (2018)
24. Lam, H., Hashmi, M., Kumar, A.: Towards a formal framework for partial compliance of business processes. In: Rodríguez-Doncel, V., Palmirani, M., Araszkiewicz, M., Casanovas, P., Pagallo, U., Sartor, G. (eds.) AI Approaches to the Complexity of Legal Systems XI-XII - AICOL International Workshops 2018 and 2020: AICOL-XI@JURIX 2018, AICOL-XII@JURIX 2020, XAILA@JURIX 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13048, pp. 90–105. Springer (2020). https://doi.org/10.1007/978-3-030-89811-3_7, https://doi.org/10.1007/978-3-030-89811-3_7
25. Lamine, K.B., Kabanza, F.: History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In: 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000), 13-15 November 2000, Vancouver, BC, Canada. pp. 312–319. IEEE Computer Society (2000). https://doi.org/10.1109/TAI.2000.889888, https://doi.org/10.1109/TAI.2000.889888
26. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Syst. J. **46**(2), 335–362 (2007). https://doi.org/10.1147/SJ.462.0335, https://doi.org/10.1147/sj.462.0335
27. Loh, C.S., Sheng, Y.: Maximum similarity index (MSI): A metric to differentiate the performance of novices vs. multiple-experts in serious games. Comput. Hum. Behav. **39**, 322–330 (2014). https://doi.org/10.1016/J.CHB.2014.07.022, https://doi.org/10.1016/j.chb.2014.07.022
28. Lu, R., Sadiq, S.W., Governatori, G.: Measurement of compliance distance in business processes. Inf. Syst. Manag. **25**(4), 344–355 (2008). https://doi.org/10.1080/10580530802384613, https://doi.org/10.1080/10580530802384613
29. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)

30. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: BPM. LNCS, vol. 6896, pp. 132–147. Springer (2011)
31. Mannila, H., Ronkainen, P.: Similarity of event sequences. In: 4th International Workshop on Temporal Representation and Reasoning, TIME '97, Daytona Beach, Florida, USA, May 10-11, 1997. pp. 136–139. IEEE Computer Society (1997)
32. Montali, M.: The ConDec Language, pp. 47–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
33. Mulo, E., Zdun, U., Dustdar, S.: Monitoring web service event trails for business compliance. In: IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2009, 14-15 December 2009, Taipei, Taiwan. pp. 1–8. IEEE Computer Society (2009)
34. Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decis. Support Syst. **100**, 41–56 (2017)
35. Ray, D., Ligatti, J.: A theory of gray security policies. In: Pernul, G., Ryan, P.Y.A., Weippl, E.R. (eds.) Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9327, pp. 481–499. Springer (2015)
36. Tidjon, L.N., Frappier, M., Mammar, A.: Intrusion detection systems: A cross-domain overview. IEEE Commun. Surv. Tutorials **21**(4), 3639–3681 (2019)
37. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P.: Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8**(3), 269–284 (2019)