Interestingness of traces in declarative process mining: the Janus LTLp$_f$ approach

Alessio Cecconi[1][0000−0001−5730−6332], Claudio Di Ciccio[1][0000−0001−5570−0475], Giuseppe De Giacomo[2][0000−0001−9680−7658], and Jan Mendling[1][0000−0002−7260−524X]

[1] Vienna University of Economics and Business, Vienna, Austria
{alessio.cecconi,claudio.di.ciccio,jan.mendling}@wu.ac.at
[2] Sapienza University of Rome, Rome, Italy
degiacomo@dis.uniroma1.it

Abstract. Declarative process mining is the set of techniques aimed at extracting behavioural constraints from event logs. These constraints are inherently of a reactive nature, in that their activation restricts the occurrence of other activities. In this way, they are prone to the principle of ex falso quod libet: they can be satisfied even when not activated. As a consequence, constraints can be mined that are hardly interesting to users or even potentially misleading. In this paper, we build on the observation that users typically read and write temporal constraints as if-statements with an explicit indication of the activation condition. Our approach is called Janus, because it permits the specification and verification of reactive constraints that, upon activation, look forward into the future and backwards into the past of a trace. Reactive constraints are expressed using Linear-time Temporal Logic with Past on Finite Traces (LTLp$_f$). To mine them out of event logs, we devise a time bi-directional valuation technique based on triplets of automata operating in an on-line fashion. Our solution proves efficient, being at most quadratic w.r.t. trace length, and effective in recognising interestingness of discovered constraints.

Keywords: process mining, declarative processes, temporal logics, separation theorem, automata theory

1 Introduction

Declarative process mining is the set of techniques aimed at extracting and validating temporal constraints out of event logs, i.e., multi-sets of finite traces. The semantics of these constraints are typically expressed using Linear Temporal Logic on Finite Traces (LTL$_f$) over activities occurring in traces. Examples of declarative process modelling languages are DECLARE [23] and DCR Graphs [11]. In DECLARE, for instance, RESPONSE(a,b) is a constraint applying the parametric template RESPONSE to activities a and b. It imposes that if a occurs in a trace, then b must occur eventually afterwards. PRECEDENCE(c,d) states that if d occurs in a trace, then c must have occurred before.

Constraints are inherently of a reactive nature in that the activation of certain conditions, e.g., the occurrence of a or d, exert restrictions on the occurrence of other activities, that is

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-319-98648-7_8

b afterwards and c beforehand in the examples. For this reason, they are prone to the principle of *ex falso quod libet*: not activated constraints are satisfied. This is a serious problem for the ambition of process mining to provide a precise understanding into the behaviour of the process. Returning not activated constraints is hardly interesting to the user or even misleading. A trace, e.g., in which neither a nor d occur, satisfies both RESPONSE(a,b) and PRECEDENCE(c,d). To avoid such spurious results, approaches have been introduced to detect *satisfaction vacuity*. These are based on the parsing of the underlying LTL$_f$ formulae [14], tailored to language-specific templates [18], and checking the state-change of accepting automata [20]. Those solutions hardly provide results that meet the intuition of users who read and write constraints [7], as they respectively provide different activation criteria depending on how formulae are written [14], are restricted to the sole standard DECLARE templates [18], and are bound to the underlying automata semantics, thus, e.g., considering the occurrence of c, and not d, as an activation for PRECEDENCE(c,d) [20].

Against this background, we introduce the Janus approach. First, with this approach, the user can explicitly define activation conditions directly within the constraint formula, similarly to what devised for Compliance Rule Graphs in [17]. The rationale is that users themselves specify what the activation is, thereby making explicit what is of interest and what is not. Because constraints are meant to be activated upon specified conditions, we refer to them as *reactive constraints*. Second, we define a *degree of interestingness* that is computed on traces for such constraints. These constraints are expressed using Linear-time Temporal Logic with Past on Finite Traces (LTLp$_f$) [22,16,2] in a specific form that singles out the activating event, conditions on its past and conditions on its the future. Third, in order to mine these constraints out of event logs, we automatically derive triples of automata based on the well-known separation theorem [8], for the analysis of past prefix, current event, and future suffix of the analysed trace, every time a new activation occurs. Fourth, we integrate these concepts into a time bi-directional constraint evaluation algorithm operating in an on-line fashion, which inspired us to call this approach Janus. Our solution proves efficient with at most quadratic complexity w.r.t. trace length, and is effective in terms of separating interesting from not interesting constraints.

The paper is structured as follows. Section 2 revisits preliminaries, in particular LTLp$_f$ and how to separate its formulae through the *separation theorem*. Section 3 presents the Janus approach, defining the *reactive constraints*, their *interestingness degree*, and how to verify them using automata theory. Section 4 presents the algorithm to retrieve the interestingness degree in an on-line fashion. Section 5 evaluates our proposal in comparison with other state of the art process miners, both using synthetic and real-life logs. Section 6 concludes the paper and identifies directions for future research.

2 Preliminaries

Event logs. In this paper, we consider a *trace* as a sequence of events $t = \langle e_1,\ldots,e_n\rangle$, where $n \in \mathbb{N}$ is the length of the trace, and events e_i belong to an alphabet of symbols Σ. Function $t(i)$ returns event e_i occurring at *instant* i in the trace. With $t_{[i:j]}$ we identify a segment (henceforth, sub-trace) extracted from trace t and ranging from instant i to instant j, where $1 \leqslant i \leqslant j \leqslant n$. t_{R} is the reverse of a trace t, i.e., such that $t(i) = t_{\mathrm{R}}(n-i+1)$ for every $1 \leqslant i \leqslant n$. An *event log* $L = \{\!|t_1,t_2,\ldots,t_m|\!\} \in \mathbb{M}(\Sigma^*)$, or *log* for short, is a multi-set of

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

Table 1: Some DECLARE constraints expressed as Reactive Constraints

Constraint	LTL_f expression [3]	RCon	Separation degree
PARTICIPATION(a)	$\Diamond \mathsf{a}$	$t_{Start} \boxdot\!\!\rightarrow \Diamond \mathsf{a}$	1
INIT(a)	a	$t_{Start} \boxdot\!\!\rightarrow \mathsf{a}$	1
END(a)	$\Box\Diamond \mathsf{a}$	$t_{End} \boxdot\!\!\rightarrow \mathsf{a}$	1
RESPONDEDEXISTENCE(a,b)	$\Diamond \mathsf{a} \rightarrow \Diamond \mathsf{b}$	$\mathsf{a} \boxdot\!\!\rightarrow (\Diamond \mathsf{b} \vee \Diamond\!\!\!\cdot\, \mathsf{b})$	2
RESPONSE(a,b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b})$	$\mathsf{a} \boxdot\!\!\rightarrow \Diamond \mathsf{b}$	1
ALTERNATERESPONSE(a,b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b}) \wedge \Box(\mathsf{a} \rightarrow \bigcirc(\neg \mathsf{a}\, \mathbf{W}\, \mathsf{b}))$	$\mathsf{a} \boxdot\!\!\rightarrow \bigcirc(\neg \mathsf{a}\, \mathbf{U}\, \mathsf{b})$	1
CHAINRESPONSE(a,b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b}) \wedge \Box(\mathsf{a} \rightarrow \bigcirc \mathsf{b})$	$\mathsf{a} \boxdot\!\!\rightarrow \bigcirc \mathsf{b}$	1
PRECEDENCE(a,b)	$\neg \mathsf{b}\, \mathbf{W}\, \mathsf{a}$	$\mathsf{b} \boxdot\!\!\rightarrow \Diamond\!\!\!\cdot\, \mathsf{a}$	1
ALTERNATEPRECEDENCE(a,b)	$(\neg \mathsf{b}\, \mathbf{W}\, \mathsf{a}) \wedge \Box(\mathsf{b} \rightarrow \bigcirc(\neg \mathsf{b}\, \mathbf{W}\, \mathsf{a}))$	$\mathsf{b} \boxdot\!\!\rightarrow \ominus(\neg \mathsf{b}\, \mathbf{S}\, \mathsf{a})$	1
CHAINPRECEDENCE(a,b)	$(\neg \mathsf{b}\, \mathbf{W}\, \mathsf{a}) \wedge \Box(\bigcirc \mathsf{b} \rightarrow \mathsf{a})$	$\mathsf{b} \boxdot\!\!\rightarrow \ominus \mathsf{a}$	1

traces t_j with $1 \leqslant j \leqslant m$, $m \in \mathbb{N}$. We shall use a compact notation denoting the *multiplicity* of traces with a superscript, e.g., t_1^5 means that t_1 occurs 5 times in L.

Example 1. Let $L = \left\{t_1^{25}, t_2^{15}, t_3^{10}, t_4^{20}, t_5^{5}, t_6^{20}, t_7^{5}\right\}$ be an event log of 100 traces, defined on the alphabet of events $\Sigma = \{\mathsf{a}, \mathsf{b}, \ldots, \mathsf{i}\}$, where $t_1 = \langle \mathsf{d,f,a,f,c,a,f,b,a,f} \rangle$, $t_2 = \langle \mathsf{f,e,d,c,b,a,g,h,i} \rangle$, $t_3 = \langle \mathsf{a,d,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,c} \rangle$, $t_4 = \langle \mathsf{d,b,a,e} \rangle$, $t_5 = \langle \mathsf{a,d,a,c,a} \rangle$, $t_6 = \langle \mathsf{b,c,d,e} \rangle$, $t_7 = \langle \mathsf{b,c,a} \rangle$. We have that $t_1(7) = \mathsf{f}$, $t_{1[3:7]} = \langle \mathsf{a,f,c,a,f} \rangle$, $t_{2R} = \langle \mathsf{i,h,g,a,b,c,d,e,f} \rangle$.

DECLARE. DECLARE is a declarative process modelling language and notation [23]. It defines a set of standard templates based on [7], abstracting from the actual semantics used to describe them. In that way the complexity of the underneath logic is hidden to the user. The template parameters are tasks occurring as events in traces. For example, CHAINPRECEDENCE is a binary template stating that the occurrence of the second task imposes the first one to occur immediately before. PRECEDENCE loosens that condition requiring the second task to occur any time before the first one. DECLARE semantics are rooted in temporal logics. Its standard template set, a part of which is listed in Table 1, is meant to be extended with custom ones that suit best the user needs [23].

DECLARE *constraints* are templates whose parameters are assigned with tasks, e.g., PRECEDENCE(b,a) applies the PRECEDENCE template on tasks a, the *activation*, and b. Constraints are verified over events of traces. Those that do not *violate* a constraint, *satisfy* it. In Ex. 1, e.g., all events of t_7 satisfy PRECEDENCE(b,a) whereas CHAINPRECEDENCE(b,a) is violated by $t_7(3)$. Notice the principle of *ex falso quod libet*: both PRECEDENCE(b,a) and CHAINPRECEDENCE(b,a) are satisfied by t_6, where a, namely the *activation*, never occurs. This is arguably misleading, and calls for an approach that considers the reactive nature of constraints, i.e., such that it *(i)* singles out activations and *(ii)* dictates the conditions to check upon activation in the future and past of the trace. In the literature, constraints are formulated in LTL_f as of [3], as listed in Table 1. To cater for temporal modalities referring to the past, we resort on an extension of the syntax of LTL_f, namely the one of Linear-time Temporal Logic with Past on Finite Traces (LTLp_f).

LTLp_f. Well-formed LTLp_f formulae are built from an alphabet $\Sigma \supseteq \{\mathsf{a}\}$ of propositional symbols and are closed under the boolean connectives, the unary temporal operators

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

$\bigcirc$(next) and $\ominus$(previous), the binary temporal operators $\mathbf{U}$ (Until) and $\mathbf{S}$ (Since):

$$\varphi ::= \mathsf{a} \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (\bigcirc\varphi) \mid (\varphi_1 \,\mathbf{U}\, \varphi_2) \mid (\ominus\varphi) \mid (\varphi_1 \,\mathbf{S}\, \varphi_2).$$

From these basic operators it is possible to derive: classical boolean abbreviations *True*,*False*,$\vee$,$\rightarrow$; constant t_{End}, verified as $\neg\bigcirc$*True*, denoting the last instant of the trace; constant t_{Start}, verified as $\neg\ominus$*True*, denoting the first instant of the trace; $\Diamond\varphi$ as *True* $\mathbf{U}\,\varphi$ indicating that φ holds true eventually before t_{End}; $\varphi_1 \,\mathbf{W}\, \varphi_2$ as $(\varphi_1 \,\mathbf{U}\, \varphi_2) \vee \Box\varphi_1$, which relaxes $\mathbf{U}$ as φ_2 may never hold true; $\boxminus\varphi$ as *True* $\mathbf{S}\,\varphi$ indicating that φ holds true eventually in the past after t_{Start}; $\Box\varphi$ as $\neg\Diamond\neg\varphi$ indicating that φ holds true from the current instant till t_{End}; $\boxminus\varphi$ as $\neg\boxminus\neg\varphi$ indicating that φ holds true from t_{Start} to the current instant.
We remark that, w.l.o.g., we consider here the non-strict semantics of $\mathbf{U}$ and $\mathbf{S}$ [12].
Given a trace t, a LTLp$_f$ formula φ is satisfied in a given instant i ($1 \leqslant i \leqslant n$) by induction of the following:

$t,i \models$ *True*; $t,i \not\models$ *False*;
$t,i \models \mathsf{a}$ iff $t(i)$ is assigned with a;
$t,i \models \neg\varphi$ iff $t,i \not\models \varphi$;
$t,i \models \varphi_1 \wedge \varphi_2$ iff $t,i \models \varphi_1$ and $t,i \models \varphi_2$;
$t,i \models \bigcirc\varphi$ iff $i < n$ and $t,i+1 \models \varphi$;
$t,i \models \ominus\varphi$ iff $i > 1$ and $t,i-1 \models \varphi$;
$t,i \models \varphi_1 \,\mathbf{U}\, \varphi_2$ iff $t,j \models \varphi_2$ with $i \leqslant j \leqslant n$, and $t,k \models \varphi_1$ for all k s.t. $i \leqslant k < j$;
$t,i \models \varphi_1 \,\mathbf{S}\, \varphi_2$ iff $t,j \models \varphi_2$ with $1 \leqslant j \leqslant i$, and $t,k \models \varphi_1$ for all k s.t. $j < k \leqslant i$.

In the following, we will classify $\bigcirc$,$\Box$,$\Diamond$, $\mathbf{U}$ as *future operators*, $\ominus$,$\boxminus$,$\boxminus$, $\mathbf{S}$ as *past operators*, and the following pairs of operators as *mirror images*: *(i)* $\bigcirc$ and $\ominus$, *(ii)* $\Box$ and $\boxminus$, *(iii)* $\Diamond$ and $\boxminus$, *(iv)* $\mathbf{U}$ and $\mathbf{S}$. We shall also name as *mirror image* of formula φ the temporal formula obtained by replacing all its operators with their mirror images [25], henceforth denoted as φ_{M}.

Definition 1 (Pure temporal formula [8]). *A LTLp$_f$ formula φ is named:* **pure past** ($\varphi^{\blacktriangleleft}$) *if it contains only past operators;* **pure present** ($\varphi^{\blacktriangledown}$) *if it contains no temporal operators at all;* **pure future** ($\varphi^{\blacktriangleright}$) *if it contains only future operators.*

For example, $\varphi^{\blacktriangleleft} = \boxminus(\mathsf{a}\,\mathbf{S}\,(\boxminus\mathsf{b}))$, $\varphi^{\blacktriangledown} = \mathsf{a} \wedge \mathsf{b} \vee \mathsf{c}$, and $\varphi^{\blacktriangleright} = \Box(\Diamond\mathsf{a} \vee (\bigcirc\mathsf{b})\,\mathbf{U}\,\mathsf{c})$ are pure past, pure present, and pure future formulae, respectively.
We argue that separating formulae into ones that refer to the sole past, future, or present, allows for a bi-directional on-line analysis of sub-traces at activation time. The separation theorem, first introduced in [8], proves that such a separation can be obtained.

Theorem 1 (Separation theorem (adapted from [8])). *Any propositional temporal formula written with $\mathbf{U}$, $\mathbf{S}$, $\bigcirc$, and $\ominus$ operators can be rewritten as a boolean combination of pure temporal formulae.*

The constructive proof of Theorem 1 in [8] provides a syntactic procedure to pull out $\ominus$, $\mathbf{S}$ from the scope of $\bigcirc$, $\mathbf{U}$ in LTLp$_f$ formulae, and vice-versa. It thus provides the base substitution rules such that their recursive application brings to the decomposition of a LTLp$_f$ formula in pure temporal formulae. We capture this notion as follows.

Definition 2 (Temporal separation, separated formula). *Let φ be a LTLp$_f$ formula over Σ. A* temporal separation *is a function $\mathscr{S} : LTLp_f \rightarrow 2^{LTLp_f \times LTLp_f \times LTLp_f}$. Indicating*

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

$\varphi^{\blacktriangleleft}$, $\varphi^{\blacktriangledown}$, and $\varphi^{\blacktriangleright}$ respectively as pure past, present, and future formulae, defined over Σ as per Def. 1, $\mathscr{S}(\varphi) = \{(\varphi^{\blacktriangleleft},\varphi^{\blacktriangledown},\varphi^{\blacktriangleright})_1,\ldots,(\varphi^{\blacktriangleleft},\varphi^{\blacktriangledown},\varphi^{\blacktriangleright})_m\}$ is such that

$$\varphi \equiv \bigvee_{j=1}^{m} (\varphi^{\blacktriangleleft} \wedge \varphi^{\blacktriangledown} \wedge \varphi^{\blacktriangleright})_j.$$

We call separated formula *any element in the co-domain of $\mathscr{S}(\varphi)$.*

For example, $(\ominus \mathsf{b} \vee \Diamond \mathsf{c}) \equiv ((\ominus \mathsf{b}) \wedge (\mathit{True}) \wedge (\mathit{True})) \bigvee ((\mathit{True}) \wedge (\mathit{True}) \wedge (\Diamond \mathsf{c}))$.

Automata. A (deterministic finite-state) automaton is a rooted finite-state labelled transition system $A = (\Sigma, S, \delta, s_0, S_F) \in \mathcal{A}$, where: Σ is the alphabet; S is a finite non-empty set of states; $\delta : S \times \Sigma \to S$ is a (total) transition function; s_0 is the initial state; $S_F \subseteq S$ is the set of accepting states. An automaton *accepts* a trace $t = \langle e_1,\ldots,e_n \rangle$ if a walk of tuples $\langle (s_0,e_1,s_1),\ldots,(s_{n-1},e_n,s_n) \rangle$, namely a *replay*, exists such that $s_i = \delta(s_{i-1},e_i)$ for $1 \leqslant i \leqslant n$ and $s_n \in S_F$. We shall name s_i in tuple (s_{i-1},e_i,s_i) as *current state* of the replay at instant i. The set of all traces accepted by A is named *language* of A, denoted as $\mathscr{L}(A)$. Figure 1 depicts an automaton whose language consists of all traces of length $n \geqslant 2$ having b as its second event. Considering the event log of Ex. 1, it accepts only $t_4 = \langle \mathsf{d},\mathsf{b},\mathsf{a},\mathsf{e} \rangle$. Reportedly approaches exist that build automata verifying any formula of Linear Temporal Logic (LTL) [26], Linear-time Temporal Logic with Past (LTLp) [24,9], or LTL$_f$ [2] on traces: such automata accept all and only the traces such that all events satisfy a formula φ. We indicate them as A_φ. The automaton of Fig. 1, e.g., verifies $\bigcirc \mathsf{b}$. Notice that automata verification considers whole traces as either satisfying or violating a formula.

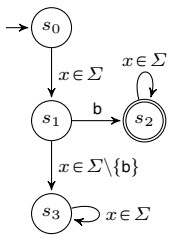


Fig. 1: An automaton verifying $\bigcirc \mathsf{b}$

3 The Janus approach

Here we present the concepts upon which our approach is built, beginning with the core notion of Reactive Constraints (RCons). RCons are meant to bear the interestingness semantics, because the role of activation is singled out from the rest of exerted conditions: only if the activation α "triggers" the constraint *and* a LTLp$_f$ formula φ is verified on the trace, then its fulfilment is interesting.

Definition 3 (Reactive Constraint (RCon)). *Given an alphabet Σ, let $\alpha \in \Sigma \cup \{t_{Start}, t_{End}\}$ be an* activation*, and φ be a LTLp$_f$ formula over Σ. A* Reactive Constraint (RCon) *Ψ is a pair (α,φ) hereafter denoted as $\Psi \triangleq \alpha \boxdot\!\!\rightarrow \varphi$. We denote the set of all RCons over Σ as $\mathscr{R}$.*

In the following, we will assume traces, automata, LTLp$_f$ formulas and RCons to be all defined over a shared alphabet Σ. Constraints of declarative process languages such as DECLARE can be expressed as RCons, as shown in Table 1. For example, the RCon corresponding to PRECEDENCE(d,a) is $\mathsf{a} \boxdot\!\!\rightarrow \Diamond \mathsf{d}$. Activations in constraints are identified according to the classification of [5]. RCons can include non-standard DECLARE constraints such as $\mathsf{a} \boxdot\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$, which imposes that if an event a occurs in a trace, either b

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

immediately precedes it, or c eventually occurs after. We say that a *activates* the RCon. We remark that although φ is a LTLp$_f$ formula, semantics of RCons detach from classical LTLp$_f$ in that every occurrence of α triggers a new verification of φ on the trace.

Definition 4 (Activator, triggering trace). *Given a trace $t \in \Sigma^*$ of length n and an instant i s.t. $1 \leqslant i \leqslant n$, an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$ is* activated *at i if $t,i \models \alpha$. Event $t(i)$ is then named* activator *of Ψ. A trace in which at least an activator of Ψ exists, is* triggering *for Ψ.*

Consider, e.g., the event log from Ex. 1 and $\Psi = \mathsf{a} \boxdot\!\!\rightarrow \Diamond\mathsf{d}$, i.e., PRECEDENCE(d,a) in DECLARE. Ψ is activated in trace $t_4 = \langle \mathsf{d},\mathsf{b},\mathsf{a},\mathsf{e} \rangle$ by $t_4(3)$; in trace $t_5 = \langle \mathsf{a},\mathsf{d},\mathsf{a},\mathsf{c},\mathsf{a} \rangle$ it is activated by $t_5(1)$, $t_5(3)$, and $t_5(5)$; in trace $t_6 = \langle \mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e} \rangle$ it is never activated; in trace $t_7 = \langle \mathsf{b},\mathsf{c},\mathsf{a} \rangle$ is activated by $t_7(3)$. Therefore t_4,t_5, and t_7 are triggering for Ψ.

Definition 5 (Interesting fulfilment). *Given a trace $t \in \Sigma^*$ of length n, an instant i s.t. $1 \leqslant i \leqslant n$, and an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$, Ψ is* interestingly fulfilled *at i if $t,i \models \alpha$ and $t,i \models \varphi$. The RCon is* violated *at instant i if $t,i \models \alpha$ and $t,i \not\models \varphi$. Otherwise, the RCon is* unaffected.

Consider again Ex. 1 and $\Psi = \mathsf{a} \boxdot\!\!\rightarrow \Diamond\mathsf{d}$. In trace t_4 the RCon is interestingly fulfilled by $t_4(3)$; in trace t_5 it is interestingly fulfilled by $t_5(3)$ and $t_5(5)$, and violated by $t_5(1)$; in trace t_6 it is unaffected at all instants, i.e., neither interestingly fulfilled nor violated; in trace t_7 it is violated by $t_7(3)$. We remark that instants at which a DECLARE constraint is satisfied can be such that the corresponding RCon is unaffected, i.e., when $t,i \models \varphi$ but $t,i \not\models \alpha$.

3.1 Measuring the interesting fulfilments of a Reactive Constraint on a log

Because each activator triggers a new check for the RCon, the degree of interestingness of a trace is analysed at the level of events as follows.

Definition 6 (Interestingness degree). *Given a trace $t \in \Sigma^*$ and an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$, we define the* interestingness degree *function $\zeta : \mathscr{R} \times \Sigma^* \rightarrow [0,1] \subseteq \mathbb{R}$ as follows:*

$$\zeta(\Psi,t) = \begin{cases} \dfrac{|\{i : t,i \models \alpha \text{ and } t,i \models \varphi\}|}{|\{i : t,i \models \alpha\}|} & \text{if } |\{i : t,i \models \alpha\}| \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively the interestingness degree measures the percentage of activations leading to (interesting) fulfilment within the trace. For instance, the interestingness degree of $\Psi = \mathsf{a} \boxdot\!\!\rightarrow \Diamond\mathsf{d}$ in $t_5 = \langle \mathsf{a},\mathsf{d},\mathsf{a},\mathsf{c},\mathsf{a} \rangle$ is $\zeta(\Psi,t_5) = 0.667$, because it is interestingly fulfilled by 2 activators out of 3. All the 20 activators of Ψ in $t_3 = \langle \mathsf{a},\mathsf{d},\mathsf{a},\ldots,\mathsf{a},\mathsf{c} \rangle$ lead to interesting fulfilment, except one (the first a event). Therefore $\zeta(\Psi,t_3) = 0.950$.

Next, we introduce the computation of two measures for the whole event log, adapting the classical definition of [1]: *support* measures how often the constraint is (interestingly) satisfied in the whole event log; *confidence* quantifies how the constraint is satisfied in the triggering traces in the event log.

Definition 7 (Support). *Given an event log $L = \{\!\{t_1,t_2,\ldots,t_m\}\!\} \in \mathbb{M}(\Sigma^*)$, and an RCon $\Psi \in \mathscr{R}$, we define as* support *of Ψ on L the function $\sigma : \mathscr{R} \times \mathbb{M}(\Sigma^*) \rightarrow [0,1] \subseteq \mathbb{R}$ calculated as the average of interestingness degree values of Ψ over all traces $t_j \in L$, with $1 \leqslant j \leqslant m$:*

$$\sigma(\Psi,L) = \begin{cases} \frac{\sum_{j=1}^{m} \zeta(\Psi,t_j)}{|L|} & \text{if } |L| > 0; \\ 0 & \text{otherwise.} \end{cases}$$

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

Table 2: Interestingness degree, support, and confidence of RCons on an example log

			Precedence(d,a)			a □→ (⊖b ∨ ◇c)		
	Trace	**Multi.**	**Activ.'s**	**Int.fulfil.'s**	ζ	**Activ.'s**	**Int.fulfil.'s**	ζ
t_1	⟨d,f,a,f,c,a,f,b,a,f⟩	25	3	3	1	3	2	0.667
t_2	⟨f,e,d,c,b,a,g,h,i⟩	15	1	1	1	1	1	1
t_3	⟨a,d,a,a,…,a,a,c⟩	10	20	19	0.950	20	20	1
t_4	⟨d,b,a,e⟩	20	1	1	1	1	1	1
t_5	⟨a,d,a,c,a⟩	5	3	2	0.667	3	2	0.667
t_6	⟨b,c,d,e⟩	20	0	0	0	0	0	0
t_7	⟨b,c,a⟩	5	1	0	0	1	0	0
	Support				0.728			0.650
	Confidence				0.910			0.813

Definition 8 (Confidence). *Given an event log* $L = \{t_1,t_2,...,t_m\} \in \mathbb{M}(\Sigma^*)$ *and an RCon* $\Psi \in \mathscr{R}$*, let* $\widetilde{L} = \{\widetilde{t_1},\widetilde{t_2},...,\widetilde{t_p}\}$ *with* $1 \leqslant p \leqslant m$ *be the portion of* L *that consists of all the traces triggering* Ψ*. We define the confidence of* Ψ *on* L *the function* $\kappa : \mathscr{R} \times \mathbb{M}(\Sigma^*) \to [0,1] \subseteq \mathbb{R}$ *as the average of interestingness degree values of* Ψ *over the* triggering *traces* $\widetilde{t_j} \in \widetilde{L}$*:*

$$\kappa(\Psi,L) = \begin{cases} \frac{\sum_{j=1}^{p} \zeta(\Psi,\widetilde{t_j})}{|\widetilde{L}|} & \text{if } |\widetilde{L}| > 0; \\ 0 & \text{otherwise.} \end{cases}$$

As seen above, e.g., the interestingness degree of the RCon of Precedence(d, a), $\Psi = \mathsf{a} \,\Box\!\!\rightarrow\, \Diamond\mathsf{d}$, is $\zeta(\Psi, t_5) = \frac{2}{3} = 0.667$. Averaging the interestingness degree of Precedence(d, a) on all traces (including their multiplicities), we obtain the support $\sigma(\Psi,L)$, which amounts to $\frac{(1\times25)+(1\times15)+(0.95\times10)+(1\times20)+(0.67\times5)+(0\times20)+(0\times5)}{100} = 0.728$. The value of Confidence $\kappa(\Psi,L)$ is $\frac{(1\times25)+(1\times15)+(0.95\times10)+(1\times20)+(0.67\times5)+(0\times5)}{80} = 0.910$, thus higher than support, because Ψ is not activated in trace t_6, hence the lower denominator. Table 2 shows in detail the interestingness degree, support, and confidence of constraints Precedence(d,a) and $\mathsf{a} \,\Box\!\!\rightarrow\, (\ominus\mathsf{b} \vee \Diamond\mathsf{c})$ on the event log seen in Ex. 1.

Considering each activation independently to compute interestingness degree, support and confidence, allows for higher resilience to noise in event logs. This is particularly evident in t_3, in which 19 occurrences of a, the activation, are preceded by d, thus fulfilling the constraint. Only the first a leads to violation. Considering the whole trace as fulfilling or not, as in [18,19], would lead to an interestingness degree of 0, instead of 0.95, thus decreasing support and confidence too. Because the constraints returned by discovery techniques are those that have a support and confidence above user-defined thresholds, that could lead to a loss of information.

3.2 An automata approach to Reactive Constraint verification

Once an RCon $\Psi = \alpha \,\Box\!\!\rightarrow\, \varphi$ is activated, its fulfilment relies on the verification of the LTLp$_f$ formula φ at the instant of activation. As seen in Theorem 1 it is possible to separate a LTLp$_f$ formula into sub-formulae, each containing either only past, only future, or no

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

temporal operators. Therefore its verification can be decoupled by splitting the trace in two independent sub-traces: one from the beginning to the activator, with which the sole past operators are verified, and one from the activator on, concerning only future operators.

Lemma 1. *Given a pure past formula $\varphi^{\blacktriangleleft}$, a pure present formula $\varphi^{\blacktriangledown}$, a pure future formula $\varphi^{\blacktriangleright}$, a trace $t \in \Sigma^*$ of length n and an instant i s.t. $1 \leqslant i \leqslant n$, the following hold true:*
$t,i \models \varphi^{\blacktriangleleft} \equiv t_{[1,i]},i \models \varphi^{\blacktriangleleft}$; $t,i \models \varphi^{\blacktriangledown} \equiv t_{[i,i]},i \models \varphi^{\blacktriangledown}$; $t,i \models \varphi^{\blacktriangleright} \equiv t_{[i,n]},i \models \varphi^{\blacktriangleright}$.

The lemma follows from definition of LTLp$_f$ semantics. For example, evaluating $\varphi^{\blacktriangleleft} = \Diamond\!\!\!-\mathsf{d}$ on $t_2 = \langle \mathsf{f,e,d,c,b,a,g,h,i} \rangle$ from Ex. 1 at instant $i = 6$ is equivalent to evaluating it on $t_{2[1:6]} = \langle \mathsf{f,e,d,c,b,a} \rangle$ at i, because $\varphi^{\blacktriangleleft}$ concerns only the prefix of t_2. Instead $\varphi^{\blacktriangleright} = \Diamond \mathsf{c}$ at i concerns only the suffix, $t_2[6:9] = \langle \mathsf{a,g,h,i} \rangle$.

Theorem 2 (Trace sub-valuation). *Given a LTLp$_f$ formula φ, a trace $t \in \Sigma^*$ of length n and an instant i s.t. $1 \leqslant i \leqslant n$, we have that $t,i \models \varphi$ if and only if $t_{[1:i]},i \models \varphi^{\blacktriangleleft}$, $t_{[i:i]},i \models \varphi^{\blacktriangledown}$, and $t_{[i:n]},i \models \varphi^{\blacktriangleright}$ for at least a $(\varphi^{\blacktriangleleft},\varphi^{\blacktriangledown},\varphi^{\blacktriangleright}) \in \mathscr{S}(\varphi)$.*

The proof follows from Theorem 1 and Lemma 1. Consider, e.g., the RCon $\Psi = \mathsf{a} \boxdot\!\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$. It follows that $\varphi = \ominus \mathsf{b} \vee \Diamond \mathsf{c}$ and $\mathscr{S}(\varphi) = \{(\ominus \mathsf{b}, \mathit{True}, \mathit{True}), (\mathit{True}, \mathit{True}, \Diamond \mathsf{c})\}$. Applying Theorem 2 on $t_2 = \langle \mathsf{f,e,d,c,b,a,g,h,i} \rangle$ from Ex. 1, we have that $t_2,6 \models \varphi$ if *(i)* $\langle \mathsf{f,e,d,c,b,a} \rangle,6 \models \ominus \mathsf{b}$ or *(ii)* $\langle \mathsf{a,g,h,i} \rangle,6 \models \Diamond \mathsf{c}$, aside of *True* formulae which are trivially satisfied. Because the first holds true, we conclude that φ is satisfied by $t_2(6)$.

As seen in Section 2, a formula φ can be verified on a trace t by checking whether t is accepted by automaton A_φ. Given a LTLp$_f$ formula φ and its temporal separation $\mathscr{S}(\varphi)$, we thus introduce the notion of separated automata set, namely a set of triples of automata, each verifying a triple of pure temporal formulae in $\mathscr{S}(\varphi)$.

Definition 9 (Separated automata set (sep.aut.set)). *Given a LTLp$_f$ formula φ we define as* separated automata set (sep.aut.set) $\mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright} \in 2^{\mathcal{A}\times\mathcal{A}\times\mathcal{A}}$ *the* set of triples $A^{\blacktriangleleft\blacktriangledown\blacktriangleright} = (A^{\blacktriangleleft}, A^{\blacktriangledown}, A^{\blacktriangleright}) \in \mathcal{A} \times \mathcal{A} \times \mathcal{A}$ *such that $A^{\blacktriangleleft} = A_{\varphi^{\blacktriangleleft}}$, $A^{\blacktriangledown} = A_{\varphi^{\blacktriangledown}}$, and $A^{\blacktriangleright} = A_{\varphi^{\blacktriangleright}}$ for every $(\varphi^{\blacktriangleleft},\varphi^{\blacktriangledown},\varphi^{\blacktriangleright}) \in \mathscr{S}(\varphi)$. We denote as* separation degree *D the number of triples of $\mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$.*

Considering the latest example $A^{\blacktriangleleft\blacktriangledown\blacktriangleright} = \{(A_{\ominus \mathsf{b}}, A_{\mathit{True}}, A_{\mathit{True}}), (A_{\mathit{True}}, A_{\mathit{True}}, A_{\Diamond \mathsf{c}})\}$, the separation degree is 2. Table 1 lists the separation degrees of some RCons of DECLARE. In light of the above, we derive from Theorem 2 the following.

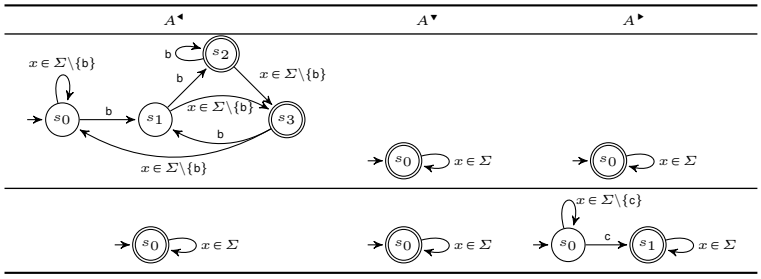
Theorem 3 (Trace sub-valuation through automata). *Given a LTLp$_f$ formula φ, its sep.aut.set $A^{\blacktriangleleft\blacktriangledown\blacktriangleright}$, a trace $t \in \Sigma^*$ of length n, and an instant i s.t. $1 \leqslant i \leqslant n$, we have that $t,i \models \varphi$ if and only if $t_{[1:i]} \in \mathscr{L}(A^{\blacktriangleleft})$, $t_{[i:i]} \in \mathscr{L}(A^{\blacktriangledown})$ and $t_{[i:n]} \in \mathscr{L}(A^{\blacktriangleright})$ for at least a $(A^{\blacktriangleleft},A^{\blacktriangledown},A^{\blacktriangleright}) \in A^{\blacktriangleleft\blacktriangledown\blacktriangleright}$.*

In the example, the application of Theorem 3 entails that $\mathsf{a} \boxdot\!\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$ is interestingly fulfilled by $t_2(6)$ if $t_{2[1:6]} = \langle \mathsf{f,e,d,c,b,a} \rangle \in \mathscr{L}(A_{\ominus \mathsf{b}})$ or $t_{2[6:9]} = \langle \mathsf{a,g,h,i} \rangle \in \mathscr{L}(A_{\Diamond \mathsf{c}})$.

Past automata reversion. To the best of our knowledge, there is no available technique to build automata that verify LTLp$_f$ formulae. We thus exploit Theorem 2 to rely on the readily available techniques for LTL$_f$, i.e., without past operators, as described in [2]. To this extent, we rely on mirror images and reversed automata.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

Table 3: Graphical representation of the separated automata set of $\mathsf{a} \boxdot\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$



Lemma 2. *Let $t \in \Sigma^*$ be a trace of length n and t_R its reverse. Given a pure past formula $\varphi^{\blacktriangleleft}$, and its mirror image $\varphi^{\blacktriangleleft}_M$, we have that $t,n \models \varphi^{\blacktriangleleft}$ iff $t_R,1 \models \varphi^{\blacktriangleleft}_M$.*

The proof follows from the semantics of future and past operators of LTLp$_f$ provided in Section 2. For instance, verifying $\varphi^{\blacktriangleleft} = \Diamond \mathsf{d}$ on $t_2 = \langle \mathsf{f,e,d,c,b,a,g,h,i} \rangle$ from Ex. 1 at instant $i = 9$ is equivalent to verifying $\varphi^{\blacktriangleleft}_{\mathrm{M}} = \Diamond \mathsf{d}$ on $t_{2R} = \langle \mathsf{i,h,g,a,b,c,d,e,f} \rangle$ at $i = 1$. Notice that this holds for sub-traces too, thus verifying $\varphi^{\blacktriangleleft}$ on t_2 at instant $i = 6$ is equivalent to verifying $\varphi^{\blacktriangleleft}_{\mathrm{M}}$ over $t_{2[1:6]R} = \langle \mathsf{a,b,c,d,e,f} \rangle$ at $i = 1$ in the light of Lemma 1.
It follows from Lemma 2 that any pure past formula can be seen as a pure future one on a reversed trace. Therefore the automaton verifying the mirror image of $\varphi^{\blacktriangleleft}$ can be used for verification on the reversed trace, as stated in the following.

Corollary 1. *Let $A_{\varphi^{\blacktriangleleft}_M}$ be the automaton verifying $\varphi^{\blacktriangleleft}_M$. Then $t,n \models \varphi^{\blacktriangleleft}$ iff $t_R \in \mathscr{L}\left(A_{\varphi^{\blacktriangleleft}_M}\right)$.*

Notice that $\varphi^{\blacktriangleleft}_{\mathrm{M}}$ is a pure future formula, therefore $A_{\varphi^{\blacktriangleleft}_{\mathrm{M}}}$ can be built by applying the technique of [2] for LTL$_f$. Furthermore, it is possible to transform the obtained automaton in order to read directly the original trace t thanks to the property of *closure under reversion* of regular languages [13].

Definition 10 (Reversed automaton [13]). *Given a trace $t \in \Sigma^*$, its reverse t_R, and the automaton $A \in \mathcal{A}$, the reversed automaton $\overleftarrow{A} \in \mathcal{A}$ is an automaton such that A accepts t if and only if $\overleftarrow{A}$ accepts t_R, i.e., $t \in \mathscr{L}(A)$ iff $t_R \in \mathscr{L}\left(\overleftarrow{A}\right)$.*

From Lemma 2 and Corollary 1 we derive the following.

Theorem 4 (Valuation through reversed automaton of mirror image). *Let $\varphi^{\blacktriangleleft}$ be a pure past formula and $\varphi^{\blacktriangleleft}_M$ its mirror image. Let $A_{\varphi^{\blacktriangleleft}_M} \in \mathcal{A}$ be the automaton verifying $\varphi^{\blacktriangleleft}_M$. Given a trace $t \in \Sigma^*$ of length n, we have that: $t,n \models \varphi^{\blacktriangleleft}$ iff $t \in \mathscr{L}\left(\overleftarrow{A}_{\varphi^{\blacktriangleleft}_M}\right)$.*

Consider the RCon $\Psi = \mathsf{a} \boxdot\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$ and the pure past formula of its sep.aut.set $\varphi^{\blacktriangleleft} = \ominus \mathsf{b}$. It is activated by $t_2(6)$. Its mirror image is $\varphi^{\blacktriangleleft}_{\mathrm{M}} = \bigcirc \mathsf{b}$. With automaton $A_{\varphi^{\blacktriangleleft}_{\mathrm{M}}}$, depicted in Fig. 1, we can verify $\varphi^{\blacktriangleleft}_{\mathrm{M}}$ over trace $t_{2[1:6]R} = \langle \mathsf{a,b,c,d,e,f} \rangle$ at $i = 1$, thereby verifying $\varphi^{\blacktriangleleft}$ over $t_{2[1:6]}$ as per Lemma 2. Thanks to Theorem 4, $\varphi^{\blacktriangleleft}_{\mathrm{M}}$ can be verified on $t_{2[1:6]}$ with the reversed automaton $\overleftarrow{A}_{\varphi^{\blacktriangleleft}_{\mathrm{M}}}$, depicted in the top-left corner of Table 3.
We remark that in this way the pure past formulae of sep.aut.sets can be verified by parsing sub-traces from the beginning of the trace till the activator event.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

Algorithm 1: Computing the interestingness degree of an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$ on trace t, given the sep.aut.set $\mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$ of φ

```
1  O ← empty bag ;
2  foreach event t(i) ∈ t do
3      foreach A◂ ∈ A◂▾▸ do perform transition t(i) on A◂ ;
4      if t(i) = α then                                              // Activation triggered
5          J ← empty set of pairs ;        // J stores the replay-state of future automata for one activation
6          foreach (A◂, A▾, A▸) ∈ A◂▾▸ do
7              if A◂ is in an accepting state and A▾ accepts ⟨t(i)⟩ then
8                  Take A▸ = (Σ, S▸, δ▸, s0▸, SF▸) and add (s0▸, A▸) to J
9          add J to O;                                   // O collects replay-states for all activations
10     foreach J ∈ O do
11         foreach (s▸, A▸) ∈ J do s▸ ← δ▸(s▸, t(i))          // Perform t(i) on A▸ and save state ;
12 if |O| > 0 then return |{J∈O : at least a (s▸, A▸)∈J is s.t. s▸∈SF▸}| / |O| else return 0 ;
```

4 The Janus algorithm

Algorithm 1 shows the pseudo-code of our on-line technique to compute the interestingness degree $\zeta(\Psi,t)$ of a RCon Ψ with respect to a trace t. Its fundamental data structure is a set of pairs, each associating an automaton to its current state, as the replay of the trace proceeds. We call it *Janus state* and denote it as $\mathcal{J}$.

More specifically, only future automata of sep.aut.sets are considered. The naïve approach would indeed parse trace t, and apply the check of Theorem 3 whenever Ψ is activated. This is an impractical solution, because it requires the replay of prefix $t_{[1:i]}$ and suffix $t_{[i:n]}$ on the respective automata at every instant of activation i. We save computation time by keeping track of the past valuation state, so that at each activation it is already known, thus improving on the sub-valuation of pure past formulae $\varphi^{\blacktriangleleft}$. We rely on the fact that automata preserve the history of replays in the reached state: if at instant i the current state of $A^{\blacktriangleleft}$ is s_i, then at $i+1$ it is known that $s_{i+1} = \delta(s_i, t(i+1))$. To this extent, the algorithm requires all past automata $A^{\blacktriangleleft}$ to be already reversed as per Theorem 4.

The runtime of the algorithm will be explained considering as input: *(i)* $t = t_1 = \langle \mathsf{d},\mathsf{f},\mathsf{a},\mathsf{f},\mathsf{c},\mathsf{a},\mathsf{f},\mathsf{b},\mathsf{a},\mathsf{f}\rangle$ from Ex. 1, *(ii)* $\Psi = \mathsf{a} \boxdot\!\!\rightarrow (\ominus\mathsf{b} \vee \Diamond\mathsf{c})$, and *(iii)* the sep.aut.set $\left\{\left(A^{\blacktriangleleft}_{\ominus\mathsf{b}}, A^{\blacktriangledown}_{True}, A^{\blacktriangleright}_{True}\right), \left(A^{\blacktriangleleft}_{True}, A^{\blacktriangledown}_{True}, A^{\blacktriangleright}_{\Diamond\mathsf{c}}\right)\right\}$ of φ, with past automaton $A^{\blacktriangleleft}_{\ominus\mathsf{b}}$ already reversed as depicted in Table 3. Let i denote the current instant in the following.

At lines 1–2, a bag of Janus states $\mathcal{O}$ is initialised, to store the states of current replays for the verification of pure future formulae. Every replay is triggered by the occurrence of an activation, thus every Janus state refers to an activator. Thereupon, trace t is parsed one event at a time starting with the left-first one, e.g., $t(1) = \mathsf{d}$. We remark that no knowledge is assumed on the subsequent events of the trace, as per the on-line setting. At line 3, past automata replay t performing each transition $t(i)$ as it is read, i.e., not waiting for the activation to occur. By contrast, future automata will begin with independent replays at each occurrence of an activator. Therefore every $A^{\blacktriangleleft}$ automaton starts a replay from $i = 1$. At line 4, the activator is captured, as, e.g., when i is equal to 3, 6, and 9, i.e., when $t(i) = \mathsf{a}$. Consequently at line 5 a new Janus state $\mathcal{J}$ is initialised to store information on the new replay. At line 7 it is checked that, for every triple in the sep.aut.set, *(i)* $A^{\blacktriangleleft}$ is in an

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

accepting state, and *(ii)* $A^{\blacktriangledown}$ accepts the trace made of the current event. If this is the case the replay on the future automaton of the triple, $A^{\blacktriangleright}$, can start. At line 8 the pair consisting of $A^{\blacktriangleright}$ and its initial state $s_0^{\blacktriangleright}$ is added to $\mathcal{J}$. In the example, the triple $\left(A^{\blacktriangleleft}_{\ominus \mathsf{b}}, A^{\blacktriangledown}_{\mathit{True}}, A^{\blacktriangleright}_{\mathit{True}}\right)$ at $i=3$ and $i=6$ has $A^{\blacktriangleleft}_{\ominus \mathsf{b}}$ not accepting the prefix $t_{[1:i]}$ because $t(i-1)\neq \mathsf{b}$. On the contrary, the replay of $A^{\blacktriangleright}$ can start at $i=9$. For what the triple $\left(A^{\blacktriangleleft}_{\mathit{True}}, A^{\blacktriangledown}_{\mathit{True}}, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)$ is concerned, $A^{\blacktriangleright}_{\Diamond \mathsf{c}}$ always starts a new replay because $A^{\blacktriangleleft}_{\mathit{True}}$ and $A^{\blacktriangledown}_{\mathit{True}}$ accept any trace.
We remark that for $A^{\blacktriangledown}$ no state is retained because the scope of a pure present formula is limited to a single event. Notice that because every triple $(A^{\blacktriangleleft}, A^{\blacktriangledown}, A^{\blacktriangleright}) \in \mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$ represents a conjunctive formula, if $A^{\blacktriangleleft}$ is not in an accepting state then the activation leads the entire triple to violation, regardless of the replay on $A^{\blacktriangleright}$.
The new Janus state $\mathcal{J}$ is added to $\mathcal{O}$ at line 9. In the example, at $i=9$, $\mathcal{O} = \left\{\left\{\left(s_1, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}, \left\{\left(s_0, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}, \left\{\left(s_0, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right), \left(s_0, A^{\blacktriangleright}_{\mathit{True}}\right)\right\}\right\}$. At lines 10–11 all states in every $\mathcal{J} \in \mathcal{O}$ are updated by executing the current transition on their respective future automata. For example, upon the reading of $t(5)=\mathsf{c}$, $\left\{\left(s_0, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}$ is updated to $\left\{\left(s_1, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}$.

An event in a trace interestingly fulfils an RCon if, upon activation, at least a triple $(A^{\blacktriangleleft}, A^{\blacktriangledown}, A^{\blacktriangleright})$ is such that $A^{\blacktriangleleft}$, $A^{\blacktriangledown}$, and $A^{\blacktriangleright}$ all accept the respective sub-traces, as per Theorem 3. By construction, this holds true if at least a future automaton in $\mathcal{J}$ is in its accepting state at the end of the replay. To measure the interestingness degree of the RCon, we thus compute the ratio between the number of all such Janus states and the cardinality of $\mathcal{O}$ at line 12. For instance, at $i=10$, $\mathcal{O} = \{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3\} = \left\{\left\{\left(s_1, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}, \left\{\left(s_0, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}, \left\{\left(s_0 A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right), \left(s_0, A^{\blacktriangleright}_{\mathit{True}}\right)\right\}\right\}$ and $|\mathcal{O}|=3$. Only $\mathcal{J}_1$ and $\mathcal{J}_3$ contain automata in accepting states, therefore $\zeta(\Psi, t) = \frac{2}{3} = 0.667$.

Computational Cost. To compute the asymptotic computational cost, we consider the worst case scenario, occurring when at each instant *(i)* Ψ is activated, and *(ii)* all past and present automata are in an accepting state. In such a case, given an event log L, a trace t of length t, an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$ for which the sep.aut.set $\mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$ of φ is generated with separation degree D, the cost of verifying Ψ on t is:

$$\sum_{j=1}^{n} (D + (n-j)D) = nD + D\sum_{j=1}^{n}(n-j) = Dn\left(1 + \frac{(n-1)}{2}\right).$$

The cost is linear in the number of activations, as each one requires a single replay of the trace for every automaton. For each activation only trace suffixes are replayed, owing to our optimisation over past formulae, hence the $1/2$ factor. For $D \ll n$, the upper-bound is $O(n^2)$, which is comparable to state-of-the-art techniques [6,19]. Because every trace of L is parsed singularly, the cost is $O(|L|)$. Finally, denoting as m the maximum amount of parameters of a template, the cost is $O(|\Sigma|^m)$ because constraints are verified for every permutation of symbols in alphabet Σ. For standard DECLARE, e.g., $m=2$.

5 Evaluation

A proof-of-concept implementation of our technique has been developed for experimentation. It is available for download at `https://github.com/Oneiroe/Janus`.

Pre-print copy of the manuscript published by Springer (available at `link.springer.com`)
identified by DOI: `10.1007/978-3-319-98648-7_8`

Table 4: Characteristics of declarative process mining approaches

	Extendibility	On-line	Interestingness	Granularity
Declare Maps Miner [18]	✓	✓	Ad hoc for DECLARE	Traces
Declare Miner 2 [19]	×	×	Ad hoc for DECLARE	Traces
MINERful [6]	×	×	×	Events over log
MINERful Vacuity Checker [20]	✓	✓	Vacuity	Traces
Janus	✓	✓	✓	Events over traces

We compare Janus to other declarative process mining techniques, highlighting specific properties and scenarios through synthetic logs. Thereafter, we evaluate our tool against a real-world event log and compare the output to a reference model.

5.1 State-of-the-art declarative process mining approaches

The following state-of-the-art declarative process discovery algorithms have been considered for comparison: *(i)* Declare Maps Miner (DMM) [18], the first declarative process miner, based on the replay of traces on automata; *(ii)* Declare Miner 2 (DM2) [19], adopting DECLARE-specific heuristics to improve on original DMM performance; *(iii)* MINERful (Mf) [6], building a knowledge base of task co-occurrence statistics, so as to mine DECLARE constraints via queries; *(iv)* MINERful Vacuity Checker (Mf-Vchk) [20], extending Mf to include semantical vacuity detection. Criteria for comparison reflect the goals of our research. They are: *(i) extendibility* over custom templates beyond standard DECLARE, *(ii)* capability of performing the analysis *on-line*, *(iii)* characterisation of constraint *interestingness*, and *(iv) granularity* of support and confidence measures with respect to the event log. Table 4 summarizes the outcome of our comparison.

Extendibility. DECLARE has been introduced as a language to be extended by users with the addition of custom templates [23]. DMM allows indeed for the check of any LTL_f formula. On the other hand Mf and DM2 work only with the DECLARE standard template set. Any new constraint outside this scope needs to be hard-coded into the framework. Janus allows for the check of any LTLp_f formula expressing an RCon.

On-line analysis. By design, only DMM and Mf-Vchk can be employed in *on-line* settings like run-time monitoring, as well as Janus, whilst DM2 and Mf operate off-line.

Interestingness. The core rationale of *interestingness* is to distinguish whether a constraint is not only satisfied but also activated in the trace. DMM and DM2 provide ad-hoc solutions only for the DECLARE standard template set, because for each template the activation condition is hard-coded. Mf checks only the satisfaction of constraints. Mf-Vchk instead relies on a semantical vacuity detection technique independent from the specific constraint or language. Nevertheless it provides misleading results with constraints involving implications in the past such as PRECEDENCE. In Janus, RCons are such that the activation is singled out in their formulation itself and its occurrence treated as a trigger in their semantics, so as to overcome those issues by design and address interestingness.

Granularity. DMM and DM2 calculate the support as the percentage of fully compliant traces. Similarly, Mf-Vchk calculates it as the percentage of traces that non-vacuously satisfy the constraint. Mf calculates support as the percentage of activations leading to constraint satisfaction over the entire event log, therefore the analysis is at the level of

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

Table 5: Results of experiments over synthetic logs for comparative evaluation

	False positives				Numerous activations in trace	Partial satisfaction
	Supp.	**Triggering tr.s**	**Discarded tr.s**	**Violating tr.s**	**Supp.**	**Supp.**
Mf	1.000	1000	0	0	0.848	0.941
Mf-VChk	0.881	881	119	0	0.100	0.875
DMM/DMM2	0.231	231	769	0	0.100	0.875
Janus	0.231	231	769	0	0.100	0.979

single events. Janus support mediates between them as it is computed as the average of interestingness degree values over the traces in the log.

5.2 Comparative evaluation over synthetic logs

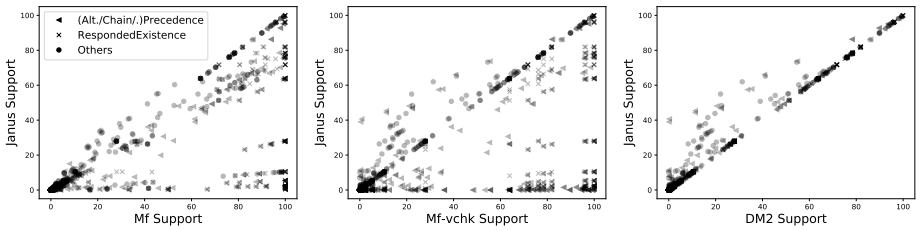
Synthetic logs have been constructed to show the behaviour of mining techniques in specific scenarios highlighting how the differences seen in Section 5.1 influence the discovery outcome. Table 5 summarizes the results.

False positives. Columns 2-5 of Table 5 show the results of an experiment conducted on a synthetic log of 1000 traces, simulating a model consisting only of PRECEDENCE(d,a). The event log, built with the MINERful log generator [4], has a high amount of non-triggering traces for that constraint: 231 traces contain a and d, 650 only d, 119 neither a nor d. The goal of this experiment is to show the distinction between interestingness and non-vacuity. Mf has natively no vacuity nor interestingness detection mechanisms. Both DMM and Janus recognise correctly the triggering traces, and discard the remaining ones when computing support. As said, the vacuity detection approach of Mf-Vchk shows instead misleading results with constraints involving a time-backward implication. In this case, it recognises as non-vacuous those traces that contain d, with or without any a. From a vacuity-detection perspective, it is logically correct because if d occurs, the constraint is satisfied regardless of the occurrence of a. Nevertheless, it is misleading because the activation of PRECEDENCE(d,a) is a, not d. Traces without a-events satisfy the constraint without any occurrence of the activation, and should not be considered as interesting – hence the name of the experiment, "False positives". Janus prevents false positives thanks to the semantics of Reactive Constraints.

Numerous activations in trace. Column 6 of Table 5 shows the support of PRECEDENCE(d,a) on event log $L = \{\!|t_1^1, t_2^9|\!\}$, with $t_1 = \langle \mathsf{da}...\mathsf{a} \rangle$ and $t_2 = \langle \mathsf{a} \rangle$. Trace t_1 satisfies the constraint and consists of a sequence of 50 a-events following a single d, but its multiplicity in the log is 1. Trace t_2 violates the constraint, and its multiplicity is 9. The goal is to show the misleading influence of the number of activations in a trace on support calculation. DMM, DM2, Mf-Vchk, and Janus are not influenced by the number of activations in t_1 because they compute support as an average over whole traces. Mf is instead highly influenced by t_1, because it contains the majority of the constraint activations, despite the higher multiplicity of t_2. We argue that the rate of interesting fulfilments per trace, not the their total amount in the event log, should be considered. Janus follows this rationale.

Partial satisfaction. Column 7 of Table 5 shows the support of PRECEDENCE(d,a) on event log $L = \{\!|t_1^1, t_2^4, t_3^3|\!\}$, with $t_1 = \langle \mathsf{a}, \mathsf{d}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a} \rangle$, $t_2 = \langle \mathsf{d}, \mathsf{a}, \mathsf{a} \rangle$, and $t_3 = \langle \mathsf{d}, \mathsf{a} \rangle$. Activations lead to fulfilment in all cases except the first event of t_1. Slight violations are

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8



(a) Mf (b) Mf-Vchk (c) DMM, DM2

Fig. 2: Comparison of support values between Janus and state-of-the-art algorithms

common in real-life logs, thus there is the risk of losing valuable information if entire traces are discarded for that. The goal is to show how the discovery techniques behave in such situations. Mf overcomes this issue owing to its support calculation at the granularity level of events. DMM and Mf-Vchk instead discard completely those traces because of the single violation. Janus assigns a lower interestingness degree to t_1, but does not discard it.

With synthetic event logs we have shown that Janus is capable of discerning interesting fulfilments, is resilient to noise, and balances the number of activations with the multiplicity of traces for support calculation. Next we show insights on the application of Janus on a real-world event log.

5.3 Evaluation on a real-life event log

Healthcare processes are known to be highly flexible [11], thus being suitable for declarative process modelling approaches [23]. The real-life log referred to as *Sepsis*[3] has been thus analysed. It reports the trajectories of patients showing symptoms of sepsis in a Dutch hospital. The model mined by Janus with support threshold equal to 10% and confidence threshold equal to 94% consists of the following constraints:

Init(ER Registration)	Alt.Precedence(ER Registration,ER Triage)	Alt.Precedence(Admission NC,Return ER)
Alt.Precedence(ER Triage,ER Sepsis Triage)	Alt.Precedence(ER Triage,Return ER)	Precedence(ER Triage,Admission NC)
RespondedExistence(IV Antibiotics,Lactic Acid)	Alt.Precedence(Admission IC,CRP)	Precedence(ER Triage,Admission IC)
RespondedExistence(IV Liquid,IV Antibiotics)	Alt.Precedence(Leucocytes,Release A)	Alt.Response(ER Registration,ER Triage)
RespondedExistence(IV Liquid,Lactic Acid)	Alt.Precedence(ER Triage,Release A)	Alt.Response(ER Registration,Leucocytes)
Precedence(ER Registration,CRP)	Alt.Precedence(CRP,Return ER)	Alt.Precedence(ER Sepsis Triage,IV Antibiotics)
Precedence(ER Registration,Leucocytes)	Alt.Precedence(Admission IC,Leucocytes)	Alt.Precedence(Leucocytes,Return ER)
Precedence(ER Registration,Admission IC)	Alt.Precedence(CRP,Release A)	Alt.Response(ER Triage,ER Sepsis Triage)
Alt.Response(ER Registration,CRP)		

We remark that all constraints comply with the normative process model reported in [21], which was designed iteratively with the help of domain experts. To check conformance of the mined model with the event log we have utilised the Declare Replayer plug-in [15], yielding fitness equal to 98%.

Figure 2 compares the output of Janus with the one of other declarative process mining techniques. Figure 2(a) shows that in absence of a dedicated mechanism, support is a metric that is not sensitive to interestingness. Indeed Mf assigns a support of 1.0 to several constraints to which Janus attributes a value ranging from 0.0 to 1.0, whereby

[3] https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-98648-7_8

the latter accounts for interestingness as shown throughout this paper. The semantical vacuity detection of Mf-Vchk partially solves this issue, as it can be noticed in Fig. 2(b). Still there are constraints assigned with a support of about 1.0 by Mf-Vchk against a value spanning over the whole range from 0.0 to 1.0 as per Janus. As expected, they are those constraints that have a time-backward component. For example, the support of ChainPrecedence(Leucocytes,Release C) on the log is 0.948 for Mf-Vchk, and 0.008 for Janus. However, other tasks than Leucocytes can immediately precede Release C as per the reference model of [21]. The constraint is indeed rarely activated, i.e., Release C occurs in 0.024% of the traces in the log, as opposed to Leucocytes, occurring in 0.964% of the traces and thus causing the misleading result of Mf-Vchk. Figure 2(c) shows that the Declare-tailored vacuity detection technique of DMM and DM2 prevents uninteresting constraints to be returned. Nevertheless, the support computation based solely on trace satisfaction of DMM and DM2 makes the assigned support be a lower bound for Janus.

The time taken by Janus to mine the event log amounted to 9s on a machine equipped with an Intel Core i5-7300U CPU at 2.60GHz, quad-core, Ubuntu 17.10 operating system. The timing is in line with existing approaches that tackle vacuity detection: 3s for DM2, 9s for Mf-Vchk, and 270s for DMM on the same machine.

6 Conclusion

In this paper, we have described Janus, an approach to model and discover Reactive Constraints on-line from event logs. With the Janus approach users can single out activations in the constraint expressions, namely the triggering conditions. Thereby interesting constraint fulfilments are discerned from the uninteresting ones by checking whether the activation occurred. Experimental evidence on synthetic and real-life event logs confirms its compatibility with previous Declare discovery techniques, yet improving on the capability of identifying interestingly fulfilled constraints.
For our research outlook we aim at enriching RCons by allowing for activations expressed as temporal logic formulae rather than propositional symbols, inspired by [3] and [17]. It is in our plans to integrate the MONA tool [10] for the automatic generation of automata for verification, as suggested in [27]. We will also study the application of Janus on run-time monitoring and off-line discovery tasks. Furthermore, future work will include the analysis of other declarative languages such as DCR graphs [11], and of multi-perspective approaches encompassing time, resources, and data. To that extent, we will employ metrics beyond support and confidence.
Acknowledgements. The work of Alessio Cecconi, Claudio Di Ciccio, and Jan Mendling has been funded by the Austrian FFG grant 861213 (CitySPIN). Giuseppe De Giacomo has been partially supported by the Sapienza project "Immersive Cognitive Environments".

References

1. Adamo, J.: Data mining for association rules and sequential patterns - sequential and parallel algorithms. Springer New York (2001)
2. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In: AAAI. pp. 1027–1033 (2014)

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-319-98648-7_8

3. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI. pp. 854–860. Association for Computing Machinery (2013)
4. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of Declare models. In: EOMAS. pp. 20–36. Springer (2015)
5. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Information Systems 64, 425–446 (2017)
6. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Transactions on Management Information Systems (TMIS) 5(4), 24 (2015)
7. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420. IEEE (1999)
8. Gabbay, D.: The declarative past and imperative future. In: Temporal logic in specification. pp. 409–448. Springer (1989)
9. Gastin, P., Oddoux, D.: LTL with past and two-way very-weak alternating automata. In: MFCS. pp. 439–448. Springer (2003)
10. Henriksen, J.G., Jensen, J.L., Jørgensen, M.E., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: MONA: Monadic Second-Order Logic in practice. In: TACAS. pp. 89–110 (1995)
11. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Declarative modelling and safe distribution of healthcare workflows. In: FHIES. pp. 39–56 (2011)
12. Hodkinson, I.M., Reynolds, M.: Separation-Past, Present, and Future. In: We Will Show Them!(2). pp. 117–142 (2005)
13. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Pearson/Addison Wesley, 3rd ed edn. (2007)
14. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. International Journal on Software Tools for Technology Transfer 4(2), 224–233 (2003)
15. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Information Systems 47, 258 – 277 (2015)
16. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Logics of Programs. pp. 196–218 (1985)
17. Ly, L.T., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: OTM. pp. 82–99. Springer (2011)
18. Maggi, F.M., Bose, R.J.C., van der Aalst, W.M.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. pp. 270–285. Springer (2012)
19. Maggi, F.M., Di Ciccio, C., Di Francescomarino, C., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Information Systems (2018)
20. Maggi, F.M., Montali, M., Di Ciccio, C., Mendling, J.: Semantical Vacuity Detection in Declarative Process Mining. In: BPM, vol. 9850, pp. 158–175. Springer (2016)
21. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. RADAR+ EMISA 1859, 72–80 (2017)
22. Markey, N.: Past is for free: On the complexity of verifying linear temporal properties with past. Electr. Notes Theor. Comput. Sci. 68(2), 87–104 (2002)
23. Pesic, M.: Constraint-based workflow management systems:shifting control to users (2008)
24. Ramakrishna, Y.S., Moser, L.E., Dillon, L.K., Melliar-Smith, P.M., Kutty, G.: An automata-theoretic decision procedure for propositional temporal logic with Since and Until. Fundam. Inform. 17(3), 271–282 (1992)
25. Reynolds, M.: The complexity of temporal logic over the reals. Annals of Pure and Applied Logic 161(8), 1063–1096 (2010)
26. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS. pp. 322–331. IEEE Computer Society (1986)
27. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTLf synthesis. In: IJCAI. pp. 1362–1369 (2017)

Pre-print copy of the manuscript published by Springer (available at `link.springer.com`)
identified by DOI: `10.1007/978-3-319-98648-7_8`

This document is a pre-print copy of the manuscript
(Cecconi et al. 2018)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-319-98648-7_8

References

Cecconi, Alessio, Claudio Di Ciccio, Giuseppe De Giacomo, and Jan Mendling (2018). "Interestingness of traces in declarative process mining: The Janus LTLpf approach". In: *BPM*. Ed. by Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke. Vol. 11080. Lecture Notes in Computer Science. Springer, pp. 121–138. ISBN: 978-3-319-98647-0. DOI: 10.1007/978-3-319-98648-7_8.

BibTeX

```
@InProceedings{   Cecconi.etal/BPM2018:Janus,
  author        = {Cecconi, Alessio and Di Ciccio, Claudio and De Giacomo,
                   Giuseppe and Mendling, Jan},
  title         = {Interestingness of traces in declarative process mining:
                   {T}he {J}anus {LTL}pf approach},
  booktitle     = {BPM},
  year          = {2018},
  pages         = {121--138},
  crossref      = {BPM2018},
  doi           = {10.1007/978-3-319-98648-7_8},
  keywords      = {Process mining; Declarative processes; Temporal logics;
                   Separation theorem; Automata theory}
}
@Proceedings{     BPM2018,
  title         = {Business Process Management - 16th International
                   Conference, {BPM} 2018, Sydney, NSW, Australia, September
                   9-14, 2018, Proceedings},
  year          = {2018},
  editor        = {Mathias Weske and Marco Montali and Ingo Weber and Jan vom
                   Brocke},
  volume        = {11080},
  series        = {Lecture Notes in Computer Science},
  publisher     = {Springer},
  isbn          = {978-3-319-98647-0}
}
```