CITYSPIN

Cyber-Physical Social Systems for City-wide Infrastructures

Deliverable 5.3: Monitoring infrastructure for linked-data event streams

Authors	:	Claudio Di Ciccio, Alessio Cecconi
Dissemination Level	:	Public
Due date of deliverable	:	31.05.2019 (v1)
Actual submission date	:	July 22, 2019
Work Package	:	5. Process Mining and Monitoring on Linked Data
Type	:	Report
Version	:	1.0

Abstract

This document provides the definition of a monitoring technique for linked-data event streams with respect to declarative constraints. The theoretical base of D5.1 is extended to deal with a streaming scenario and the result of D5.2 used to interpret the input stream. The technique is grounded in temporal logic and automata theory. The basic step is the reshaping of the constraints as reactive elements, in order to respond a soon as possible to the incoming events. A fine grain analysis in conducted on the constraints with measures at the granularity of the singles events. The validation of the constraints is independently achieved for each constraint activation. Especially, leveraging automata theory, this validation can be performed efficiently online. A detailed algorithm to implement the technique is delivered.

FFG

Project Funded by FFG – IKT der Zukunft Programme

Project Number: 861213 **Start date: 01.10.2017** **Duration: 30 months**

History

Version	Date	Reason	Revisited by
1.0	31.05.2019	Submission	CDC, AC

Author List

Project Partner	Name(Initial)	Contact Information
WU Wien	Alessio Cecconi (AC)	cecconi@ai.wu.ac.at
WU Wien	Claudio Di Ciccio (CDC)	claudio.di.ciccio@ai.wu.ac.at

Executive Summary

This deliverable expands the analytical tool delivered by WP5 defining a monitoring technique for linked-data event streams. The previous deliverable D5.2 provided the interface to the incoming linked-data from WP4 and a discovery technique for declarative processes. Leveraging this interface, instead of mining rules out of the event log, the monitor checks the compliance of reactive declarative constraints with respect to the event stream is proposed.

The technique extends the theoretical base of D5.1 to deal with a streaming scenario and uses the result of D5.2 used to interpret the input stream. Grounded in temporal logic and automata theory, the basic step is the reshaping of the constraints as reactive elements, in order to respond a soon as possible to the incoming events. A fine grain analysis in conducted on the constraints with measures at the granularity of the singles events. The validation of the constraints is independently achieved for each constraint activation. Especially, leveraging automata theory, this validation can be performed efficiently online. A detailed algorithm to implement the technique is delivered.

This deliverable is largely based on the paper “Interestingness of Traces in Declarative Process Mining: The Janus LTLp$_f$ Approach”[1], published into the *16th International Conference on Business Process Management* (BPM18).

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project goal

The goal of the CitySPIN project is to create a platform for cyber-physical social systems in order to facilitate innovative Smart City infrastructure services. The analysis and monitoring of real data stemming from different sources of smart city infrastructures and services is a pillar of the overarching objective of the project.

1.2 Work package goal

The high-level goal of this WP is to enable process mining and process understanding on Linked Data sources that aggregate, enrich, and filter existing raw data. The operations to that extent are meant to be both off-line, in order to carry out process analytics on historical information and provide insights on the existing processes, and on-line, so as to enable adaptive monitoring on facts as they happen within running process instances.

1.3 Relation to other work packages

WP5 cooperates with the other work packages as follows. First and foremost, process mining and adaptive monitoring will be conducted on data stemming from the Scalable Linked Data Platform, designed, realised and implemented in WP4. The tools generated will be integrated and evaluated in the context of WP7. The process analytics and monitoring of real world information will be subject to secure secure/privacy-aware data access (WP6). The showcase of the project outcome including process analytics and monitoring will be based upon use cases and scenarios elicited and illustrated in the context of WP3.

1.4 Deliverable goal

The goal of Deliverable D5.3 is to define a monitoring infrastructure and architecture for linked data event stream. This continuous monitoring of the stream should be sensitive to the information brought by the event data. More specifically, the monitoring consists in a continuous verification of the adherence of the incoming events sequences to user desired declarative rules. A fine granular measure is defined for this purpose. The approach extend the theory of D5.1 and is empowered by the features of D5.2.

1.5 Deliverable structure

Section 2 illustrates the new concept of event log in a streaming scenario and present the logic system and formal verification instruments upon which our technique is based. Section 3 explains the approach in details, guiding from the specification of constraints as reactive elements, showing how to measure their interaction the the event stream, till how to evaluate them and especially how to do it online. While Sec.3 proves the theoretical robustness of the technique, Section 4 shows step by step the algorithm to implement it. Finally, Section 5 remarks the results and the open questions.

2 Preliminaries

In this section, we illustrate the basic concepts upon which our conformance checking approach is based.

2.1 Streaming Event Logs

In Deliverable D5.1 we defined an event log as the collection of ordered events records. A single event represents the execution of a task, a sequence of events is called trace, and a set of traces compose an event log. Nevertheless, the perspective considered was of a static containers.

A stream is an expanding sequence of events, where new events are recorded continuously. It is like observing the event log while it written, during the execution of the process, rather then when it is completed and stored. Moreover without knowing when it is going to end.

Definition 1 (Streaming event log)**.** *An* streaming event log *is a tuple* $L(E, \Sigma, C, \ell, \iota, \preccurlyeq, \widehat{\Delta}, \widehat{\alpha})$*, where*

$E \triangleq \{e_1, \ldots, e_\infty\}$ *is a semi-infinite set of constant symbols named* events*;*

$\Sigma \triangleq \{a_1, \ldots, a_{|\Sigma|}\}$ *is a finite non-empty set of constant symbols named* activities*;*

$C \triangleq \{\rho_1, \ldots, \rho_\infty\}$ *is a semi-infinite set of constant symbols named* cases*;*

$\ell : E \to \Sigma$ *is the* labelling function *connecting events to activities;*

$\iota : E \to C$ *is the surjective* instance function *associating events to cases;*

$\preccurlyeq \subseteq E \times E$ *is the reflexive* total order *relation over events; we shall denote as* $\prec$ *the corresponding strict total order, excluding from* $\preccurlyeq$ *the* (e, e) *pairs;*

$\widehat{\Delta} \triangleq \{\Delta_1, \ldots, \Delta_{n+m}\} \triangleq \widehat{\Delta_\leqslant} \cup \widehat{\Delta_=}$ *is the* universe of interpretation*, consisting of* $\widehat{\Delta_\leqslant} \triangleq \{\Delta_{\leqslant_1}, \ldots, \Delta_{\leqslant_n}\}$*, the set of* ordered (interpretation) domains*, and* $\widehat{\Delta_=} \triangleq \{\Delta_{=_1}, \ldots, \Delta_{=_m}\}$*, the set of* unordered (interpretation) domains*, with* $m, n \in \mathbb{N}$ *and* $\Delta_\leqslant \cap \Delta_= = \varnothing$*. For* $\widehat{\Delta_\leqslant}$ *a total order relation* $\leqslant$ *is defined. For* $\widehat{\Delta_=}$ *only an equivalence relation* $=$ *is defined instead.*

$\widehat{\alpha} \triangleq \widehat{\alpha_\leqslant} \cup \widehat{\alpha_=} \triangleq \{\alpha_1, \ldots, \alpha_{n+m}\}$ *is the* universe of attribute functions*, consisting of* $\widehat{\alpha_\leqslant} \triangleq \{\alpha_{\leqslant_1}, \ldots, \alpha_{\leqslant_n}\}$ *where* $\alpha_{\leqslant_k} : E \to \Delta_{\leqslant_k} \cup \{\perp\}$ *for* $1 \leqslant k \leqslant n$ *and* $\widehat{\alpha_=} \triangleq \{\alpha_{=_1}, \ldots, \alpha_{=_m}\}$ *where* $\alpha_{=_l} : E \to \Delta_{=_l} \cup \{\perp\}$ *for* $1 \leqslant l \leqslant m$*, associating events to* attribute values *that are of ordered and unordered domains, respectively; the special symbol* $\perp$ *denotes absence of knowledge about the attribute value.*

The given definition matches the one defined in D5.1, but for the semi-infiniteness of the events and cases set. The event sequence must have a start event because either the process or its monitoring must start at a precise point in time. If the initial event is not known, e.g., because the monitoring of the process started after its beginning, the starting of the monitoring defines the start of the event sequence. Anther crucial point is that, despite the infiniteness of events and cases, the single cases are composed of a finite number of events. This is because by definition a process must have a beginning and an ending. Thus it may be execute an infinite amount of time (infinite number of cases), but each execution is composed by a finite number of events.

Thanks to the process ontologies devised in Deliverable D5.2, it is possible to read process events out of linked data. Nevertheles, instead of verifying the declarative constraints through *SHACLare*, we are going to propose a new technique.

2.2 LTLp_f

To cater for temporal modalities referring to the past, we resort on an extension of the syntax of Linear Temporal Logic (LTL), namely the one of Linear-time Temporal Logic with Past on Finite Traces (LTLp_f). Even in a streaming settings, the finite semantic assumption holds because cases are finite.

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

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

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* $\mathbf{U}$ φ 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;
- $\blacklozenge\varphi$ as *True* $\mathbf{S}$ φ 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 $\neg\blacklozenge\neg\varphi$ 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 $\mathbf{S}$ [2].

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* ; $\quad t, i \not\models$ *False* ; $t, i \models \mathsf{a}$ iff $t(i)$ is assigned with a;
$t, i \models \neg\varphi$ iff $t, i \not\models \varphi$; $\qquad t, i \models \varphi_1 \wedge \varphi_2$ iff $t, i \models \varphi_1$ and $t, i \models \varphi_2$;
$t, i \models \bigcirc\varphi$ iff $i < n$ and $t, i+1 \models \varphi$; $\qquad t, i \models \ominus\varphi$ 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*, $\ominus, \boxminus, \blacklozenge, \mathbf{S}$ as *past operators*, and the following pairs of operators as *mirror images*: *(i)* $\bigcirc$ and $\ominus$, *(ii)* $\Box$ and $\boxminus$, *(iii)* $\Diamond$ and $\blacklozenge$, *(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 [3], henceforth denoted as φ_{M}.

Definition 2 (Pure temporal formula [4]). *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, $\varphi^{\blacktriangleleft} = \boxminus(\mathsf{a} \ \mathbf{S} \ (\blacklozenge\mathsf{b}))$, $\varphi^{\blacktriangledown} = \mathsf{a} \wedge \mathsf{b} \vee \mathsf{c}$, 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 [4], proves that such a separation can be obtained.

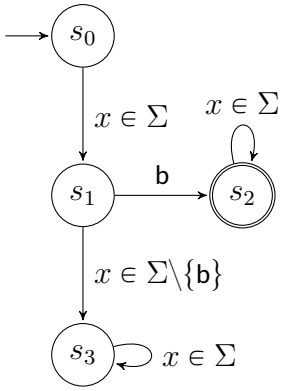


Figure 1: An automaton verifying ○b

Theorem 1 (Separation theorem (adapted from [4])). *Any propositional temporal formula written with* $\mathbf{U}$, $\mathbf{S}$, ○, *and* ⊖ *operators can be rewritten as a boolean combination of pure temporal formulae.*

The constructive proof of Theorem 1 in [4] provides a syntactic procedure to pull out ⊖, $\mathbf{S}$ from the scope of ○, $\mathbf{U}$ 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 3 (Temporal separation, separated formula). *Let* φ *be a* $LTLp_f$ *formula over* Σ. *A* temporal separation *is a function* $\mathscr{S} : LTLp_f \to 2^{LTLp_f \times LTLp_f \times LTLp_f}$. *Indicating* $\varphi^{\blacktriangleleft}$, $\varphi^{\blacktriangledown}$, *and* $\varphi^{\blacktriangleright}$ *respectively as pure past, present, and future formulae, defined over* Σ *as per Def. 2,* $\mathscr{S}(\varphi) = \{(\varphi^{\blacktriangleleft}, \varphi^{\blacktriangledown}, \varphi^{\blacktriangleright})_1, \ldots, (\varphi^{\blacktriangleleft}, \varphi^{\blacktriangledown}, \varphi^{\blacktriangleright})_m\}$ *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, $(\ominus \mathsf{b} \vee \Diamond \mathsf{c}) \equiv ((\ominus \mathsf{b}) \wedge (\mathit{True}) \wedge (\mathit{True})) \bigvee ((\mathit{True}) \wedge (\mathit{True}) \wedge (\Diamond \mathsf{c}))$.

2.3 Automata

A (deterministic finite-state) automaton is a rooted finite-state labelled transition system $A = (\Sigma, S, \delta, s_0, S_\mathsf{F}) \in \mathcal{A}$, 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; $S_\mathsf{F} \subseteq S$ 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 $s_n \in S_\mathsf{F}$. 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 Example 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 LTL [5], Linear-time Temporal Logic with Past (LTLp) [6, 7], or Linear Temporal Logic on Finite Traces (LTL_f) [8] 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 ○b. Notice that automata verification considers whole traces as either satisfying or violating a formula.

Table 1: Some Declare *constraints expressed as Reactive Constraints*

Constraint	LTL$_f$ expression [9]	RCon	Separation degree
Participation(a)	$\Diamond$a	$t_{Start} \boxdot\!\!\rightarrow \Diamond$a	1
Init(a)	a	$t_{Start} \boxdot\!\!\rightarrow$ a	1
End(a)	$\Box\Diamond$a	$t_{End} \boxdot\!\!\rightarrow$ a	1
RespondedExistence(a, b)	$\Diamond$a $\rightarrow \Diamond$b	a $\boxdot\!\!\rightarrow (\Diamond$b $\vee \Diamond^-$b)	2
Response(a, b)	$\Box$(a $\rightarrow \Diamond$b)	a $\boxdot\!\!\rightarrow \Diamond$b	1
AlternateResponse(a, b)	$\Box$(a $\rightarrow \Diamond$b) $\wedge \Box$(a $\rightarrow \bigcirc(\neg$a $\mathbf{W}$ b))	a $\boxdot\!\!\rightarrow \bigcirc(\neg$a $\mathbf{U}$ b)	1
ChainResponse(a, b)	$\Box$(a $\rightarrow \Diamond$b) $\wedge \Box$(a $\rightarrow \bigcirc$b)	a $\boxdot\!\!\rightarrow \bigcirc$b	1
Precedence(a, b)	$\neg$b $\mathbf{W}$ a	b $\boxdot\!\!\rightarrow \Diamond^-$a	1
AlternatePrecedence(a, b)	($\neg$b $\mathbf{W}$ a) $\wedge \Box$(b $\rightarrow \bigcirc(\neg$b $\mathbf{W}$ a))	b $\boxdot\!\!\rightarrow \ominus(\neg$b $\mathbf{S}$ a)	1
ChainPrecedence(a, b)	($\neg$b $\mathbf{W}$ a) $\wedge \Box(\bigcirc$b $\rightarrow$ a)	b $\boxdot\!\!\rightarrow \ominus$a	1

3 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.

3.1 Reactive Constraints

In Deliverable D5.1 we already suggested that a constraint is composed of an activation component and a reaction one. We are now giving a formal definition for it.

Definition 4 (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 a $\boxdot\!\!\rightarrow \Diamond^-$d. Activations in constraints are identified according to the classification of [10]. RCons can include non-standard Declare constraints such as a $\boxdot\!\!\rightarrow (\ominus$b $\vee \Diamond$c), which imposes that if an event a occurs in a trace, either b 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 5 (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 Example 1 and Ψ = a $\boxdot\!\!\rightarrow \Diamond^-$d, i.e., Precedence(d, a) in Declare. Ψ is activated in trace $t_4 = \langle$d, b, a, e$\rangle$ by $t_4(3)$; in trace $t_5 = \langle$a, d, a, c, 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 6 (Interesting fulfilment). *Given a trace $t \in \Sigma^*$ of length n, an instant i s.t. $1 \leqslant i \leqslant n$, and an RCon $\Psi = \alpha \,\square\!\!\rightarrow \varphi$, 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 the following example Example 1

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

Given a formula $\Psi = \mathsf{a} \,\square\!\!\rightarrow \Diamond \mathsf{d}$. 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 $t, i \models \varphi \text{ but } t, i \not\models \alpha$.

As a result each activation of the constraint is independent form the others and can be verified while new events flow into the stream. Either the violation or satisfaction of the constraint is triggered as soon as the, respectively, wring or right events is recorded into the stream. This include also the end of the trace, if the constraint checks some property that should hold within the entire trace.

3.2 Measure

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 7 (Interestingness degree). *Given a trace $t \in \Sigma^*$ and an RCon $\Psi = \alpha \,\square\!\!\rightarrow \varphi$, we define the* interestingness degree *function $\zeta : \mathscr{R} \times \Sigma^* \to [0, 1] \subseteq \mathbb{R}$ 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 $\Psi = \mathsf{a} \,\square\!\!\rightarrow \Diamond \mathsf{d}$ 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 $t_3 = \langle \mathsf{a}, \mathsf{d}, \mathsf{a}, \dots, \mathsf{a}, \mathsf{c} \rangle$ lead to interesting fulfilment, except one (the first a event). Therefore $\zeta(\Psi, t_3) = 0.950$. Both results give a more detailed information of the RCon behaviour in the trace compared to a strict conformance checking, which in these examples would have return 0 because of the presence of violations.

Also this result, aggregated at the level of the single trace, can be computed online and updated as soon as new events pertaining that trace are recorded into the stream. Next, we introduce the computation of two measures for the whole streaming event log, adapting the classical definition of [11]: *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 8 (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* $\sigma : \mathscr{R} \times \mathbb{M}(\Sigma^*) \to [0,1] \subseteq \mathbb{R}$ *calculated as the average of interestingness degree values of* Ψ *over all traces* $t_j \in L$, *with* $1 \leqslant j \leqslant m$:

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

Definition 9 (Confidence). *Given an event log* $L = \{|t_1, t_2, \ldots, t_m|\} \in \mathbb{M}(\Sigma^*)$ *and an RCon* $\Psi \in \mathscr{R}$, *let* $\widetilde{L} = \{|\widetilde{t_1}, \widetilde{t_2}, \ldots, \widetilde{t_p}|\}$ *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), $\Psi = \mathsf{a} \boxdot\!\!\rightarrow \Diamond\mathsf{d}$, 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. **??** shows in detail the interestingness degree, support, and confidence of constraints Precedence(d, a) and $\mathsf{a} \boxdot\!\!\rightarrow (\ominus\mathsf{b} \vee \Diamond\mathsf{c})$ on the event log seen in Example 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 [12, 13], 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.

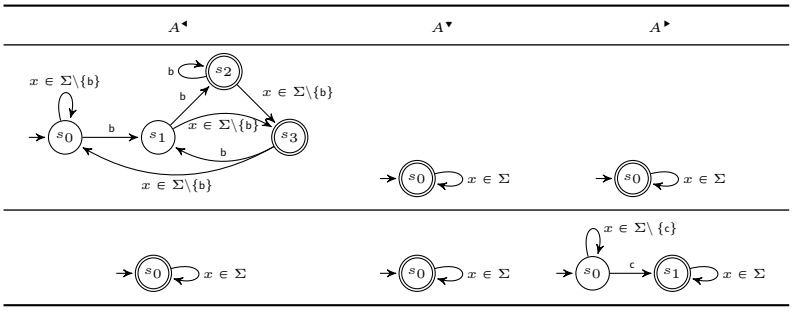
3.3 Reactive Constraint verification

Once an RCon $\Psi = \alpha \boxdot\!\!\rightarrow \varphi$ 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 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:*
$t, i \models \varphi^{\blacktriangleleft} \equiv t_{[1,i]}, i \models \varphi^{\blacktriangleleft}$; $\quad t, i \models \varphi^{\blacktriangledown} \equiv t_{[i,i]}, i \models \varphi^{\blacktriangledown}$; $\quad t, i \models \varphi^{\blacktriangleright} \equiv t_{[i,n]}, i \models \varphi^{\blacktriangleright}$.

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

Table 2: Graphical representation of the separated automata set of $\mathsf{a} \boxdot\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$



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 $t_2 = \langle \mathsf{f}, \mathsf{e}, \mathsf{d}, \mathsf{c}, \mathsf{b}, \mathsf{a}, \mathsf{g}, \mathsf{h}, \mathsf{i} \rangle$ from Example 1, we have that $t_2, 6 \models \varphi$ if *(i)* $\langle \mathsf{f}, \mathsf{e}, \mathsf{d}, \mathsf{c}, \mathsf{b}, \mathsf{a} \rangle, 6 \models \ominus \mathsf{b}$ or *(ii)* $\langle \mathsf{a}, \mathsf{g}, \mathsf{h}, \mathsf{i} \rangle, 6 \models \Diamond \mathsf{c}$, 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 10 (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 $A^{\blacktriangleleft\blacktriangledown\blacktriangleright} = \left\{\left(A_{\ominus \mathsf{b}}, A_{\mathit{True}}, A_{\mathit{True}}\right), \left(A_{\mathit{True}}, A_{\mathit{True}}, A_{\Diamond \mathsf{c}}\right)\right\}$, 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 $t_{2[1:6]} = \langle \mathsf{f}, \mathsf{e}, \mathsf{d}, \mathsf{c}, \mathsf{b}, \mathsf{a} \rangle \in \mathscr{L}\left(A_{\ominus \mathsf{b}}\right)$ or $t_{2[6:9]} = \langle \mathsf{a}, \mathsf{g}, \mathsf{h}, \mathsf{i} \rangle \in \mathscr{L}(A_{\Diamond \mathsf{c}})$.

3.4 Online verification through 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 [8]. To this extent, we rely on mirror images and reversed automata.

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 $t_2 = \langle \mathsf{f}, \mathsf{e}, \mathsf{d}, \mathsf{c}, \mathsf{b}, \mathsf{a}, \mathsf{g}, \mathsf{h}, \mathsf{i} \rangle$ from Example 1 at instant

$i = 9$ is equivalent to verifying $\varphi^{\blacktriangleleft}_{\mathsf{M}} = \Diamond \mathsf{d}$ on $t_{2R} = \langle \mathsf{i}, \mathsf{h}, \mathsf{g}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f} \rangle$ 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 $\varphi^{\blacktriangleleft}_{\mathsf{M}}$ over $t_{2[1:6]R} = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f} \rangle$ 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 3.1. *Let $A_{\varphi^{\blacktriangleleft}_{\mathsf{M}}}$ be the automaton verifying $\varphi^{\blacktriangleleft}_{\mathsf{M}}$. Then $t, n \models \varphi^{\blacktriangleleft}$ iff $t_R \in \mathscr{L}\left(A_{\varphi^{\blacktriangleleft}_{\mathsf{M}}}\right)$.*

Notice that $\varphi^{\blacktriangleleft}_{\mathsf{M}}$ is a pure future formula, therefore $A_{\varphi^{\blacktriangleleft}_{\mathsf{M}}}$ can be built by applying the technique of [8] 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 [14].

Definition 11 (Reversed automaton [14]). *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 3.1 we derive the following.

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

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 $\varphi^{\blacktriangleleft}_{\mathsf{M}} = \bigcirc \mathsf{b}$. With automaton $A_{\varphi^{\blacktriangleleft}_{\mathsf{M}}}$, depicted in Fig. 1, we can verify $\varphi^{\blacktriangleleft}_{\mathsf{M}}$ over trace $t_{2[1:6]R} = \langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f} \rangle$ at $i = 1$, thereby verifying $\varphi^{\blacktriangleleft}$ over $t_{2[1:6]}$ as per Lemma 2. Thanks to Theorem 4, $\varphi^{\blacktriangleleft}_{\mathsf{M}}$ can be verified on $t_{2[1:6]}$ with the reversed automaton $\overleftarrow{A}_{\varphi^{\blacktriangleleft}_{\mathsf{M}}}$, depicted in the top-left corner of Table 2.
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.

4 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:

- $t = t_1 = \langle \mathsf{d}, \mathsf{f}, \mathsf{a}, \mathsf{f}, \mathsf{c}, \mathsf{a}, \mathsf{f}, \mathsf{b}, \mathsf{a}, \mathsf{f} \rangle$ from Example 1,

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 $\mathcal{O} \leftarrow$ empty bag ;
2 **while** event $t(i) \in t$ **do**
3 **foreach** $A^{\blacktriangleleft} \in \mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$ **do** perform transition $t(i)$ on $A^{\blacktriangleleft}$;
4 **if** $t(i) = \alpha$ **then** `// Activation triggered`
5 $\mathcal{J} \leftarrow$ empty set of pairs ; `// J stores the replay-state of future automata for one activation`
6 **foreach** $(A^{\blacktriangleleft}, A^{\blacktriangledown}, A^{\blacktriangleright}) \in \mathcal{A}^{\blacktriangleleft\blacktriangledown\blacktriangleright}$ **do**
7 **if** $A^{\blacktriangleleft}$ *is in an accepting state and* $A^{\blacktriangledown}$ *accepts* $\langle t(i) \rangle$ **then**
8 Take $A^{\blacktriangleright} = (\Sigma, S^{\blacktriangleright}, \delta^{\blacktriangleright}, s_0^{\blacktriangleright}, S_F^{\blacktriangleright})$ and add $(s_0^{\blacktriangleright}, A^{\blacktriangleright})$ to $\mathcal{J}$
9 add $\mathcal{J}$ to $\mathcal{O}$; `// O collects replay-states for all activations`
10 **foreach** $\mathcal{J} \in \mathcal{O}$ **do**
11 **foreach** $(s^{\blacktriangleright}, A^{\blacktriangleright}) \in \mathcal{J}$ **do** $s^{\blacktriangleright} \leftarrow \delta^{\blacktriangleright}(s^{\blacktriangleright}, t(i))$ `// Perform` $t(i)$ `on` $A^{\blacktriangleright}$ `and save state ;`
12 **if** $|\mathcal{O}| > 0$ **then return** $\frac{\left|\left\{\mathcal{J} \in \mathcal{O} : \text{ at least a } (s^{\blacktriangleright}, A^{\blacktriangleright}) \in \mathcal{J} \text{ is s.t. } s^{\blacktriangleright} \in S_F^{\blacktriangleright}\right\}\right|}{|\mathcal{O}|}$ **else return** 0 ;

- $\Psi = \mathsf{a} \boxdot\!\!\rightarrow (\ominus \mathsf{b} \vee \Diamond \mathsf{c})$, and
- the sep.aut.set $\left\{\left(A^{\blacktriangleleft}_{\ominus \mathsf{b}}, A^{\blacktriangledown}_{\mathit{True}}, A^{\blacktriangleright}_{\mathit{True}}\right), \left(A^{\blacktriangleleft}_{\mathit{True}}, A^{\blacktriangledown}_{\mathit{True}}, A^{\blacktriangleright}_{\Diamond \mathsf{c}}\right)\right\}$ of φ, with past automaton $A^{\blacktriangleleft}_{\ominus \mathsf{b}}$ already reversed as depicted in Table 2.

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 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 $t(5) = \mathsf{c}$, $\{(s_0, A^{\blacktriangleright}_{\Diamond\mathsf{c}})\}$ is updated to $\{(s_1, A^{\blacktriangleright}_{\Diamond\mathsf{c}})\}$.

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 $i = 10$, $\mathcal{O} = \{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3\} = \{\{(s_1, A^{\blacktriangleright}_{\Diamond\mathsf{c}})\}, \{(s_0, A^{\blacktriangleright}_{\Diamond\mathsf{c}})\}, \{(s_0 A^{\blacktriangleright}_{\Diamond\mathsf{c}}), (s_0, A^{\blacktriangleright}_{\mathit{True}})\}\}$ and $|\mathcal{O}| = 3$. Only $\mathcal{J}_1$ and $\mathcal{J}_3$ contain automata in accepting states, therefore $\zeta(\Psi, t) = \frac{2}{3} = 0.667$.

4.1 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 $\Psi = \alpha \,\square\!\!\rightarrow\, \varphi$ 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 [15, 13]. 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 Concluding remarks and research outlook

The discussion has been focused on events occurrence to make clearer the technique. However, it is equivalently applicable on the other attributes associated the event itself because the structure of any RCon $\Psi = \alpha \,\square\!\!\rightarrow\, \varphi$. Indeed the formulas are extensible to cover the multiple perspectives as seen in Deliverable D5.1.

In the present work, we leverage the process ontology developed in Deliverable D5.2 , to be able to read a stream of linked event data as a process traces. Then the checking is computed outside of the linked world, to be relinked afterwards only once retrieved the result.The advantage of *SHACLare*, proposed in D5.2, was to be natively employing semantic technologies. It is indeed our next step to integrate this new streaming feature into it to have this advantage back.

6 Summary

In this deliverable, we presented a technique to monitor online linked data event streams with respect to declarative constraints. We extended the logic behind MP-Declare, proposed in D5.1, to deal with the streaming setting, exploiting the reactive nature of constraints. We derived the ability to interpret streams of linked data from D5.2 results.

Each constraint activation is independent from another one, even in within the same process case. This leaded to a fine granular measurement system more resistant to noise, both for the singles traces and the entire observed stream. We exploited automata theory to perform efficiently the verification of reactive constrains online, as soon as new events are recorded into the stream. We detailed the implementation of the technique with an algorithm.

References

[1] A. Cecconi, C. Di Ciccio, G. De Giacomo, and J. Mendling, "Interestingness of Traces in Declarative Process Mining: The Janus LTLpf Approach," in *Business Process Management*. Springer International Publishing, 2018, vol. 11080, pp. 121–138.

[2] I. M. Hodkinson and M. Reynolds, "Separation-Past, Present, and Future." in *We Will Show Them!(2)*, 2005, pp. 117–142.

[3] M. Reynolds, "The complexity of temporal logic over the reals," *Annals of Pure and Applied Logic*, vol. 161, no. 8, pp. 1063–1096, May 2010.

[4] D. Gabbay, "The declarative past and imperative future," in *Temporal logic in specification*. Springer, 1989, pp. 409–448.

[5] M. Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification," in *Proceedings of the First Symposium on Logic in Computer Science*. Cambridge: IEEE Computer Society, 1986, pp. 322–331.

[6] Y. S. Ramakrishna, L. E. Moser, L. K. Dillon, P. M. Melliar-Smith, and G. Kutty, "An automata-theoretic decision procedure for propositional temporal logic with Since and Until," *Fundam. Inform.*, vol. 17, no. 3, pp. 271–282, 1992.

[7] P. Gastin and D. Oddoux, "LTL with past and two-way very-weak alternating automata," in *MFCS*, vol. 3. Springer, 2003, pp. 439–448.

[8] G. De Giacomo, R. De Masellis, and M. Montali, "Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness." in *AAAI*, 2014, pp. 1027–1033.

[9] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. Association for Computing Machinery, 2013, pp. 854–860.

[10] C. Di Ciccio, F. M. Maggi, M. Montali, and J. Mendling, "Resolving inconsistencies and redundancies in declarative process models," *Information Systems*, vol. 64, pp. 425–446, March 2017.

[11] J. Adamo, *Data mining for association rules and sequential patterns - sequential and parallel algorithms*, J. Adamo, Ed. Springer New York, 2001.

[12] F. M. Maggi, R. J. C. Bose, and W. M. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *International Conference on Advanced Information Systems Engineering*. Springer, 2012, pp. 270–285.

[13] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Information Systems*, 2018.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation*, 3rd ed. Boston: Pearson/Addison Wesley, 2007, oCLC: ocm69013079.

[15] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Transactions on Management Information Systems (TMIS)*, vol. 5, no. 4, p. 24, 2015.