

Efficient Discovery of Target-Branched Declare Constraints

Claudio Di Ciccio^{a,*}, Fabrizio Maria Maggi^b, Jan Mendling^a

^aVienna University of Business and Economics, Austria

^bUniversity of Tartu, Estonia

Abstract

Process discovery is the task of generating process models from event logs. Mining processes that operate in an environment of high variability is an ongoing research challenge because various algorithms tend to produce spaghetti-like process models. This is particularly the case when procedural models are generated. A promising direction to tackle this challenge is the usage of declarative process modelling languages like Declare, which summarise complex behaviour in a compact set of behavioural constraints on activities. A Declare constraint is branched when one of its parameters is the disjunction of two or more activities. For example, branched Declare can be used to express rules like “in a bank, a mortgage application is always eventually followed by a notification to the applicant by phone *or* by a notification by e-mail”. However, branched Declare constraints are expensive to be discovered. In addition, it is often the case that hundreds of branched Declare constraints are valid for the same log, thus making, again, the discovery results unreadable. In this paper, we address these problems from a theoretical angle. More specifically, we define the class of Target-Branched Declare constraints and investigate the formal properties it exhibits. Furthermore, we present a technique for the efficient discovery of compact Target-Branched Declare models. We discuss the merits of our work through an evaluation based on a prototypical implementation using both artificial and real-life event logs.

Keywords: Process Mining, Knowledge Discovery, Declarative Process

1. Introduction

2 Process discovery is the important initial step of business process manage-
3 ment that aims at arriving at an as-is model of an investigated process [1]. Due

*Corresponding author.

E-mail address: claudio.di.ciccio@wu.ac.at

Postal address: Vienna University of Economics and Business, Institute for Information Business (Building D2, Entrance C) – Welthandelsplatz 1, A-1020 Vienna, Austria

Phone number: +43 1 31336 5222

Preprint submitted to Elsevier

28th July 2015

4 to this step being difficult and time-consuming, various techniques have been
5 proposed to automatically discover a process model from event logs. These log
6 data are often generated by information systems that support parts or the en-
7 tirety of a process. The result is typically presented as a Petri net or a similar
8 kind of flow chart and the automatic discovery is referred to as process mining.

9 While process mining has proven to be a powerful technique for structured
10 and standardised processes, there is an ongoing debate on how processes with a
11 high degree of variability can be effectively mined. One approach to this problem
12 is to generate a declarative process model, which rather shows the constraints
13 of behaviour instead of the available execution sequences. The resulting models
14 are represented in languages like Declare. In many cases, they provide a way
15 to represent complex, unstructured behaviour in a compact way, which would
16 look overly complex in a spaghetti-like Petri net.

17 Declare is a process modelling language first introduced in [2]. The language
18 defines a set of classes of constraints, the Declare templates, that are considered
19 the most interesting ones for describing business processes. Templates are para-
20 meterised and constraints are instantiations of templates on real activities. For
21 example, the *Response* constraint, stating that “activity *pay* is always eventually
22 followed by activity *send invoice*” is an instantiation of the Declare template
23 *Response* specifying that “an activity *x* is always eventually followed by an activ-
24 ity *y*”. Templates have a graphical representation and formal semantics based
25 on Linear Temporal Logic on Finite Traces (LTL_f). This allows Declare models
26 to be verifiable and executable. Figure 1a shows the graphical representation of
27 the *Response* template. Its LTL_f semantics is $\Box(x \rightarrow \Diamond y)$. Constraints inherit
28 the graphical representation and the LTL_f semantics from the corresponding
29 templates.

30 The current techniques for the discovery of Declare models [3, 4, 5, 6, 7] are
31 limited to the discovery of constraints based on the standard set of Declare tem-
32 plates. This means that the discovered constraints will involve one activity for
33 each parameter specified in the corresponding templates. However, as described
34 in [2], a constraint can define more than one activity for each parameter. For
35 example, a *Response* constraint can be used to express rules like “in a bank,
36 a mortgage application is always eventually followed by a notification to the
37 applicant by phone *or* by a notification by e-mail”. In this rule, the “mortgage
38 application” plays the role of the *activation*. “Notification by phone” and “noti-
39 fication by e-mail” constitute the so-called *targets* of the constraint. In this case,
40 we say that the target parameter branches and, in the graphical representation,
41 this is displayed by multiple arcs connecting the activation to the branched tar-
42 gets. In LTL_f semantics, a branched parameter is replaced by a disjunction
43 of parameters. Figure 1b shows the graphical representation of the *Response*
44 template branching on the target. Its LTL_f semantics is $\Box(x \rightarrow \Diamond(y \vee z))$.

45 Target-Branched Declare (TBDeclare) extends Declare by encompassing
46 constraints that branch on target parameters, thus providing the process mod-
47 ellers with the possibility of defining a much wider set of constraints. In this
48 paper, we address the problem of mining TBDeclare constraints efficiently. At
49 the same time, the technique we propose aims at limiting the sheer amount of

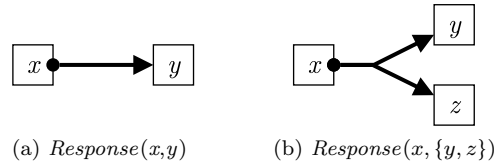


Figure 1: Declare (a) and Target-Branched Declare (b) *Response* templates.

50 returned constraints to the set of the most meaningful ones. To this extent, we
 51 rely on formal properties of TBDeclare, i.e., (i) set-dominance and (ii) subsumption
 52 hierarchy. Set-dominance is based on the observation that, for example,
 53 stating that “a is always eventually followed by b or c” entails that “a is al-
 54 ways eventually followed by b, c or d”, i.e., since the set of targets for the first
 55 constraint is included in the set of targets for the second constraint, the first
 56 constraint is stronger than the second one. In this case, if both constraints hold
 57 in the provided event log, only the stronger one will be discovered. In addition,
 58 Declare constraints are not independent, but form a subsumption hierarchy.
 59 Therefore, a constraint (e.g., a is eventually followed by b or c) is redundant if
 60 a stronger constraint holds (e.g., a is directly followed by b or c). Also in this
 61 case, it is possible to keep the stronger constraint and discard the weaker one
 62 in the discovered model. The key idea of our proposed approach is to exploit
 63 set-dominance and subsumption hierarchy relationships, in combination with
 64 the use of interestingness metrics like constraint support and confidence [5], to
 65 drastically prune the set of discovered constraints. We present formal proofs to
 66 demonstrate the merits of this approach and a prototypical implementation for
 67 emphasising its feasibility and efficiency.

68 In this paper, we extend the work presented in [8] in four directions: (i) the-
 69oretical discussion, (ii) algorithm presentation, (iii) implementation improve-
 70ment, and (iv) evaluation. From a foundational perspective, this paper form-
 71ally elaborates on how the monotonicity of LTL_f temporal operators can be
 72exploited to prove set-dominance for TBDeclare. The algorithm is presented
 73in thorough detail here: it describes all the procedures undertaken to mine
 74the constraints, along with trailing examples. The implementation of the al-
 75gorithm is also improved now, as an entirely new technique for the computation
 76of *AlternateResponse* and *AlternatePrecedence* constraints has been devised. In
 77this way, a major limitation of the process discovery algorithm presented in [8]
 78is resolved. Furthermore, this has enabled us to cover a broader range of exper-
 79iments including the application to an additional benchmark based on the use
 80of the log provided for the BPI challenge 2014 [9].

81 Against this background, this paper is structured as follows. Section 2 in-
 82troduces the essential concepts of LTL_f and Declare as a background of our
 83work. Section 3 provides the formal foundations for mining Target-Branched
 84constraints. Section 4 defines the construction of a knowledge base from which
 85the final constraint set is built. Section 5 describes the performance evaluation.

86 Section 6 investigates our contribution in the light of related work. Section 7
87 concludes the paper with an outlook on future research.

88 2. Background

89 Process mining is the set of techniques that aims at understanding the be-
90 haviour of a process, given as input a set of data reporting the executions of
91 such a process, i.e., an event log L . An event log consists of a collection of
92 traces t_i , with $i \in [1, |L|]$ and $|L|$ being the size of the log, recording informa-
93 tion about process instance executions. A trace is a sequence of events. Events
94 are log entries specifying the execution data referred to an activity of the pro-
95 cess. In the following, we will assume that each event is uniquely corresponding
96 to the execution of a single activity. Therefore, we will interchangeably adopt
97 the terms “event” and “activity” occurring in the log. The set of the activities
98 that may occur in the log is called log alphabet. Hereinafter, the generic log
99 alphabet is denoted as Σ . Elements of Σ will be collectively indicated as a, b, c .
100 Denoting the set of sequences of activities as Σ^* , and indicating a multi-set as
101 $\mathfrak{M}(\cdot)$, we have that $L \in \mathfrak{M}(\Sigma^*)$.

102 One of the challenges in process mining is the compact presentation of the
103 mined behaviour. It has been observed that procedural models such as Petri nets
104 tend to become overly complex for flexible processes that are situated in dynamic
105 environments. Therefore, it has been argued to rather utilise declarative process
106 modelling languages (like Declare) in such contexts, in order to facilitate a better
107 understanding of the mined process models by humans [10, 11]. Declare has its
108 formal foundation in linear temporal logic with finite trace semantics, which
109 we introduce in Section 2.1. Section 2.2, then, describes Declare and how it is
110 grounded in linear temporal logic.

111 2.1. Linear Temporal Logic over Finite Traces

112 Linear Temporal Logic (LTL) [12] is a language meant to express properties
113 that hold true in systems that change their state over time. The behaviour of
114 such systems is expressed in the form of a temporal structure, i.e., a transition
115 system [13]. LTL was originally proposed in computer science as a specification
116 language for concurrent programs. It was thought, in fact, to be adopted for
117 the formal verification of server systems and very large system circuits, which
118 in theory run infinitely. The states of such systems are expressed in terms of
119 propositional formulae. The evolution is defined by transitions between states.

A typical LTL formula expressing a *fairness* condition is $\Box \diamond \Phi$, where Φ is
a propositional formula, indicating the condition to always (\Box) eventually (\diamond)
hold true. LTL_f [14, 15] is the variant of LTL interpreted over finite system
executions. It adopts the syntax of LTL. Formulae of LTL_f are built from
a set \mathcal{A} of propositional symbols (*atomic propositions*) and are closed under
the boolean connectives (\neg , unary, and \vee , \wedge , \rightarrow , binary) and the temporal
operators \bigcirc (*next*), \diamond (*eventually*), \Box (*always*), unary, and \mathcal{U} (*until*) and \mathcal{W}

(*weak until*), binary. The syntax is defined as follows.

$$\begin{array}{lcl}
\varphi, \psi = & \alpha & | \rho & | \lambda & & \text{(with } \alpha \in \mathcal{A}\text{)} \\
\rho = & \neg\varphi & | \varphi \vee \psi & | \varphi \wedge \psi & | \varphi \rightarrow \psi \\
\lambda = & \bigcirc\varphi & | \diamond\varphi & | \square\varphi & | \varphi \mathcal{U} \psi & | \varphi \mathcal{W} \psi
\end{array}$$

Intuitively, $\bigcirc\varphi$ means that φ holds true in the next instant in time, $\diamond\varphi$ signifies that φ holds eventually before the last instant in time (included), $\square\varphi$ expresses the fact that from the current state until the last instant in time φ holds, $\varphi \mathcal{U} \psi$ says that ψ holds eventually in the future and φ holds until that point, and $\varphi \mathcal{W} \psi$ relaxes $\varphi \mathcal{U} \psi$ in the sense that either from the current state until the last instant in time φ holds, or $\varphi \mathcal{U} \psi$ holds.

The semantics of LTL_f is provided in terms of finite runs, i.e., finite sequences of consecutive instants in time, represented by finite words π over $2^{\mathcal{A}}$. The instant i in run π is denoted as $\pi(i)$, with $i \in [1, |\pi|]$, with $|\pi|$ being the length of the run. In the following, we indicate that, e.g., atomic proposition α is interpreted as true (\top) at instant i in π with $\alpha \in \pi(i)$. Conversely, if $\alpha \notin \pi(i)$, α is interpreted as false (\perp). Given a finite run π , we inductively define when an LTL_f formula φ (respectively ψ) is true at an instant i , denoted as $\pi, i \models \varphi$ (respectively $\pi, i \models \psi$), as:

- 134 $\pi, i \models \alpha$ for $\alpha \in \mathcal{A}$, iff $\alpha \in \pi(i)$ (α is interpreted as true in $\pi(i)$);
- 135 $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- 136 $\pi, i \models \varphi \wedge \psi$ iff $\pi, i \models \varphi$ and $\pi, i \models \psi$;
- 137 $\pi, i \models \varphi \vee \psi$ iff $\pi, i \models \varphi$ or $\pi, i \models \psi$;
- 138 $\pi, i \models \bigcirc\varphi$ iff $\pi, i+1 \models \varphi$, having $i < |\pi|$;
- 139 $\pi, i \models \varphi \mathcal{U} \psi$ iff for some $j \in [i, |\pi|]$, we have that $\pi, j \models \psi$, and for all
- 140 $k \in [i, j-1]$, we have that $\pi, k \models \varphi$.

The semantics of the remaining operators can be derived by recalling that:

$$\begin{array}{ll}
142 & \varphi \rightarrow \psi \equiv \neg\varphi \vee \psi & 144 & \square\varphi \equiv \neg\diamond\neg\varphi; \\
143 & \diamond\varphi \equiv \top \mathcal{U} \varphi; & 145 & \varphi \mathcal{W} \psi \equiv \square\varphi \vee (\varphi \mathcal{U} \psi).
\end{array}$$

We recall here that, given two LTL_f formulas φ, ψ , $\varphi \models \psi$ (φ models ψ) iff $\forall i \in [1, \pi], \pi, i \models \varphi$ entails $\pi, i \models \psi$. As clarified in [16], temporal operators enjoy the property of *monotonicity* [17]. A function $f : X \rightarrow Y$, where X and Y are partially ordered sets under the binary relation \leq , is *monotonic* iff, given $x, x' \in X$ such that $x \leq x'$, then $f(x) \leq f(x')$. f is said to be *antimonotonic* iff, given $x, x' \in X$ such that $x \leq x'$, then $f(x) \geq f(x')$, where \geq is the inverse of \leq .

With a slight abuse of notation, given formulae φ and ψ , such that $\varphi \models \psi$, then:

- 155 1. a unary operator \bullet is monotonic iff $\bullet\varphi \models \bullet\psi$;
- 156 2. a unary operator \bullet is antimonotonic iff $\bullet\varphi \models \bullet\psi$;
- 157 3. a binary operator \otimes is monotonic iff $\varphi \models \varphi \otimes \psi$;

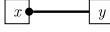
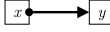





Template	Formalisation	Notation	Activ.	Target
<i>RespondedExistence</i> (x, y)	$\diamond x \rightarrow \diamond y$		x	y
<i>Response</i> (x, y)	$\square(x \rightarrow \diamond y)$		x	y
<i>Precedence</i> (x, y)	$\neg y \mathcal{W} x$		y	x
<i>AlternateResponse</i> (x, y)	$\square(x \rightarrow \bigcirc(\neg x \mathcal{U} y))$		x	y
<i>AlternatePrecedence</i> (x, y)	$(\neg y \mathcal{W} x) \wedge \square(y \rightarrow \bigcirc(\neg y \mathcal{W} x))$		y	x
<i>ChainResponse</i> (x, y)	$\square(x \rightarrow \bigcirc y)$		x	y
<i>ChainPrecedence</i> (x, y)	$\square(\bigcirc y \rightarrow x)$		y	x

Table 1: Graphical notation and LTL_f formalisation of some Declare templates.

158 4. a binary operator \otimes is antimonic iff $\varphi \equiv \varphi \otimes \psi$.

159 Monotonicity holds for propositional logic operators \vee , \wedge and \rightarrow , whereas \neg is
160 antimonic [17]. Temporal operators \bigcirc , \diamond , \square , \mathcal{U} and \mathcal{W} are monotonic
161 as well [13]. LTL_f syntax and semantics will be used in the remainder of the
162 paper.

163 2.2. Declare

164 One of the most frequently used declarative languages is Declare, first in-
165 troduced in [2]. Instead of explicitly specifying the allowed sequences of events,
166 Declare consists of a set of constraints that are applied to activities and must
167 be valid during the process execution. Constraints are based on templates that
168 define parametrised classes of temporal properties. Templates have a graphi-
169 cal representation and their semantics can be formalised using LTL_f . In this
170 way, analysts work with the graphical representation of templates, while the
171 underlying formulae remain hidden. Table 1 summarises the most commonly
172 used Declare templates. For a complete specification see [2]. Here, we indicate
173 template parameters with x or y symbols. Generic symbols of real activities in
174 their instantiations (generic elements of a generic log alphabet) are indicated as
175 $\Sigma = \{a, b, c, d, \dots\}$. Assigned activity identifiers are denoted as sans-serif letters
176 $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \Sigma$, where Σ is a set of activities that is assigned to a generic log alphabet
177 Σ . Hence, \mathbf{a} is a possible assignment of a . Following the same rationale, L is the
178 symbol for the formal parameter denoting a generic log, whereas \mathbf{L} is a concrete
179 log. We remark here that the interpretation of Declare constraints restricts the
180 common interpretation of LTL_f in that two literals cannot be true at the same
181 time. Furthermore, the run π on which an LTL_f formula is evaluated is, in this
182 context, a finite trace \vec{t} of a log. We will univocally map atomic propositions of
183 LTL_f to the occurrence of an activity in the log alphabet ($\mathcal{A} \equiv \Sigma$).

184 The formulae shown in Table 1 can be readily formulated using natural
185 language. The *RespondedExistence* template specifies that if x occurs, then y
186 should also occur (either before or after x). The *Response* template specifies

187 that when x occurs, then y should eventually occur after x . The *Precedence*
 188 template indicates that y should occur only if x has occurred before. Tem-
 189 plates *AlternateResponse* and *AlternatePrecedence* strengthen the *Response* and
 190 *Precedence* templates respectively by specifying that activities must alternate
 191 without repetitions in between. Even stronger ordering relations are specified
 192 by templates *ChainResponse* and *ChainPrecedence*. These templates require
 193 that the occurrences of the two activities (x and y) are next to each other.

194 In order to illustrate these semantics, consider the *Response* constraint
 195 $\Box(a \rightarrow \Diamond b)$. This constraint indicates that if a occurs, b must eventually
 196 follow. Given a log $L = \{\vec{t}_1, \vec{t}_2, \vec{t}_3, \vec{t}_4\}$, where $\vec{t}_1 = \langle a, a, b, c \rangle$, $\vec{t}_2 = \langle b, b, c, d \rangle$,
 197 $\vec{t}_3 = \langle a, b, c, b \rangle$, and $\vec{t}_4 = \langle a, b, a, c \rangle$, this constraint is satisfied in \vec{t}_1 , \vec{t}_2 , and \vec{t}_3 ,
 198 but not in \vec{t}_4 , because the second instance of a is not followed by a b in such
 199 trace.

200 An *activation* of a constraint in a trace is an event whose occurrence imposes
 201 some obligations on the occurrence of another event (the *target*) in the same
 202 trace. E.g., a is the activation and b is the target for the *Response* constraint
 203 $\Box(a \rightarrow \Diamond b)$, because the execution of a forces b to be executed eventually. When
 204 a trace is compliant w.r.t. a constraint, every activation leads to a *fulfilment*.
 205 Consider, again, the *Response* constraint $\Box(a \rightarrow \Diamond b)$. In trace \vec{t}_1 , the constraint
 206 is activated and fulfilled twice, whereas, in \vec{t}_3 , the same constraint is activated
 207 and fulfilled only once. On the other hand, when a trace is non-compliant, an
 208 activation can lead to a fulfilment but at least one activation in the trace leads
 209 to a *violation*. For example, in trace \vec{t}_4 , the *Response* constraint $\Box(a \rightarrow \Diamond b)$
 210 is activated twice: the first activation leads to a fulfilment (eventually b oc-
 211 curs) and the second activation to a violation (b does not occur subsequently).
 212 An algorithm to identify fulfilments and violations in a trace is presented in [18].
 213

214 In the following, we will use \mathcal{C} to denote the set of Declare templates. Formally,
 215 a Declare template $\mathcal{C}/n \in \mathcal{C}$ is a predicate of *arity* $n \geq 1$, with \mathcal{C} represent-
 216 ing the *name* of the template and *arity* n specifying the number of *parameters*.
 217 For instance, the aforementioned *Response*₂ constraint has arity $n = 2$ and
 218 *Response* as name. $\mathcal{C}(x, y)$ is an example of the notation we adopt to explic-
 219 itly identify the two parameters (x, y) of template \mathcal{C} of arity 2. In standard
 220 Declare, constraints are templates whose parameters are assigned single distinct
 221 activities. We will denote as \mathcal{C}^Σ the set of constraints that are obtained
 222 by assigning parameters of every $\mathcal{C}/n \in \mathcal{C}$ to n -permutations of distinct activ-
 223 ities in the log alphabet Σ . Having, e.g., $\Sigma = \{a, b, c\}$, \mathcal{C}^Σ would comprise
 224 *Response*(a, b), *Response*(b, a), *Response*(b, c), *Response*(c, b), *Response*(a, c),
 225 *Response*(c, a), *RespondedExistence*(a, b), *RespondedExistence*(b, a), etc. We
 226 will use $\mathcal{C}(a, b) \in \mathcal{C}^\Sigma$ for indicating a generic constraint that assigns activit-
 227 ies a and b ($a, b \in \Sigma$) to the parameters of the corresponding template \mathcal{C} . \mathcal{C} is
 228 a shorthand notation that denotes a generic constraint.

229 In the remainder of this paper, we will focus on Declare constraints of arity
 230 2 known as *relation constraints*. In particular, we will consider the ones that
 231 are listed in Table 1, hereinafter indicated as *unidirectional positive relation*

232 *constraints*. The activation and target of a relation constraint C are henceforth
 233 denoted as $C|_{\bullet}$ and $C|_{\Rightarrow}$. Thus, $Response(a, b)|_{\bullet}$ is a and $Response(a, b)|_{\Rightarrow}$
 234 is b . Vice-versa, $Precedence(a, b)|_{\bullet}$ is b and $Precedence(a, b)|_{\Rightarrow}$ is a . Table 1
 235 reports activations and targets for all the listed templates. In the semantics of
 236 a unidirectional positive relation constraint $C = \mathcal{C}(x, y)$,

- 237 1. $\{C|_{\bullet}\} \cap \{C|_{\Rightarrow}\} = \emptyset$, and
- 238 2. $C|_{\Rightarrow}$ always falls under an even number of negations \neg .

239 In the literature, relation templates of arity 2 that do not impose rule 1 are
 240 known as “coupling constraints”, whereas those that do not impose rule 2 are
 241 named “negative constraints” [19].

242 Table 1 contains the list of activations and targets for the templates we
 243 consider in this paper. In Section 3, we will explain how the standard Declare
 244 specification is extended towards Target-Branched Declare.

245 2.3. Support and confidence

246 To evaluate the relevance of a Declare constraint, we adopt two metrics
 247 proposed in the association rule mining literature [20]. The first one is meant to
 248 assess the reliability of a constraint w.r.t. a log, i.e., *support*. The second metric
 249 is meant to assess the relevance of a constraint w.r.t. a log, i.e., *confidence*.

250 The *support* of a Declare constraint C in an event log is defined as the
 251 proportion of fulfilments $\checkmark_L(C)$ of C in log L . For relation constraints, we
 252 can rely on the concept of activation. Therefore, we specify the support as
 253 the fraction of occurring activations of C that do not violate the constraint,
 254 w.r.t. the total number of activations in the log. Formally, let $\#_L(a)$ be the
 255 function $\# : \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$, with \mathbb{N} set of positive integers, that counts the
 256 occurrences of activity $a \in \Sigma$ in log $L \in \mathfrak{M}(\Sigma^*)$; let $\checkmark_L(C)$ be the function
 257 $\checkmark : \mathfrak{C}^\Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$ that counts the number of fulfilments of constraint $C \in$
 258 \mathfrak{C}^Σ in log $L \in \mathfrak{M}(\Sigma^*)$. Then, the support $\mathcal{S}_L(C)$ of a constraint C in a log L is
 259 defined as a function $\mathcal{S} : \mathfrak{C}^\Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow [0, 1] \subseteq \mathbb{R}$, with \mathbb{R} set of real numbers,
 260 expressed as follows:

$$\mathcal{S}_L(C) = \frac{\checkmark_L(C)}{\#_L(C|_{\bullet})} \quad (1)$$

261 where $C|_{\bullet}$ is the activation of constraint C .

262 The second metric is meant to assess the relevance of a constraint w.r.t. a
 263 log. It is named *confidence*, and scales the support of a constraint by the number
 264 of traces containing its activation. For the definition of confidence \mathcal{C} of relation
 265 constraints, we rely on the notion of *activity-related log fraction* [21], i.e., the
 266 fraction of traces in which a given activity occurs at least once. Let $\varnothing_L(a)$ be
 267 the function $\varnothing : \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$ that counts the traces of log $L \in \mathfrak{M}(\Sigma^*)$ in
 268 which activity $a \in \Sigma$ does *not* occur. Then, the activity-related log fraction is
 269 expressible as $1 - \frac{\varnothing_L(a)}{|L|}$ where $|L|$ is the number of traces in L . Therefore, given

270 a log $L \in \mathfrak{M}(\Sigma^*)$ and a constraint $C \in \mathfrak{C}^\Sigma$, the confidence of C can be defined
 271 as a function $\mathcal{C} : \mathfrak{C}^\Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow [0, 1] \subseteq \mathbb{R}$, expressed as follows:

$$\mathcal{C}_L(C) = \mathcal{S}_L(C) \times \left(1 - \frac{\varnothing_L(C|\bullet)}{|L|}\right). \quad (2)$$

272 3. Target-Branched Declare

273 In this section, we define Target-Branched Declare (TBDeclare). It extends
 274 Declare such that the target is not a single activity but a set of activities (see
 275 Table 2). This means that $Response(a, \{b, c\})$ is a TBDeclare constraint stat-
 276 ing that “if a occurs, b or c must eventually follow”. $\{b, c\}$ is referred to as a
 277 set-parameter. The cardinality of this set, indicating the number of branches, is
 278 called *branching factor* of the constraint. The class of TBDeclare exhibits some
 279 interesting properties, i.e., subsumption hierarchy and set-dominance. Sub-
 280 sumption hierarchy has already been investigated in [22] for branched Declare.
 281 In the following, we prove that the property of set-dominance holds. Then,
 282 we discuss implications of this property in terms of constraint support. These
 properties will be exploited in the mining algorithm.

TBDeclare template	LTL _f semantics
$RespondedExistence(x, Y)$	$\diamond x \rightarrow \diamond \left(\bigvee_{i=1}^{\beta} y_i\right)$
$Response(x, Y)$	$\square \left(x \rightarrow \diamond \left(\bigvee_{i=1}^{\beta} y_i\right)\right)$
$AlternateResponse(x, Y)$	$\square \left(x \rightarrow \bigcirc \left(\neg x \mathcal{U} \bigvee_{i=1}^{\beta} y_i\right)\right)$
$ChainResponse(x, Y)$	$\square \left(x \rightarrow \bigcirc \left(\bigvee_{i=1}^{\beta} y_i\right)\right)$
$Precedence(Y, x)$	$\neg x \mathcal{W} \left(\bigvee_{i=1}^{\beta} y_i\right)$
$AlternatePrecedence(Y, x)$	$Precedence(Y, x) \wedge \square \left(x \rightarrow \bigcirc Precedence(Y, x)\right)$
$ChainPrecedence(Y, x)$	$\square \left(\bigcirc x \rightarrow \left(\bigvee_{i=1}^{\beta} y_i\right)\right)$

Table 2: LTL_f semantics for TBDeclare templates ($Y = \{y_1, \dots, y_\beta\}$, with β *branching factor* of the constraint).

283 Formally, TBDeclare is a sub-class of a more general class of constraints
 284 extending standard Declare, which we will henceforth refer to as *Multi-valued*
 285 *Declare*. As said in Section 2.2, standard Declare imposes that template param-
 286 eters are interpreted as single activities of the log alphabet Σ . Multi-valued
 287 Declare comprises the same set of templates of standard Declare, yet allowing
 288 the interpretation of parameters as elements of a boolean algebraic structure
 289 $\langle \Sigma, \star \rangle$ (a.k.a. groupoid [23]) consisting of a set of symbols Σ (i.e., the log alpha-
 290 bet) and a binary operator \star under which the structure is closed. Thus, given
 291 $a, b \in \Sigma$, and $\rho = a \star b$, then $\rho \in \langle \Sigma, \star \rangle$. A semigroup $\langle \Sigma, \star \rangle$ is a groupoid s.t.
 292 operation \star is associative.
 293

294 Branched Declare [2] restricts the algebraic structure to a join-semilattice
 295 $\langle \Sigma, \vee \rangle$, i.e., an idempotent commutative semigroup, where \vee is the join-
 296 operation [24]. For any semigroup $\langle \Sigma, * \rangle$ a natural partial-order relation
 297 can be defined [25]. A fortiori, we define it here for join-semilattices as
 298 $\rho \geq \rho'$ iff $\rho \vee \rho' = \rho$, for $\rho, \rho' \in \langle \Sigma, \vee \rangle$. In the domain of boolean algebras, \geq is
 299 defined by the inverse entailment relation \models , and the equality $=$ by the logical
 300 equivalence \equiv . Indeed, e.g., considering $a \vee b$ as ρ and a as ρ' , we clearly have
 301 that $a \vee b \models a$ as $a \vee b \vee a \equiv a \vee b$.

302 TBDeclare belongs to the class of unidirectional positive relation constraints
 303 that further restrict Branched Declare as follows: the interpretation of the target
 304 can be an element of $\langle \Sigma, \vee \rangle$, whereas the activation is interpretable only as a
 305 single activity in Σ . Thus, the example TBDeclare constraint given at the
 306 beginning of this section interprets the activation of *Response* as a and its target
 307 as $b \vee c$, where a , b and c are respectively assigned with \mathbf{a} , \mathbf{b} and \mathbf{c} , meaning
 308 that “if \mathbf{a} occurs, \mathbf{b} or \mathbf{c} must eventually follow.” Notice that the join-operation
 309 of the join-semilattice in boolean algebra is such that semantics of Branched
 310 Declare and TBDeclare can still be expressed within LTL_f .

311 3.1. Set-Dominance

312 In this section, we formally prove that set-dominance holds for TBDeclare,
 313 mainly relying on the property of monotonicity of the LTL_f temporal operators.
 314 To this extent, we first define the class of Monotonic Branched Declare. Then,
 315 we show that two Monotonic Branched Declare constraints C and C' are such
 316 that if the assigned parameters of C are included in the assigned parameters of
 317 C' , then the support of C is lower than or equal to the support of the C' . The
 318 property of monotonic non-decreasing trend of support w.r.t. the containment
 319 of set-parameters, will also be simply referred to as *set-dominance* for short.
 320 Finally, we show that for all Branched Declare constraints, the property of
 321 set-dominance holds true.

322 Preliminarily, we notice that, for join-semilattices, a bijective mapping
 323 μ can be established that connects elements of $\langle \Sigma, \vee \rangle$ to elements of $\langle \Sigma, \cup \rangle$
 324 where \cup is the set-union operation: $\mu : \langle \Sigma, \vee \rangle \rightarrow \langle \Sigma, \cup \rangle$. It can be shown
 325 that μ is an isomorphism preserving the partial-order \geq defined on $\langle \Sigma, \vee \rangle$
 326 by the partial-order given by set-containment \supseteq in $\langle \Sigma, \cup \rangle$. In the following,
 327 we indicate by means of a capital letter, e.g., X or Y , a parameter that is
 328 interpreted as an element of $\langle \Sigma, \vee \rangle$. Therefore, *Response*(x, Y) specifies a
 329 template where the second parameter is interpreted as an element of $\langle \Sigma, \vee \rangle$
 330 (cf. Table 2). Without loss of generality, we identify every element ρ in $\langle \Sigma, \vee \rangle$
 331 by its μ -mapped set $S = \mu(\rho)$ in $\langle \Sigma, \cup \rangle$. Hence, *Response* having a as the
 332 activation and $b \vee c$ as the target will be denoted as *Response*(a, S) where
 333 $S = b \cup c$. S will thus be also referred to as *set-parameter*. As an alternative,
 334 we will also adopt for such constraint the following notation: *Response*($a, \{b, c\}$).
 335

336
 337 Monotonic Branched Declare is the class of Multi-valued Declare templates
 338 for which it holds true that any two constraints $\mathcal{C}(R_1, \dots, R_n)$ and $\mathcal{C}(S_1, \dots, S_n)$,

339 obtained as instantiations of the same template $\mathcal{C}/_n \in \mathfrak{C}$ with set-parameters
 340 R_1, \dots, R_n and S_1, \dots, S_n , with $S_i \supseteq R_i$ for every $i \in [1, n]$, are such that
 341 $\mathcal{C}(R_1, \dots, R_n) \models \mathcal{C}(S_1, \dots, S_n)$.

342 **Theorem 1 (Set-dominance of Monotonic Branched Declare).** *Given*
 343 *the non-empty sets of activities R_1, \dots, R_n and S_1, \dots, S_n of a log alphabet Σ*
 344 *such that $\Sigma \supseteq S_i \supseteq R_i$ for every $i \in [1, n]$, a log L and a Monotonic Branched*
 345 *Declare template \mathcal{C} , then the support of $C' = \mathcal{C}(S_1, \dots, S_n)$ is greater than or*
 346 *equal to the support of $C = \mathcal{C}(R_1, \dots, R_n)$, i.e., $\mathcal{S}_L(C') \supseteq \mathcal{S}_L(C)$.*

347 **Proof 1.** *Because the log L over which the support is evaluated is the same*
 348 *for both constraints, we focus on the number of fulfilments, namely $\checkmark_L(C)$ and*
 349 *$\checkmark_L(C')$, for constraints C and C' . By definition of Monotonic Branched De-*
 350 *clare, if $S_i \supseteq R_i$ for every $i \in [1, n]$, where n is the arity of constraint template*
 351 *\mathcal{C} , then $C \models C'$. Therefore, due to the definition of model for a constraint*
 352 *w.r.t. a log, we have that $\checkmark_L(C) \leq \checkmark_L(C')$. The proof proceeds per absurdo. If*
 353 *$\checkmark_L(C) > \checkmark_L(C')$, there would necessarily exist at least a case that verifies C*
 354 *but not C' . This would contradict the fact that $C \models C'$. \square*

355 **Lemma 1 (Monotonicity of TBDeclare).** *Target-Branched Declare be-*
 356 *longs to the class of Monotonic Branched Declare, i.e., given an activity a*
 357 *in the log alphabet Σ , two non-empty sets of activities S and S' such that*
 358 *$S \subseteq S' \subseteq \Sigma$, and a TBDeclare template \mathcal{C} , then $\mathcal{C}(a, S) \models \mathcal{C}(a, S')$.*

359 **Proof 2.** *In the base case, $S = S' = \{b_1, \dots, b_n\}$. Therefore, $\mathcal{C}(a, S) \equiv \mathcal{C}(a, S')$.*
 360 *For the proof in the inductive case $S' = S \cup \{b_{n+1}\}$ where $b_{n+1} \notin S$, we resort*
 361 *on the fact that the semantics of constraint templates of Declare are expressible*
 362 *by means of LTL_f . Among operators used in LTL_f , \neg is known to be anti-*
 363 *monotonic, whereas all the other LTL_f operators are monotonic. The target of*
 364 *a Declare unidirectional positive relation constraint template always falls under*
 365 *an even number of \neg operators. By definition of TBDeclare, only the target is*
 366 *meant to be replaced by elements of the boolean join-semilattice $\langle \Sigma, \vee \rangle$. Hence,*
 367 *the target set-parameter always lets the activities assigned fall under an even*
 368 *number of negations. This guarantees the monotonicity of the constraint, due*
 369 *to the principle of non-contradiction.¹ \square*

370 The section now proceeds with the application of the inductive part of the
 371 proof to each template under examination, listed in Table 1.

372 *RespondedExistence.* $\text{RespondedExistence}(a, S') \equiv \diamond a \rightarrow \diamond (\bigvee_{i=1}^n b_i \vee b_{n+1})$.
 373 Recalling that, given two LTL_f formulae φ and ψ :

374 (a) $\varphi \rightarrow \psi \equiv \neg \varphi \vee \psi$, and

375 (b) $\diamond(\varphi \vee \psi) \equiv \diamond \varphi \vee \diamond \psi$,

¹Given a boolean formula φ , $\neg(\neg \varphi) \equiv \varphi$.

376 we have that $RespondedExistence(a, S') \equiv \neg \diamond a \vee (\bigvee_{i=1}^n \diamond b_i) \vee \diamond b_{n+1}$. Con-
 377 sequently, $RespondedExistence(a, S') \equiv RespondedExistence(a, S) \vee \diamond b_{n+1}$.
 378 Given two LTL_f formulae φ and ψ

379 (c) $\varphi \models \varphi \vee \psi$

380 due to the monotonicity of \vee . Therefore, Lemma 1 for *RespondedExistence* is
 381 proven.

382 *Response*. $Response(a, S') \equiv \square(\neg a \vee \diamond(\bigvee_{i=1}^n b_i) \vee \diamond b_{n+1})$ due to (a) and (b).
 383 We have also that:

384 (d) if $\varphi \models \psi$, then $\square\varphi \models \square\psi$

385 for the monotonicity of the temporal operators in LTL_f. Therefore,
 386 $\square\varphi \models \square(\varphi \vee \psi)$, because of (c). Since $Response(a, S) \equiv \square(\neg a \vee \diamond(\bigvee_{i=1}^n b_i))$,
 387 we have that Lemma 1 holds true for *Response*.

388 *AlternateResponse*. As a consequence of the application of (a),
 389 $AlternateResponse(a, S') \equiv \square(\neg a \vee \bigcirc(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i \vee b_{n+1})))$, whereas
 390 $AlternateResponse(a, S) \equiv \square(\neg a \vee \bigcirc(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i)))$.

391 Given the LTL_f formulae φ , ψ and ψ' ,

392 (e) if $\psi \models \psi'$, then $\varphi \mathcal{U} \psi \models \varphi \mathcal{U} \psi'$

393 due to the monotonicity of the temporal operators in LTL_f. Therefore, we have
 394 that $(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i)) \models (\neg a \mathcal{U}(\bigvee_{i=1}^n b_i \vee b_{n+1}))$, because of (c).

395 Furthermore, given two LTL_f formulae φ and ψ ,

396 (f) if $\varphi \models \psi$, then $\bigcirc\varphi \models \bigcirc\psi$

397 due to the monotonicity of the temporal operators in LTL_f. As a consequence,
 398 $\bigcirc(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i)) \models \bigcirc(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i \vee b_{n+1}))$.

399 Given the LTL_f formulae φ , ψ and ψ' ,

400 (g) if $\psi \models \psi'$, then $\varphi \vee \psi \models \varphi \vee \psi'$.

401 This leads to the conclusion that Lemma 1 holds true for *AlternateResponse*,
 402 considering (d).

403 *ChainResponse*. Given two LTL_f formulae φ and ψ , we have that:

404 (h) $\bigcirc(\varphi \vee \psi) \equiv \bigcirc\varphi \vee \bigcirc\psi$.

405 Applying (a) and (h), we have that $ChainResponse(a, S') \equiv \square(\neg a \vee \bigcirc(\bigvee_{i=1}^n b_i) \vee \bigcirc b_{n+1})$,
 406 whereas $ChainResponse(a, S) \equiv \square(\neg a \vee \bigcirc(\bigvee_{i=1}^n b_i))$. Lemma 1 is proven for
 407 *ChainResponse* then, due to (c) and (d).

408 *Precedence*. By definition of \mathcal{W} , we have that
 409 $Precedence(S', a) \equiv (\square\neg a) \vee (\neg a \mathcal{U}(\bigvee_{i=1}^n b_i \vee b_{n+1}))$, and
 410 $Precedence(S, a) \equiv (\square\neg a) \vee (\neg a \mathcal{U}(\bigvee_{i=1}^n b_i))$. Lemma 1 naturally ex-
 411 tends to the case of *Precedence* by applying (c), since it is already proven
 412 that $(\neg a \mathcal{U}(\bigvee_{i=1}^n b_i)) \models (\neg a \mathcal{U}(\bigvee_{i=1}^n b_i \vee b_{n+1}))$ (see demonstration for
 413 *AlternateResponse*).

414 *AlternatePrecedence*. For what *AlternatePrecedence* is regarded, the two terms
 415 of the conjunction have to be considered separately. The first term refers to
 416 *Precedence*, and it is already proven that $\text{Precedence}(S, a) \models \text{Precedence}(S', a)$.
 417 The second term is $\Box(\neg a \vee \bigcirc \text{Precedence}(S, a))$ for *AlternatePrecedence*(S, a)
 418 and $\Box(\neg a \vee \bigcirc \text{Precedence}(S', a))$ for *AlternatePrecedence*(S', a), due to (a). As
 419 a consequence, $\Box(\neg a \vee \bigcirc \text{Precedence}(S, a)) \models \Box(\neg a \vee \bigcirc \text{Precedence}(S', a))$,
 420 due to (c) and (d). As a conclusion, since it is known that

421 (i) if $\varphi \models \varphi'$ and $\psi \models \psi'$ then $\varphi \wedge \psi \models \varphi' \wedge \psi'$

422 we can conclude that Lemma 1 holds true for *AlternatePrecedence*.

423 *ChainPrecedence*. We have that $\text{ChainPrecedence}(a, S') \equiv \Box(\neg \bigcirc a \vee (\bigvee_{S \in S} b_i) \vee b_{n+1})$
 424 and $\text{ChainPrecedence}(a, S) \equiv \Box(\neg \bigcirc a \vee (\bigvee_{S \in S} b_i))$, due to (a). Considering
 425 (c) and (d), it is thus proven that Lemma 1 is verified.

426
 427 Following Theorem 1 describes the monotonic non-decreasing trend of the
 428 support for constraints w.r.t. set-containment of the target set of activities for
 429 TBDeclare.

430 **Corollary 1 (Set-dominance of TBDeclare)**. *Given an activity a in the*
 431 *log alphabet Σ , two non-empty sets of activities S, S' such that $\Sigma \supseteq S' \supseteq S$,*
 432 *a log L and a TBDeclare template \mathcal{C} , then the support of $\mathcal{C}(a, S')$ is greater than*
 433 *or equal to the support of $\mathcal{C}(a, S)$, i.e., $\mathcal{S}_L(\mathcal{C}(a, S')) \geq \mathcal{S}_L(\mathcal{C}(a, S))$.*

434 **Proof 3.** *Directly follows from Theorem 1 and Lemma 1.* □

435 As a final remark, we highlight that the notion of support introduced in Equa-
 436 tion (1) especially for relation constraints, is still compliant with Corollary 1 in
 437 the light of the proof of Theorem 1. In fact, it still holds that the denominator
 438 of the proportion remains the same for both constraints, as the activations are
 439 the same along the log, and the activations that do not violate $\mathcal{C}(a, S')$ cannot
 440 be less than the ones of $\mathcal{C}(a, S)$. Otherwise, if at least a fulfilment of $\mathcal{C}(a, S)$
 441 were not a fulfilment of $\mathcal{C}(a, S')$, it would constitute a counterexample against
 442 Lemma 1, according to which $\mathcal{C}(a, S) \models \mathcal{C}(a, S')$.

443
 444 In the following section, we show how the discovery algorithm exploits the
 445 fact that the support of TBDeclare is monotonously non-decreasing w.r.t. the
 446 set-containment relation of target set-parameters.

447 4. Discovery

448 This section describes MINERful for Target-Branched Declare (TB-
 449 MINERful), a three-step algorithm that, starting from an input log L , (i) builds
 450 a knowledge base, which keeps statistics on activity occurrences in L ; (ii) queries
 451 the knowledge base for support and confidence of constraints in L ; (iii) prunes
 452 constraints not having sufficient support and confidence. The input of the al-
 453 gorithm is a log L . Three thresholds can be specified: (i) *branching factor*, i.e.,

454 the maximum branching factor allowed for the discovered constraints, (ii) *min-*
 455 *imum support*, and (iii) *minimum confidence*.

456 4.1. The Knowledge Base

457 The first step is the construction of a knowledge base, which keeps statistics
 458 on the occurrences of activities in the log. It comprises the 9 functions listed
 459 further below in this section. \emptyset and $\#$ were already outlined in Section 2.3 and
 460 are here formally defined for the sake of completeness.

461 Following the same rationale of the symbology introduced in Section 2.2,
 462 set-parameters are here indicated with symbols $S, T \subseteq \Sigma$. $S = \{b, c\}$ is possibly
 463 assigned with $\{b, c\}$.

464 Let $\mathfrak{K} = \{\vdash, \dashv, \# \dashv, \blacktriangleright, \blacktriangleleft, \curvearrowright, \curvearrowleft\}$ and $\mathfrak{K}_1 = \{\emptyset, \#\}$ be two sets of functions
 465 defined as follows. In the examples, $=_\nu$ and $=_{\nu_1}$ specify the number that would
 466 be assigned to the functions respectively in \mathfrak{K} and \mathfrak{K}_1 , given a log. As an example
 467 log we use $L = \{\langle a, a, b, a, c, a \rangle, \langle a, a, b, a, c, a, d \rangle\}$ defined over $\Sigma = \{a, b, c, d\}$.

468 $\emptyset : \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\emptyset(a, L)$ (hereinafter, $\emptyset_L(a)$ for short) counts
 469 the traces of $L \in \mathfrak{M}(\Sigma^*)$ in which $a \in \Sigma$ did not occur. For instance,
 470 $\emptyset_L(a) =_{\nu_1} 0$, because a occurs in every trace in L . $\emptyset_L(d) =_{\nu_1} 1$, instead.

471 $\# : \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\#(a, L)$ (hereinafter, $\#_L(a)$ for short) counts
 472 the occurrences of $a \in \Sigma$ in $L \in \mathfrak{M}(\Sigma^*)$. Therefore, $\#_L(a) =_{\nu_1} 8$.

473 $\vdash : \Sigma \times \wp(\Sigma) \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$.² Function $\vdash(a, S, L)$ (hereinafter, $\vdash_L(a, S)$ for
 474 short) counts the occurrences of $a \in \Sigma$ with no following $b \in S =$
 475 $\{b_1, \dots, b_\beta\}$ (for any $\beta \in [1, |\Sigma|]$) in the traces of $L \in \mathfrak{M}(\Sigma^*)$. In the
 476 example, $\vdash_L(a, \{d\}) =_\nu 4$, $\vdash_L(a, \{b\}) =_\nu 4$, and $\vdash_L(a, \{b, c\}) =_\nu 2$.

477 $\dashv : \Sigma \times \wp(\Sigma) \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\dashv(a, S, L)$ (hereinafter, $\dashv_L(a, S)$ for
 478 short) counts the occurrences of $a \in \Sigma$ with no preceding $b \in S =$
 479 $\{b_1, \dots, b_\beta\}$ (for any $\beta \in [1, |\Sigma|]$) in the traces of $L \in \mathfrak{M}(\Sigma^*)$. Thus,
 480 e.g., $\dashv_L(a, \{d\}) =_\nu 8$, $\dashv_L(a, \{b\}) =_\nu 4$, and $\dashv_L(a, \{b, c\}) =_\nu 4$.

481 $\# \dashv : \Sigma \times \wp(\Sigma) \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\# \dashv(a, S, L)$ (hereinafter, $\# \dashv_L(a, S)$ for
 482 short) counts the occurrences of $a \in \Sigma$ with no co-occurring $b \in S =$
 483 $\{b_1, \dots, b_\beta\}$ (for any $\beta \in [1, |\Sigma|]$) in the traces of $L \in \mathfrak{M}(\Sigma^*)$. Therefore,
 484 $\# \dashv_L(a, \{d\}) =_\nu 4$, and $\# \dashv_L(a, \{b, d\}) =_\nu 0$.

485 $\blacktriangleright : \Sigma \times \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\blacktriangleright(a, b, L)$ (hereinafter, $\blacktriangleright_L(a, b)$ for short)
 486 counts the occurrences of $a \in \Sigma$ having $b \in \Sigma$ as the next event in the
 487 traces of $L \in \mathfrak{M}(\Sigma^*)$. Hence, e.g., $\blacktriangleright_L(a, b) =_\nu 2$, and $\blacktriangleright_L(a, d) =_\nu 1$.

488 $\blacktriangleleft : \Sigma \times \Sigma \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\blacktriangleleft(a, b, L)$ (hereinafter, $\blacktriangleleft_L(a, b)$ for short)
 489 counts the occurrences of $a \in \Sigma$ having $b \in \Sigma$ as the preceding event in the
 490 traces of $L \in \mathfrak{M}(\Sigma^*)$. In the example, $\blacktriangleleft_L(a, b) =_\nu 2$, and $\blacktriangleleft_L(a, d) =_\nu 0$.

²By $\wp(\Sigma)$, we mean the power set of Σ .

491 $\varrho: \Sigma \times \wp(\Sigma) \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\varrho(a, S, L)$ (hereinafter, $\varrho_L(a, S)$ for
 492 short) counts how many times, after an occurrence of $a \in \Sigma$, a repeats
 493 until the first $b \in S = \{b_1, \dots, b_\beta\}$ follows in the same trace, for all the
 494 traces of $L \in \mathfrak{M}(\Sigma^*)$. If no $b \in S$ appears in the trace after a , the
 495 repetitions after a are not counted. In the example, $\varrho_L(a, \{b\}) =_\nu 2$,
 496 $\varrho_L(a, \{c\}) =_\nu 4$, $\varrho_L(a, \{b, c\}) =_\nu 2$, and $\varrho_L(a, \{b, d\}) =_\nu 3$.

497 $\leftarrow\varrho: \Sigma \times \Sigma^\beta \times \mathfrak{M}(\Sigma^*) \rightarrow \mathbb{N}$. Function $\leftarrow\varrho(a, S, L)$ (hereinafter, $\leftarrow\varrho_L(a, S)$ for
 498 short) is similar to $\varrho_L(a, S)$, but reading the traces of $L \in \mathfrak{M}(\Sigma^*)$ con-
 499 trariwise. Thus, $\leftarrow\varrho_L(a, \{b\}) =_\nu 2$, $\leftarrow\varrho_L(a, \{c\}) =_\nu 0$, $\leftarrow\varrho_L(a, \{b, c\}) =_\nu 0$,
 500 and $\leftarrow\varrho_L(a, \{b, d\}) =_\nu 2$.

501 The knowledge base is thus a tuple $\mathcal{KB} = \langle \Sigma, L, \mathfrak{K}, \mathfrak{K}_1, \nu, \nu_1 \rangle$, consisting of a
 502 log alphabet Σ , a log $L \in \mathfrak{M}(\Sigma^*)$, two sets of functions \mathfrak{K} and \mathfrak{K}_1 , and two
 503 interpretation functions $\nu_1: \mathfrak{K}_1 \times \Sigma \times L \rightarrow \mathbb{N}$ and $\nu: \mathfrak{K} \times \Sigma \times \wp(\Sigma) \times L \rightarrow \mathbb{N}$.
 504 ν_1 assigns an integer value to the functions of the knowledge base that pertain
 505 to a single activity, for all the activities in the log alphabet, on the basis of the
 506 given log. ν assigns an integer value to the functions of the knowledge base
 507 that pertain to the interplay of every activity with all subsets of other activities
 508 in the log alphabet, on the basis of the given log. Next, we discuss how the
 509 knowledge base is built based on an input log.

510 4.2. Building the Knowledge Base

511 The objective of the algorithm for building the knowledge base formally is
 512 the definition of the interpretation functions that is consistent with the given
 513 log and log alphabet. To this extent, we adopt different approaches for different
 514 functions. However, the common characteristic is that they do not need more
 515 than one parse of the traces of the log to update the knowledge base. This leads
 516 to a reduction in the computation time. In particular, it makes the algorithm
 517 linear w.r.t. the number of traces.

518 The rationale behind the technique is that the parsing of the log is done for
 519 counting (i) the occurrences and misses of single activities $a \in \Sigma$, and (ii) the
 520 co-occurrences and misses of pairs of activities $a, b \in \Sigma$ in each trace. Variables
 521 storing such counts will be named (i) *singleton counters* and (ii) *pairwise coun-*
 522 *ters*, respectively. Singleton and pairwise counters refer to specific elements of
 523 the knowledge base. For the sake of readability, counters will be henceforth
 524 identified by a \mathbb{N} (tele-type) letter, indexed by the (parametric) activities that
 525 they consider. The symbol put at the apex specifies the element of the know-
 526 ledge base for which the counter is meant to be utilised. For instance, singleton
 527 counter $N_a^\#$ counts the total number of occurrences of a in the log. In log
 528 $L = \{\langle a, a, b, a, c, a \rangle, \langle a, a, b, a, c, a, d \rangle\}$, $N_a^\# = 8$. Pairwise counter $N_{a,b}^\#$ is ded-
 529 icated to counting the occurrences of a in a trace, after which no b occurs. In log
 530 L , $N_{a,b}^\# = 4$. $N_{a,b}^\#$ is discussed in detail in Section 4.2.2.

531 Pairwise counters do not take into account the relation of an activity a with
 532 *sets* of other activities, though. On the other hand, computing a value for each
 533 $S \in \wp(\Sigma \setminus \{a\})$ would be impractical. Therefore, we build *differential cumulative*

534 set-counters. They are named “cumulative” because they derive co-occurrences
 535 and misses of activity a with sets of activities $S = \bigcup_{i=1}^{\beta} \{b_i : b_i \in \Sigma \setminus \{a\}\}$ with
 536 $\beta \in [1, |\Sigma|]$, starting from the values of single pairwise counters that refer to
 537 pairs of activities a, b_i . They are qualified as “differential” due to the fact that
 538 they store values by differences. In the remainder, differential cumulative set-
 539 counters will be identified by symbol Δ (indicating the differential nature), put
 540 in front of the pairwise counter from which they are derived. For instance, $\Delta N_{a,S}^{\dagger}$
 541 is a differential cumulative set-counter that stores the (differential) number of
 542 cases in which a is not followed by any of the activities in S .

543 Co-inductively, given $S \subseteq \Sigma \setminus \{a\}$, $\Delta N_{a,S}^{\dagger}$ reports the difference between
 544 (i) the number of times in which no $b \in S$ occurred and (ii) $\sum_{T \supseteq S} \Delta N_{a,T}^{\dagger}$, hav-
 545 ing $T \subseteq \Sigma \setminus \{a\}$. After parsing log L , we thus have the following values:

546 (i) $\Delta N_{a,\{b\}}^{\dagger} = 1$, (ii) $\Delta N_{a,\{b,c\}}^{\dagger} = 1$, (iii) $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, (iv) $\Delta N_{a,\{b,d\}}^{\dagger} = 1$,
 547 (v) $\Delta N_{a,\{d\}}^{\dagger} = 2$.

548 In fact, none of the activities in $\{b, c\}$ occurred after a in 2 cases. It also
 549 holds true that none of the activities in $\{b, c, d\}$ occurred after a in 1 case, and
 550 $\{b, c\} \subseteq \{b, c, d\}$. Therefore,

551 $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, and $\Delta N_{a,\{b,c\}}^{\dagger} = 1$, i.e., $\Delta N_{a,\{b,c\}}^{\dagger} = 2 - \Delta N_{a,\{b,c,d\}}^{\dagger}$.

552 By the same line of reasoning, since b did not occur after a in 4 cases, $\Delta N_{a,\{b\}}^{\dagger} = 1$,
 553 i.e., $\Delta N_{a,\{b\}}^{\dagger} = 4 - \Delta N_{a,\{b,c\}}^{\dagger} - \Delta N_{a,\{b,d\}}^{\dagger} - \Delta N_{a,\{b,c,d\}}^{\dagger}$.

554 The next section explains the procedure computing such values in detail.

555 The differential cumulative set-counters are used to compactly store the
 556 values to assign to interpretation functions. In the case of \dagger , it is done as
 557 follows:

$$\dagger_L(a, S) =_{\nu} \sum_{T \supseteq S} \Delta N_{a,T}^{\dagger}$$

558 In the example log, indeed,

559 $\dagger_L(a, \{b\}) =_{\nu} 4$, and $\dagger_L(a, \{b, c\}) =_{\nu} 2$,

560 i.e.,

561 $\dagger_L(a, \{b\}) =_{\nu} \Delta N_{a,\{b\}}^{\dagger} + \Delta N_{a,\{b,c\}}^{\dagger} + \Delta N_{a,\{b,c,d\}}^{\dagger}$, and

562 $\dagger_L(a, \{b, c\}) =_{\nu} \Delta N_{a,\{b,c\}}^{\dagger} + \Delta N_{a,\{b,c,d\}}^{\dagger}$.

563 4.2.1. The main algorithm

564 Algorithm 1 shows the main algorithm that leads to the building of the
 565 knowledge base, based on a log alphabet Σ over a log L .

566 *Notations and conventions.* In the remainder of this section, we will assume
 567 that the concatenation operator \circ is defined for sequences, i.e., given a sequence
 568 $\vec{s} = \langle s_1, \dots, s_{|\vec{s}|} \rangle$ and an element s' , then $\vec{s} \circ s' = \langle s_1, \dots, s_{|\vec{s}|}, s' \rangle$. Since a trace
 569 of a log is defined as a sequence of events, \circ also applies to appending events to
 570 traces. If we indicate with $\mathfrak{k} \in \mathfrak{K}$ the generic function of the set of functions \mathfrak{K} ,
 571 the generic pairwise counter on activities $a, b \in \Sigma$ will be denoted as $N_{a,b}^{\mathfrak{k}}$, and
 572 the generic differential cumulative set-counter on $a \in \Sigma, S \subseteq \Sigma$ as $\Delta N_{a,S}^{\mathfrak{k}}$. As a

Algorithm 1: EVALUATEKB(Σ, L), the main algorithm for the building of the knowledge base

Input: A log alphabet Σ and a log $L = \langle \vec{t}_1, \dots, \vec{t}_{|L|} \rangle \in \Sigma^*$, where $\vec{t}_i = \langle t_1, \dots, t_{|\vec{t}_i} \rangle$
Output: The knowledge base, whose values are assigned on the basis of Σ and L

```

1 for  $i \leftarrow 1$  to  $|L|$  do
2      $\vec{t}_i^R \leftarrow \vec{t}_i$  with events in reverse order
3     foreach  $a \in \Sigma, b \in \Sigma \setminus \{a\}$  do // Reset of pairwise counters
4          $N_{a,b}^\# \leftarrow 0; N_{a,b}^\# \leftarrow 0; N_{a,b}^{\# \rightarrow} \leftarrow 0; N_{a,b}^{\# \leftarrow} \leftarrow 0$ 
5         // Update of singleton counters */
6         foreach  $a \in \Sigma : a \notin \vec{t}_i$  do // Activities not occurring in trace  $\vec{t}_i$ 
7              $N_a^\emptyset \leftarrow N_a^\emptyset + 1$ 
8         for  $j \leftarrow 1$  to  $|\vec{t}_i|$  do
9              $a \leftarrow t_{i,j}$ 
10             $N_a^\# \leftarrow N_a^\# + 1$ 
11            // Update of pairwise counters and differential cumulative set-counters */
12            foreach  $\Delta N_{a,S}^\# \in \text{EVALMISSINGAFTER}(\Sigma, \vec{t}_i)$  do  $\Delta N_{a,S}^\# \leftarrow \Delta N_{a,S}^\# \boxplus \Delta N_{i,a,S}^\#$ 
13            foreach  $\Delta N_{i,a,S}^\# \in \text{EVALMISSINGBEFORE}(\Sigma, \vec{t}_i^R)$  do  $\Delta N_{a,S}^\# \leftarrow \Delta N_{a,S}^\# \boxplus \Delta N_{i,a,S}^\#$ 
14            foreach  $\Delta N_{a,S}^\# \in \text{EVALMISSING}(\Sigma, \vec{t}_i)$  do  $\Delta N_{a,S}^\# \leftarrow \Delta N_{a,S}^\# \boxplus \Delta N_{i,a,S}^\#$ 
15            foreach  $\Delta N_{a,S}^{\# \rightarrow} \in \text{EVALFOLLOWINGREPSINBETWEEN}(\Sigma, \vec{t}_i)$  do  $\Delta N_{a,S}^{\# \rightarrow} \leftarrow \Delta N_{a,S}^{\# \rightarrow} \boxplus \Delta N_{i,a,S}^{\# \rightarrow}$ 
16            foreach  $\Delta N_{i,a,S}^{\# \leftarrow} \in \text{EVALPRECEDINGREPSINBETWEEN}(\Sigma, \vec{t}_i^R)$  do  $\Delta N_{a,S}^{\# \leftarrow} \leftarrow \Delta N_{a,S}^{\# \leftarrow} \boxplus \Delta N_{i,a,S}^{\# \leftarrow}$ 
17            foreach  $N_{i,a,b}^{\# \rightarrow} \in \text{EVALFOLLOWING}(\Sigma, \vec{t}_i)$  do  $N_{a,b}^{\# \rightarrow} \leftarrow N_{a,b}^{\# \rightarrow} + N_{i,a,b}^{\# \rightarrow}$ 
18            foreach  $N_{i,a,b}^{\# \leftarrow} \in \text{EVALPRECEDING}(\Sigma, \vec{t}_i^R)$  do  $N_{a,b}^{\# \leftarrow} \leftarrow N_{a,b}^{\# \leftarrow} + N_{i,a,b}^{\# \leftarrow}$ 

```

573 shorthand notation for sets of pairwise counters referring to all $a \in \Sigma, b \in \Sigma \setminus \{a\}$,
574 we will adopt the usual pairwise counter notation, having a $\forall\forall$ pedix in place of
575 the referred activities (e.g., $N_{\forall\forall}^\# = \bigcup_{a \in \Sigma, b \in \Sigma \setminus \{a\}} N_{a,b}^\#$).

576 *Description of the algorithm.* The algorithm iterates over every trace of L . At
577 every iteration i , the pairwise counters are reset to 0, and a variable \vec{t}_i^R keeps
578 a clone of the trace under analysis, with events reversed in their original order.
579 Thereafter, singleton counters are updated. For every activity $a \in \Sigma$ that does
580 not occur in the trace under analysis, N_a^\emptyset is incremented by 1. In the sample
581 trace $\langle \mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{a}, \mathbf{c}, \mathbf{a} \rangle$, $N_{\mathbf{a}}^\emptyset = 0$, because \mathbf{a} occurs in it. $N_{\mathbf{d}}^\emptyset = 1$ instead. For each
582 activity a , counter $N_a^\#$ is incremented by 1 every time a occurs. Thus, $N_{\mathbf{a}}^\# = 4$
583 in the sample trace, since \mathbf{a} occurs 4 times, whereas $N_{\mathbf{d}}^\# = 0$. Consequently, we
584 have the following, for every $a \in \Sigma$:

$$\#_L(a) =_{\nu_1} N_a^\# \quad (3)$$

585 and

$$\emptyset_L(a) =_{\nu_1} N_a^\emptyset. \quad (4)$$

586
587

Algorithm 2: Procedure PAIRWISE2DIFF($\Sigma, N_{\forall v}^t$), deriving differential cumulative set-counters from pairwise counters.

Input: A log alphabet Σ and a set of pairwise counters $N_{\forall v}^t$
Output: A set of differential cumulative set-counters derived from $N_{\forall v}^t$

```

1 foreach  $a \in \Sigma$  do
2   for  $n \leftarrow 1$  to  $\max_{b \in \Sigma} \{N_{a,b}^t \subseteq N_{\forall v}^t\}$  do
3      $S_n \leftarrow \{b : N_{a,b}^t \geq n\}$ 
4      $\vec{n} = \{n : S_n \neq \emptyset\}$  sorted by  $n$  descending
5     for  $i \leftarrow 1$  to  $|\vec{n}| - 1$  do
6        $n \leftarrow \vec{n}_i$ 
7        $n' \leftarrow \vec{n}_{i+1}$ 
8        $\Delta N_{a,S_n}^t \leftarrow (n - n')$ 
9 return  $\bigcup_{\substack{a \in \Sigma \\ n \in \vec{n}}} \Delta N_{a,S_n}^t$ 

```

588 The computation of pairwise counters and corresponding differential cumulative
589 set-counters are generally less trivial. Therefore, separate subsections
590 follow that describe each dedicated procedure (EVALMISSINGAFTER, EVAL-
591 MISSINGBEFORE, ...). All such procedures except EVALFOLLOWING and
592 EVALPRECEDING return new sets of differential cumulative set-counters (each
593 identified as $\Delta N_{i,a,S}^{\dagger}$, $\Delta N_{i,a,S}^{\ddagger}$, ...). Each element of these new sets are used to update
594 the current value of the corresponding differential cumulative set-counter.
595 We assume that all differential cumulative set-counters are initially assigned
596 with a default value of 0. The addition operation over differential cumulative
597 set-counters, \boxplus , is defined as follows:

$$\Delta N_{a,S}^t \boxplus \Delta N_{a',S'}^t = \begin{cases} \Delta N_{a,S}^t + \Delta N_{a',S'}^t & \text{if } a = a' \text{ and } S = S' \\ \Delta N_{a,S}^t & \text{otherwise} \end{cases}$$

598 Given an example log $L = \{\langle a, a, b, a, c, a \rangle, \langle a, a, b, a, c, a, d \rangle, \langle c, a, a, d \rangle\}$, the
599 parsing of the first trace leads to the following values of the differential
600 cumulative set-counters referred to activity a :

601 (i) $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, (ii) $\Delta N_{a,\{b,d\}}^{\dagger} = 1$, and (iii) $\Delta N_{a,\{d\}}^{\dagger} = 2$.

602 After the analysis of the second trace, we have:

603 (i) $\Delta N_{a,\{b\}}^{\dagger} = 1$, (ii) $\Delta N_{a,\{b,c\}}^{\dagger} = 1$, (iii) $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, (iv) $\Delta N_{a,\{b,d\}}^{\dagger} = 1$, and

604 (v) $\Delta N_{a,\{d\}}^{\dagger} = 2$.

605 Finally, the third trace leads to the following values:

606 (i) $\Delta N_{a,\{b\}}^{\dagger} = 1$, (ii) $\Delta N_{a,\{b,c\}}^{\dagger} = 3$, (iii) $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, (iv) $\Delta N_{a,\{b,d\}}^{\dagger} = 1$,

607 (v) $\Delta N_{a,\{d\}}^{\dagger} = 2$.

608

609 Procedures EVALFOLLOWING and EVALPRECEDING return instead sets of
610 pairwise counters (each identified as $N_{i,a,b}^{\leftarrow}$ and $N_{i,a,b}^{\rightarrow}$). Therefore, the update
611 operation is an addition. The following subsections explain in detail all the
612 procedures that compute values for pairwise counters and differential cumulative

Algorithm 3: Procedure EVALMISSINGAFTER(Σ, \vec{t}), evaluating subsequent missing occurrences of activities in a trace

Input: A log alphabet Σ and a trace $\vec{t} = \langle t_1, \dots, t_{|\vec{t}|} \rangle$
Output: The set of differential cumulative set-counters ΔN^{\ddagger} derived from trace \vec{t}

```

1 for  $i \leftarrow 1$  to  $|\vec{t}|$  do
2    $b \leftarrow t_i$ 
3   foreach  $a \in \Sigma \setminus \{b\}$  do
4      $N_{a,b}^{\ddagger} \leftarrow 0$  // Flush operation ↓
5      $N_{b,a}^{\ddagger} \leftarrow N_{b,a}^{\ddagger} + 1$ 
6  $N_{\forall\forall}^{\ddagger} \leftarrow \bigcup_{a \in \Sigma, b \in \Sigma \setminus \{a\}} N_{a,b}^{\ddagger}$ 
7 return PAIRWISE2DIFF( $\Sigma, N_{\forall\forall}^{\ddagger}$ )

```

	Trace						$N_{a,\cdot}^{\ddagger}$			$\Delta N_{a,\cdot}^{\ddagger}$		
	a	a	b	a	c	a	$N_{a,b}^{\ddagger} =$	$N_{a,c}^{\ddagger} =$	$N_{a,d}^{\ddagger} =$	$\Rightarrow \Delta N_{a,\{b,c,d\}}^{\ddagger} =$		
$N_{a,b}^{\ddagger}$	1	2	↓	1		2	1 +	1	1 +	$\Rightarrow \Delta N_{a,\{b,c,d\}}^{\ddagger} =$	1	
$N_{a,c}^{\ddagger}$	1	2		3	↓	1	1 =		1 +	$\Rightarrow \Delta N_{a,\{b,d\}}^{\ddagger} =$	1	
$N_{a,d}^{\ddagger}$	1	2		3		4	2		2 =	$\Rightarrow \Delta N_{a,\{d\}}^{\ddagger} =$	2	
									4			

(a) Computation of $N_{a,\cdot}^{\ddagger}$. (b) Computation of $\Delta N_{a,\cdot}^{\ddagger}$, given the values of $N_{a,\cdot}^{\ddagger}$.

Table 3: Computation of $N_{a,\cdot}^{\ddagger}$ and $\Delta N_{a,\cdot}^{\ddagger}$, given a sample trace: $\langle a, a, b, a, c, a \rangle$.

613 set-counters. Each subsection concludes with the assignment of the formulation
614 of the interpretation function, on the basis of the referring pairwise counter or
615 differential cumulative set-counter.

616 4.2.2. Count of missing events after an activity

617 For evaluating $\ddagger_L(a, S)$, procedure EVALMISSINGAFTER computes for every
618 $b \in \Sigma \setminus \{a\}$ the value $N_{a,b}^{\ddagger}$. Algorithm 3 lists its pseudocode. Table 3a shows how
619 $N_{a,\cdot}^{\ddagger}$ values are computed for $\langle a, a, b, a, c, a \rangle$. $N_{a,b}^{\ddagger}$ is incremented by 1 every time
620 a is read, while parsing the trace. When b is read, $N_{a,b}^{\ddagger}$ is reset to 0. The ↓
621 symbol indicates this operation (“flush”). At the end of the trace, the value
622 stored in $N_{a,b}^{\ddagger}$ reports the occurrences of a after which no b occurred. In the
623 example, we have $N_{a,b}^{\ddagger} = 2$, $N_{a,c}^{\ddagger} = 1$ and $N_{a,d}^{\ddagger} = 4$.

624 The output of the procedure is a set of differential cumulative set-counters,
625 obtained by invoking the PAIRWISE2DIFF procedure. Passing from pairwise
626 counters to differential cumulative set-counters is a linear procedure, whose
627 pseudocode is listed in Algorithm 2, for general sets of pairwise counters, and
628 sketched in Table 3b, especially for ΔN^{\ddagger} . For each $a \in \Sigma$, all pairwise counters
629 $N_{a,b}^{\ddagger}$ (for every $b \in \Sigma$) are indexed according to their value n . A set of activities
630 S_n contains those $b \in \Sigma$ such that $N_{a,b}^{\ddagger} \geq n$. In the example, considering a as

$\Delta N_{a,\cdot}^{\dagger}$	\Rightarrow	$\dagger_L(a, \cdot)$
$\{b, c, d\} = 1$	\Rightarrow	$\dagger_L(a, \{b, c, d\}) = \nu \dagger_L(a, \{c, d\}) = \nu \dagger_L(a, \{c\}) = 1$
$\{b, d\} = 1$	\Rightarrow	$\dagger_L(a, \{b, d\}) = \nu \dagger_L(a, \{b\}) = 2$
$\{d\} = 2$	\Rightarrow	$\dagger_L(a, \{d\}) = 4$

Table 4: Interpretation of $\dagger_L(a, \cdot)$ for a w.r.t. all subsets of log alphabet $\Sigma = \{a, b, c, d\}$, given $\Delta N_{a,\cdot}^{\dagger}$, for a w.r.t. $\{b, c, d\}$, $\{b, d\}$, and $\{d\}$

631 the assignment of a , $S_4 = \{d\}$, $S_2 = \{b, d\}$, and $S_1 = \{b, c, d\}$, because $N_{a,d}^{\dagger} = 4$,
632 $N_{a,b}^{\dagger} = 2$, and $N_{a,c}^{\dagger} = 1$. A sequence \vec{n} is thus created that stores the values of
633 pairwise counters in descending order. In the example, \vec{n} is $\{4, 2, 1\}$. Elements
634 of \vec{n} are meant to act as an index for sets S_n . All elements in the sequence are
635 indeed visited from the first to the second last. For each of them, a differential
636 cumulative set-counter $\Delta N_{a,S_n}^{\dagger}$ is created that associates a to S_n . The value of
637 $\Delta N_{a,S_n}^{\dagger}$ is assigned with $n - n'$, where n' is the following element in the list.

638 For example, in $\langle a, a, b, a, c, a \rangle$, we have that $\Delta N_{a,\{b,c,d\}}^{\dagger} = 1$, $\Delta N_{a,\{b,d\}}^{\dagger} = 1$,
639 and $\Delta N_{a,\{d\}}^{\dagger} = 2$. Table 3b shows the passage from $\{N_{a,b}^{\dagger}, N_{a,c}^{\dagger}, N_{a,d}^{\dagger}\}$ to $\Delta N_{a,\{b,c,d\}}^{\dagger}$,
640 $\Delta N_{a,\{b,d\}}^{\dagger}$ and $\Delta N_{a,\{d\}}^{\dagger}$ for the sample trace. It is straightforward to see that the
641 differential accumulation ($\Delta N_{a,S}^{\dagger}$) allows for keeping fewer values in memory (3 in
642 the example) than the possible entries for the knowledge base ($\dagger_L(a, S)$, which
643 amounts to 6). The memory saving is possible as we do not store information
644 about those $\Delta N_{a,S}^{\dagger}$ that amount to 0 as, for instance, $\Delta N_{a,\{c,d\}}^{\dagger}$ in the example of
645 Table 3b.

646 As previously said, the interpretation of $\dagger_L(a, S)$ can be derived from this
647 compact data structures as follows:

$$\dagger_L(a, S) = \nu \sum_{T \supseteq S} \Delta N_{a,T}^{\dagger} \quad (5)$$

648 Table 4 shows the application of this derivation step for the sample trace.

649 4.2.3. Count of missing events before the occurrence of an activity

650 The technique seen for $\dagger_L(a, S)$ extends to the computation of $\dagger_L(a, S)$
651 with slight modifications. In fact, $\dagger_L(a, S)$ executes the procedures described
652 above (i.e., computation of pairwise counters and derivation of differential cu-
653 mulative set-counters, for every trace), although reversing the order in which
654 the traces are parsed. We report the pseudocode in Algorithm 4 for the sake
655 of completeness. Thus, e.g., the pairwise counter $N_{a,b}^{\dagger}$ is assigned with values
656 in the same way in which $N_{a,b}^{\dagger}$ was computed, although parsing $\langle a, c, a, b, a, a \rangle$
657 in place of $\langle a, a, b, a, c, a \rangle$ (see Table 5). Thereafter, the differential cumulative
658 set-counter $\Delta N_{a,S}^{\dagger}$ is derived from $N_{a,b}^{\dagger}$, exactly as $\Delta N_{a,S}^{\dagger}$ is derived from $N_{a,b}^{\dagger}$.

Algorithm 4: Procedure EVALMISSINGBEFORE(Σ, \vec{t}^R), evaluating preceding missing occurrences of activities in a reversed trace.

Input: A log alphabet Σ and a reversed trace $\vec{t}^R = \langle t_1, \dots, t_{|\vec{t}^R|} \rangle$
Output: The set of differential cumulative set-counters ΔN^{\dagger} derived from reversed trace \vec{t}^R

```

1 for  $i \leftarrow 1$  to  $|\vec{t}^R|$  do
2    $b \leftarrow t_i$ 
3   foreach  $a \in \Sigma \setminus \{b\}$  do
4      $N_{a,b}^{\dagger} \leftarrow 0$  // Flush operation ↓
5      $N_{b,a}^{\dagger} \leftarrow N_{b,a}^{\dagger} + 1$ 
6  $N_{\forall \forall}^{\dagger} \leftarrow \bigcup_{a \in \Sigma, b \in \Sigma \setminus \{a\}} N_{a,b}^{\dagger}$ 
7 return PAIRWISE2DIFF( $\Sigma, N_{\forall \forall}^{\dagger}$ )

```

Reversed trace						$N_{a,\cdot}^{\dagger}$			$\Delta N_{a,\cdot}^{\dagger}$		
	a	c	a	b	a	a					
$N_{a,b}^{\dagger}$	1		2	↓	1	2	$N_{a,b}^{\dagger} = 2$	$N_{a,c}^{\dagger} = 2 +$	$N_{a,d}^{\dagger} = 2 +$	$\Rightarrow \Delta N_{a,\{b,c,d\}}^{\dagger} = 2$	
$N_{a,c}^{\dagger}$	1	↓	1		2	3		1 =	1 +	$\Rightarrow \Delta N_{a,\{c,d\}}^{\dagger} = 1$	
$N_{a,d}^{\dagger}$	1		2		3	4		3	1 =	$\Rightarrow \Delta N_{a,\{d\}}^{\dagger} = 1$	
									4		

(a) Computation of $N_{a,\cdot}^{\dagger}$. (b) Computation of $\Delta N_{a,\cdot}^{\dagger}$, given the values of $N_{a,\cdot}^{\dagger}$.

Table 5: Computation of $N_{a,\cdot}^{\dagger}$ and $\Delta N_{a,\cdot}^{\dagger}$, given a sample trace: $\langle a, a, b, a, c, a \rangle$.

659 After a trace has been completely parsed, $\Delta N_{a,S}^{\dagger}$ is \boxplus -added to the differential
660 cumulative set-counter. As a consequence, we have that:

$$\dagger_L(a, S) =_{\nu} \sum_{T \supseteq S} \Delta N_{a,T}^{\dagger} \quad (6)$$

661

662 4.2.4. Count of missing events in the same trace in which an activity occurs

663 For what $\dagger_L(a, S)$ is concerned, its computation is based on the differential
664 cumulative set-counter $\Delta N_{a,S}^{\dagger}$, in turn derived from pairwise counter $N_{a,b}^{\dagger}$. The
665 pseudocode is listed in Algorithm 5 (procedure EVALMISSING). $N_{a,b}^{\dagger}$ stores for
666 each trace and each $a \in \Sigma$ either 0, if b occurs in the trace at least once, or the
667 number of occurrences of a , if b did not occur in the trace. Referring to a and
668 trace $\langle a, a, b, a, c, a \rangle$, we have that: (i) $N_{a,b}^{\dagger} = 0$, (ii) $N_{a,c}^{\dagger} = 0$, and (iii) $N_{a,d}^{\dagger} = 4$.
669 The accumulation of $N_{a,b}^{\dagger}$ in $\Delta N_{a,S}^{\dagger}$ is performed in the same way seen for $\Delta N_{a,S}^{\dagger}$
670 and $\Delta N_{a,S}^{\dagger}$. It follows that:

$$\dagger_L(a, S) =_{\nu} \sum_{T \supseteq S} \Delta N_{a,T}^{\dagger} \quad (7)$$

Algorithm 5: Procedure EVALMISSING(Σ, \vec{t}), evaluating missing co-occurrences of activities in a trace

Input: A log alphabet Σ and a trace $\vec{t} = \langle t_1, \dots, t_{|\vec{t}|} \rangle$
Output: The set of differential cumulative set-counters $\Delta N^\#$ derived from trace \vec{t}

```

1  foreach  $a \in \Sigma, b \in \Sigma \setminus \{a\}$  do  $?_{a,b}^\# \leftarrow \top$ 
2  for  $i \leftarrow 1$  to  $|\vec{t}|$  do
3       $a \leftarrow t_i$ 
4      foreach  $a \in \Sigma \setminus \{b\}$  do
5          if  $?_{a,b}^\# = \top$  then
6               $N_{a,b}^\# \leftarrow N_{a,b}^\# + 1$ 
7          if  $?_{b,a}^\# = \top$  then
8               $N_{b,a}^\# \leftarrow 0$  // Flush operation  $\downarrow$ 
9               $?_{b,a}^\# \leftarrow \perp$ 
10  $N_{\forall\forall}^\# \leftarrow \bigcup_{a \in \Sigma, b \in \Sigma \setminus \{a\}} N_{a,b}^\#$ 
11 return PAIRWISE2DIFF ( $\Sigma, N_{\forall\forall}^\#$ )

```

671 4.2.5. Count of repeated occurrences of an activity before other events

672 For the computation of $\forall\rightarrow_L(a, S)$, we here present a far more efficient calcu-
673 lation as opposed to the one used in [8]. The new algorithm follows the general
674 framework seen so far (computation of values for pairwise counters $N_{a,b}^{\rightarrow}$ first,
675 then derivation of differential cumulative set-counters $\Delta N_{a,S}^{\rightarrow}$). Its pseudocode is
676 reported in Algorithm 6 (procedure EVALFOLLOWINGREPSINBETWEEN). The
677 input traces are sliced into sub-traces, at every new occurrence of a following
678 the first one. Given, e.g., the sample trace $\langle a, a, b, a, c, a, d \rangle$, it is sliced into the
679 following sub-traces (see Table 6): (i) $\langle a \rangle$, (ii) $\langle a, b \rangle$, (iii) $\langle a, c \rangle$, and (iv) $\langle a, d \rangle$.
680 Thereafter, pairwise counter $N_{a,b}^{\rightarrow}$ is computed for every sub-trace except the
681 last one (sub-trace iv, $\langle a, d \rangle$, in the example). The calculation of $N_{a,b}^{\rightarrow}$ is sim-
682 ilar to the one of $N_{a,b}^\#$ for entire traces (see Section 4.2.2), with one exception:
683 not all activities $b \in \Sigma$ are considered, but only those that occur in the trace
684 under analysis. For instance, on a trace like $\langle a, a, b, a, c, a \rangle$, $N_{a,b}^{\rightarrow}$ would not be
685 computed for $b \in \{d\}$. In the example of Table 6, $\langle a, a, b, a, c, a, d \rangle$, sub-trace
686 ii, $\langle a, b \rangle$, leads to the following values of $N_{a,b}^{\rightarrow}$: $N_{a,b}^{\rightarrow} = 0$, $N_{a,c}^{\rightarrow} = 1$, and $N_{a,d}^{\rightarrow} = 1$.
687 The rationale is, that for every pair of a 's in the trace, the b event which misses
688 in-between will eventually occur after at least two occurrences of a . Therefore,
689 a is repeated at least twice before b . In fact, the last sub-trace is not considered
690 in the computation of $N_{a,b}^{\rightarrow}$, because the missing b represents an event which does
691 not occur at all after a . However, this case is already covered by $N_{a,b}^\#$.

692 A new value for differential cumulative set-counter $\Delta N_{a,S}^{\rightarrow}$ is aggregated from
693 $N_{a,b}^{\rightarrow}$ for each $b \in S$ at every slicing point, i.e., before the next occurrence of a .
694 Thereafter, it is \boxplus -added to the preceding values. In the example, $\Delta N_{a,\{b,c,d\}}^{\rightarrow} = 1$
695 is calculated for subtrace i, $\langle a \rangle$. Then, from subtrace ii ($\langle a, b \rangle$), $\Delta N_{a,\{c,d\}}^{\rightarrow} = 1$ is

Algorithm 6: Procedure EVALFOLLOWINGREPSINBETWEEN(Σ, \vec{t}), evaluating missing co-occurrences of activities in a trace

Input: A log alphabet Σ and a trace $\vec{t} = \langle t_1, \dots, t_{|\vec{t}|} \rangle$
Output: The set of differential cumulative set-counters ΔN^{\rightarrow} derived from trace \vec{t}

```

1  foreach  $a \in \Sigma$  do
2  |  $?_a^{\rightarrow} \leftarrow \perp$  // A flag checking whether  $a$  already occurred in the trace
3  |  $\vec{s}_a \leftarrow \diamond$  // A sub-trace appending events after  $a$ 
4   $S^i \leftarrow \{\}$  // Stores pairs that index with activity  $a$  the activity sets before which  $a$ 
   recurrent
5  for  $i \leftarrow 1$  to  $|\vec{t}|$  do
6  |  $a \leftarrow t_i$ 
7  | if  $?_a^{\rightarrow} = \top$  then
   | /* Increment the pairwise counters for activities not in the subtrace */
8  | | foreach  $b \in \Sigma \setminus \{a\}$  do
9  | | | if  $b \notin \vec{s}_a$  then
10 | | | |  $N_{a,b}^{\rightarrow} \leftarrow N_{a,b}^{\rightarrow} + 1$ 
   | | | /* Derive differential cumulative set-counters */
11 | | | foreach  $\Delta N_{\vec{s}_a, S}^{\rightarrow} \in \text{PAIRWISE2DIFF} \left( \Sigma, \bigcup_{b \in \Sigma \setminus \{a\}} N_{a,b}^{\rightarrow} \right)$  do
12 | | | |  $S^i \leftarrow S^i \cup \langle a, S \rangle$ 
13 | | | |  $\Delta N_{i, S}^{\rightarrow} \leftarrow \Delta N_{i, S}^{\rightarrow} \boxplus \Delta N_{\vec{s}_a, S}^{\rightarrow}$ 
14 | | | | foreach  $b \in \Sigma \setminus \{a\}$  do  $N_{a,b}^{\rightarrow} \leftarrow 0$ ; // Reset the pairwise counters
15 | | | |  $\vec{s}_a \leftarrow \diamond$  // Reset the substring related to  $a$ 
16 | | else
17 | | |  $?_a^{\rightarrow} \leftarrow \top$ 
18 | | |  $\vec{s}_a \leftarrow \vec{s}_a \circ a$ 
19 | | | foreach  $b \in \Sigma \setminus \{a\}$  do
20 | | | | if  $?_b^{\rightarrow} = \top$  then
21 | | | | |  $\vec{s}_b \leftarrow \vec{s}_b \circ a$ 
22 return  $\bigcup_{\langle a, S \rangle \in S^i} \Delta N_{i, S}^{\rightarrow}$ 

```

696 computed. $\Delta N_{a, \{b, d\}}^{\rightarrow} = 1$ stems from sub-trace iii, i.e., $\langle a, c \rangle$. It follows that:

$$\mathfrak{q}_{\rightarrow L}(a, S) = \nu \sum_{T \supseteq S} \Delta N_{a, T}^{\rightarrow} \quad (8)$$

697

698 *4.2.6. Count of repeated occurrences of an activity before other events on re-*
699 *versed traces*

700 The calculation of $\leftarrow \mathfrak{p}_L(a, S)$ executes the operations described for $\mathfrak{q}_{\rightarrow L}(a, S)$,
701 reversing the order in which the trace is parsed. Thus, the pairwise counter
702 $N_{a,b}^{\leftarrow}$ is assigned with values in the same way in which $N_{a,b}^{\rightarrow}$ was computed, but
703 parsing, e.g., $\langle d, a, c, a, b, a, a \rangle$ in place of $\langle a, a, b, a, c, a, d \rangle$ (see Table 7). Traces
704 are divided into sub-traces at every occurrence of the activation. In the example,
705 the reversed trace $\langle d, a, c, a, b, a, a \rangle$ is thus sliced into: (i) $\langle d, a, c \rangle$, (ii) $\langle a, b \rangle$,
706 (iii) $\langle a \rangle$, and (iv) $\langle a \rangle$. The differential cumulative set-counter $\Delta N_{a, S}^{\leftarrow}$ is derived
707 from $N_{a,b}^{\leftarrow}$ exactly as $\Delta N_{a, S}^{\rightarrow}$ stems from $N_{a,b}^{\rightarrow}$. Each time a sub-trace is parsed,

Sliced trace	$N_{a,b}^{\uparrow+}$	$N_{a,c}^{\uparrow+}$	$N_{a,d}^{\uparrow+}$	$\Delta N_{a,\cdot}^{\uparrow+}$
$\langle a \rangle \leftarrow$	1	1	1	$\Delta N_{a,\{b,c,d\}}^{\uparrow+} = 1$
$\langle a, b \rangle \leftarrow$		1	1	$\Delta N_{a,\{c,d\}}^{\uparrow+} = 1$
$\langle a, c \rangle \leftarrow$	1		1	$\Delta N_{a,\{b,d\}}^{\uparrow+} = 1$
$\langle a, d \rangle$				

Table 6: Computation of $N_{a,\cdot}^{\uparrow+}$ and $\Delta N_{a,\cdot}^{\uparrow+}$, given a sample trace: $\langle a, a, b, a, c, a, d \rangle$. The \leftarrow symbol indicates the point in which the trace has been split (i.e., before the next occurrence of a).

Sliced trace	$N_{a,b}^{\leftarrow+}$	$N_{a,c}^{\leftarrow+}$	$N_{a,d}^{\leftarrow+}$	$\Delta N_{a,\cdot}^{\leftarrow+}$
$\langle d, a, c \rangle \leftarrow$	1		1	$\Delta N_{a,\{b,d\}}^{\leftarrow+} = 1$
$\langle a, b \rangle \leftarrow$		1	1	$\Delta N_{a,\{c,d\}}^{\leftarrow+} = 1$
$\langle a \rangle \leftarrow$	1	1	1	$\Delta N_{a,\{b,c,d\}}^{\leftarrow+} = 1$
$\langle a \rangle$				

Table 7: Computation of $N_{a,\cdot}^{\leftarrow+}$ and $\Delta N_{a,\cdot}^{\leftarrow+}$, given a sample trace, $\langle a, a, b, a, c, a, d \rangle$, which is reversed into $\langle d, a, c, a, b, a, a \rangle$. The \leftarrow symbol indicates the point in which the trace has been split (i.e., before the next occurrence of a).

708 $N_{a,b}^{\leftarrow+}$ is reset to 0 for every $a, b \in \Sigma$. After the next sub-trace has been parsed,
709 $\Delta N_{a,S}^{\leftarrow+}$ is \boxplus -added by the differential cumulative set-counter. The last sub-trace
710 in the reversed trace ($\langle a \rangle$, in the example) is not considered in the computation.
711 Therefore, we have that:

$$\leftarrow^{\rho} L(a, S) =_{\nu} \sum_{T \supseteq S} \Delta N_{a,T}^{\leftarrow+} \quad (9)$$

712

713 4.2.7. Count of events immediately following the occurrence of an activity

714 In order to compute the value of $\blacktriangleright^{\rho} L(a, b)$, the pairwise counter $N_{a,b}^{\blacktriangleright+}$ is
715 utilised. For each trace, $N_{a,b}^{\blacktriangleright+}$ stores the occurrences of b immediately following
716 a , as described in Algorithm 7 (procedure EVALFOLLOWING). In $\langle a, a, b, a, c, a \rangle$,
717 e.g., $N_{a,b}^{\blacktriangleright+} = 1$, $N_{a,c}^{\blacktriangleright+} = 1$ and $N_{a,d}^{\blacktriangleright+} = 0$. Traces in event logs are defined as *sequences*
718 of events. As such, two events cannot be contemporary. Therefore, only one
719 event can immediately follow the occurrence of an activity a . Owing to this,
720 our technique does not require the usage of differential cumulative set-counters
721 here:

$$\blacktriangleright^{\rho} L(a, b) =_{\nu} N_{a,b}^{\blacktriangleright+} \quad (10)$$

722 The same observation holds true for the computation of $N_{a,b}^{\blacktriangleleft+}$.

Algorithm 7: Procedure EVALMISSING(Σ, \vec{t}), evaluating missing co-occurrences of activities in a trace

Input: A log alphabet Σ and a trace $\vec{t} = \langle t_1, \dots, t_{|\vec{t}|} \rangle$
Output: All values of \sharp , interpreted over Σ and \vec{t}

```

1 for  $i \leftarrow 1$  to  $|\vec{t}|$  do
2   if  $i > 1$  then  $a \leftarrow b$ 
3    $b \leftarrow t_i$ 
4   if  $i > 1$  then  $N_{a,b}^{\rightarrow} \leftarrow N_{a,b}^{\rightarrow} + 1$ 
5  $N_{\forall V}^{\rightarrow} \leftarrow \bigcup_{a \in \Sigma, b \in \Sigma \setminus \{a\}} N_{a,b}^{\rightarrow}$ 
6 return  $N_{\forall V}^{\rightarrow}$ 

```

Target-Branched Declare constraint	Support
$RespondedExistence(a, S)$	$1 - \frac{\sharp_L(a, S)}{\#_L(a)}$
$Response(a, S)$	$1 - \frac{\sharp_L(a, S)}{\#_L(a)}$
$AlternateResponse(a, S)$	$1 - \frac{\sharp_L(a, S) + \wp_L(a, S)}{\#_L(a)}$
$ChainResponse(a, S)$	$\frac{\sum_{b \in S} \blacktriangleright_L(a, S)}{\#_L(a)}$
$Precedence(S, a)$	$1 - \frac{\sharp_L(a, S)}{\#_L(a)}$
$AlternatePrecedence(S, a)$	$1 - \frac{\sharp_L(a, S) + \blacktriangleleft_L(a, S)}{\#_L(a)}$
$ChainPrecedence(S, a)$	$\frac{\sum_{b \in S} \blacktriangleleft_L(a, S)}{\#_L(a)}$

Table 8: Target-Branched Declare constraints and support functions.

723 *4.2.8. Count of events immediately preceding the occurrence of an activity*

724 The computation of $\blacktriangleleft_L(a, b)$ takes advantage of pairwise counter $N_{a,b}^{\leftarrow}$. For
725 each trace, $N_{a,b}^{\leftarrow}$ stores the occurrences of b immediately preceding a . Instruc-
726 tions of EVALPRECEDING are the same as EVALFOLLOWING (Algorithm 7) but
727 applied to a reversed trace. In $\langle a, a, b, a, c, a, d \rangle$, e.g., $N_{a,b}^{\leftarrow} = 1$, $N_{a,c}^{\leftarrow} = 1$ and
728 $N_{a,d}^{\leftarrow} = 0$. To determine these values, the same technique adopted for $N_{a,b}^{\rightarrow}$ can be
729 utilised after reversing the trace. In the sample trace, $\langle d, a, c, a, b, a, a \rangle$ would be
730 parsed in place of $\langle a, a, b, a, c, a, d \rangle$:

$$\blacktriangleleft_L(a, b) =_{\nu} N_{a,b}^{\leftarrow} \quad (11)$$

731 *4.3. Querying the Knowledge Base*

732 Once the knowledge base is built, the support of constraints can be cal-
733 culated. Table 8 lists the functions adopted to this end for each TBDeclare
734 constraint. All queries build upon a Laplacian concept of probability with sup-
735 port being computed as the number of supporting cases divided by the total

736 number of cases. In particular, the total number of cases is the count of oc-
737 currences of the activation $a \in \Sigma$ in the log $\#_L(a)$. For $ChainResponse(a, S)$,
738 supporting cases are those occurrences of a immediately followed by some $b \in S$,
739 i.e., $\blacktriangleright_L(a, b)$. Supporting cases can be summed up because if a is followed
740 by a given $b \in S$ in a trace, it cannot be immediately followed by any other
741 event $c \in S$. In other words, the two cases are mutually exclusive. However,
742 this assumption does not hold true, e.g., for $Response(a, S)$. Therefore, in this
743 case, we consider the non-supporting cases, when a is *not* followed by any of
744 the $b \in S$, i.e., $\nabla_L(a, S)$. We get that $P(E) = 1 - P(\bar{E})$ with $P(E)$ being the
745 probability of E and \bar{E} its negation. Hence, the support of $Response(a, S)$ is
746 $1 - \frac{\nabla_L(a, S)}{\#_L(a)}$. Likewise, the support of $RespondedExistence(a, S)$ is computed
747 on the basis of the non-supporting cases. The support of $AlternateResponse(a, S)$
748 is based on the cases when either (i) a is not followed by any $b \in S$ ($\nabla_L(a, S)$),
749 or (ii) a occurs more than once before the first occurrence of $b \in S$ ($\blacktriangleright_L(a, S)$).
750 The two conditions are mutually exclusive. Therefore, it is appropriate to sum
751 them up. Analogous considerations lead to the definition of support functions
752 for $Precedence(S, a)$, $AlternatePrecedence(S, a)$ and $ChainPrecedence(S, a)$.

753 4.4. Pruning the Returned Constraints

754 The power-set of activities in the log alphabet amounts to $2^{|\Sigma|-1}$. There-
755 fore, if we name the number of TBDeclare templates as N , up to $N \times 2^{|\Sigma|-1}$
756 constraints can potentially hold true. When a maximum limit of the branching
757 factor β to the cardinality of the set is imposed, this number is reduced to

$$|\Sigma| \times N \times \sum_{i=1}^{\min\{\beta, |\Sigma|-1\}} \binom{|\Sigma|-1}{i}$$

758 However, even with branching factor set to 3 and $|\Sigma| = 10$, already 3,087 con-
759 straints have to be evaluated. A model including such a number of constraints
760 would be hardly comprehensible for humans [26, 27]. In order to reduce this
761 number, we adopt pruning based on set-dominance and on hierarchy subsump-
762 tion.

763 4.4.1. Pruning Based on Set-Dominance.

764 The idea of this pruning approach is that if, e.g., $Response(a, \{b, c\})$ and
765 $Response(a, \{b, c, d\})$ have the same support, the first is more informative than
766 the second. Indeed, stating that “if a is executed then either b or c would
767 eventually follow”, entails that also “either b, c or d would eventually follow”.
768 In general terms, the support of TBDeclare constraints that are instantiations
769 of the same template and share the activation increases according to the set-
770 containment relation of target activities (see Corollary 1). To this end, the
771 mining algorithm distributes the discovered constraints, along with their com-
772 puted support, on a structure like the Hasse Diagram of Figure 2. This is a
773 Direct-acyclic graph, such that a breadth-first search can be implemented. For
774 each constraint, the pruning technique visits the nodes, from the biggest in size

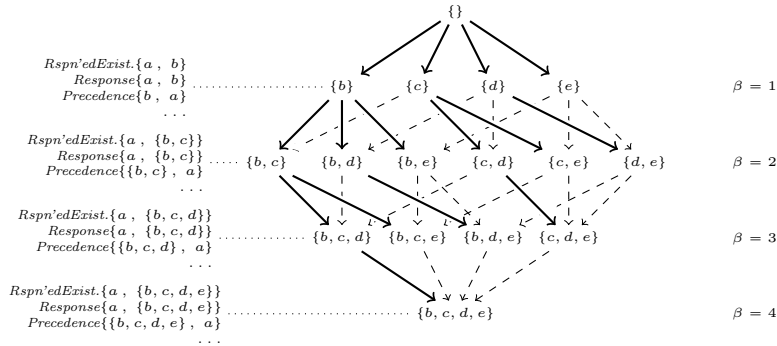


Figure 2: A Hasse Diagram representing the Partial Order set containment relation. Containing sets are at the head of connecting arcs, contained sets are at the tail.

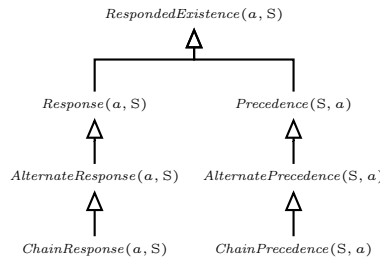


Figure 3: Diagram showing the subsumption hierarchy relation. Constraints that are subsumed are at the tail.

775 to the smallest. For instance, it can start from $Response(a, \{b, c, d, e\})$, i.e., the
 776 sink node, if the branching factor is equal to the size of the log alphabet. Given
 777 the current node, it checks whether in one of the parent nodes a constraint
 778 is stored (i.e., $Response(a, \{b, c, d\})$, $Response(a, \{b, c, e\})$, $Response(a, \{b, d, e\})$,
 779 $Response(a, \{c, d, e\})$) with greater or equal support. If so, it marks the current
 780 as redundant, and proceeds the visit towards the parent nodes that are not
 781 already marked as redundant. Otherwise, it marks all the ancestors as redun-
 782 dant. The parsing ends when either (i) the visit reaches the root node or (ii) no
 783 parent, which is not already marked as redundant, is available for the visit.

784 4.4.2. Pruning Based on Hierarchy Subsumption.

785 As investigated in [4, 22, 28], Declare constraints are not independent,
 786 but partially form a subsumption hierarchy. We consider a constraint $C(a, S)$
 787 subsumed by another constraint $C'(a, S)$ when all the traces that comply
 788 with $C(a, S)$ also comply with $C'(a, S)$. $Response(a, S)$, e.g., is subsumed
 789 by $RespondedExistence(a, S)$. Figure 3 depicts the subsumption hierarchy
 790 for TBDeclare constraints. It follows that a subsumed constraint always

791 has a support which is less than or equal to the subsuming one. This
792 pruning technique aims at keeping those constraints that are the most re-
793 strictive, among the most supported. Therefore, it labels as redundant
794 every constraint C which is at the same time (i) subsumed by another
795 constraint C' , and (ii) having a lower support than C' . Therefore, if,
796 e.g, given a log L defined over a log alphabet Σ s.t. $a \in \Sigma$ and $S \subseteq \Sigma$,
797 $\mathcal{S}_L(\text{RespondedExistence}(a, S)) > \mathcal{S}_L(\text{Response}(a, S))$, then $\text{Response}(a, S)$ is
798 marked as redundant. However, if $\mathcal{S}_L(\text{RespondedExistence}(a, S)) =$
799 $\mathcal{S}_L(\text{Response}(a, S))$, then $\text{Response}(a, S)$ is preferred. This is due to the fact
800 that more restrictive constraints hold more information than the less restrict-
801 ive ones. The pruning approach is based on the monotone non-decrement of
802 support (cf. Figure 3). It operates as follows. Starting from the root of the
803 hierarchy tree, if a constraint has a support equal to one of the children, it is
804 marked as redundant and the visit proceeds with the children. If a child has
805 a support which is lower than the parent, it is marked as redundant. All its
806 children will be automatically marked as redundant as well, as they cannot have
807 a higher support.

808 Both pruning techniques complement one another in reducing the set of the
809 discovered constraints.

810 5. Experiments and Evaluation

811 In this section, we investigate the efficiency and effectiveness of our ap-
812 proach. In particular, we compare the performances of the new proposed al-
813 gorithms w.r.t. the ones described in [8]. Section 5.1 shows the results obtained
814 by applying the proposed technique to synthetic logs. Section 5.2 validates our
815 approach by using event logs from a process to solve disruptions of ICT-services
816 in the Rabobank Netherlands Group ICT and from a loan application process
817 of a Dutch financial institute. All experiments were run on a server machine
818 equipped with Intel Xeon CPU E5-2650 v2 2.60GHz, using 1 64-bit CPU core
819 and 16GB main memory quota.

820 5.1. Evaluation Based on Simulation

821 To test the effectiveness and the efficiency of our approach, we have defined
822 a simple Declare model including the following constraints:

- | | | | |
|-----|--|-----|--|
| 823 | • $\text{ChainPrecedence}(\{a, b\}, c)$ | 826 | • $\text{RespondedExistence}(a, \{b, c, d, e\})$ |
| 824 | • $\text{ChainPrecedence}(\{a, b, d\}, c)$ | 827 | • $\text{Response}(a, \{b, c\})$ |
| 825 | • $\text{AlternateResponse}(a, \{b, c\})$ | 828 | • $\text{Precedence}(\{a, b, c, d\}, e)$ |

829 and we have simulated it to generate a compliant event log as described in [4].
830 In our experiments, we focus on different characteristics of the discovery task
831 including average length of the traces, number of traces, and number of activ-
832 ities. Moreover, we consider characteristics of the discovered model including

833 minimum support and maximum branching factor. In our experiments, we have
 834 run the algorithm varying the value of one variable at a time. The remaining
 835 variables were fixed and corresponding to 4 and 25 for minimum and maximum
 836 trace length respectively, 10,000 for log size, 8 for log alphabet size, 1.0 for
 837 support threshold, and 3 for branching factor. Each configuration has been
 838 averaged over 10 randomly generated logs.

Branch.F	Supp.T	Equal	Restr.	None	Branch.F	Supp.T	Equal	Restr.	None
1	0.85	0	1	13	5	0.85	2	1	85.6
	0.9	0	1	12.6		0.9	2	1	86.9
	0.95	0	1	9.1		0.95	2	1	81.7
	1	0	0	0		1	2	1	17
2	0.85	2	4.1	95.9	6	0.85	2	1	28.3
	0.9	2	3.4	73.9		0.9	2	1	25.8
	0.95	2	2	69.3		0.95	2	1	22.9
	1	2	0	0		1	2	1	15.8
3	0.85	2	3	232.2	7	0.85	2	1	23.2
	0.9	2	3	209.1		0.9	2	1	19.4
	0.95	2	2.8	159.7		0.95	2	1	18.8
	1	2	1	2.4		1	2	1	16.8
4	0.85	2	1	203.7	8	0.85	2	1	24.5
	0.9	2	1	202.2		0.9	2	1	21.1
	0.95	2	1	186.9		0.95	2	1	18.5
	1	2	1	10		1	2	1	15.1

Table 9: Summary of matching constraints in the mined process.

839 *Effectiveness.* First, we demonstrate the effectiveness of our approach by investigat-
 840 ing the reduction effect of the proposed pruning techniques. In particular,
 841 we analyse the trend of the variable “number of discovered constraints” as a
 842 function of log alphabet size, branching factor, and support threshold, in logarithmic
 843 scale.

844 Figure 4a shows the trend of the number of discovered constraints by varying
 845 the log alphabet size. Different curves refer to different configurations of the
 846 miner: without any pruning (diamonds); with set-containment-based pruning
 847 (crosses); with set-containment- and hierarchy-based pruning (asterisks); with
 848 set-containment- and hierarchy-based pruning and support threshold (points);
 849 with support threshold only (triangles). This plot provides evidence that as the
 850 number of activities in the log alphabet increases, the number of discovered con-

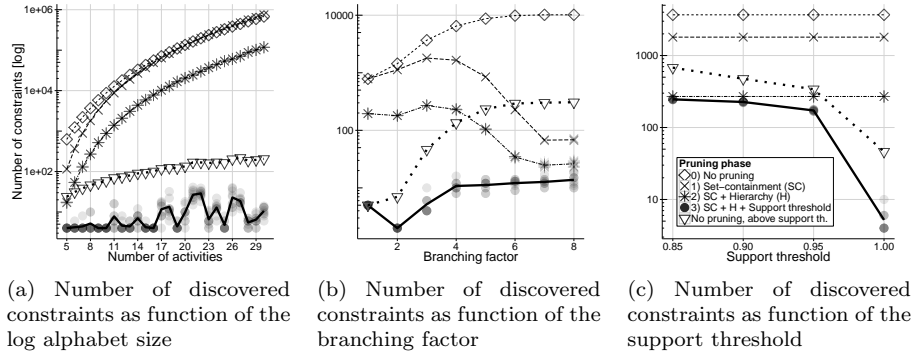


Figure 4: Effectiveness tests performed on synthetic logs.

851 straints increases as well. However, we discover a lower increase of constraints
 852 as we proceed further in the sequence of pruning techniques. Moreover, there
 853 is a significant difference between the number of discovered constraints with filtering
 854 based on the minimum support threshold only, and based on the pruning
 855 techniques presented in this paper. This improvement yields a reduction ratio
 856 of 94.84% (205.6 versus 10.6, on average), for a log alphabet size of 30.

857 Figure 4b shows the trend of the number of discovered constraints by varying
 858 the branching factor. Without pruning, or with the simple filtering by minimum
 859 support threshold, the number of discovered constraints increases as the number
 860 of branches increases. On the other hand, when we apply the set-dominance
 861 and hierarchy-based pruning techniques, the number of discovered constraints
 862 is approximately constant up to a branching value of 3. After this value, the
 863 number of constraints decreases. When we apply all the proposed pruning techniques
 864 together, the number of constraints eventually increases. In addition,
 865 the number of constraints obtained by applying set-dominance and subsumption
 866 hierarchy converges to the number of constraints discovered when all the
 867 pruning techniques are applied together. The difference between the number
 868 of discovered constraints with support threshold and the number of discovered
 869 constraints after using the pruning techniques presented in this paper is quantified
 870 (branching factor of 8) in a reduction ratio of 95.51% (307.7 versus 13.8,
 871 on average).

872 The plot in Figure 4c confirms that for any threshold between 0.85 and 1.0,
 873 the number of constraints discovered by applying all the pruning techniques is
 874 lower than the one obtained by applying the support threshold filtering only.
 875 The reduction ratio is indeed 88.74% (46.2 versus 5.2, on average) when the
 876 threshold is set to 1.0.

877 An additional experiment to test the effectiveness of our approach is illustrated
 878 in Table 9. Here, for different values of branching factor (ranging from
 879 1 to 8) and support threshold (ranging from 0.85 to 1), we evaluate the capability
 880 of the discovery algorithm to rediscover the model that was used for log

881 generation. In particular, for each combination of branching factor and support
882 threshold, we have generated 10 random logs starting from the model described
883 at the beginning of this section. Then, we have considered the average number
884 of constraints correctly discovered (column *Equal* in the table) and the average
885 number of discovered constraints that strengthen one of the constraints
886 of the original model (column *Restr.* in the table).³ In column *None* in the
887 table, we show the average number of additional constraints discovered. These
888 constraints are characteristic of each specific (random) log but still compliant
889 with the original model. Note that the branching factor affects the number of
890 constraints correctly discovered since, for example, if we specify a maximum
891 branching factor equal to 2, it will be impossible to discover a constraint with
892 3 branches.

893 The constraints correctly discovered with branching factor equal to
894 2 and support threshold equal to 1 are *ChainPrecedence*($\{a,b\},c$) and
895 *AlternateResponse*($a,\{b,c\}$). This model contains the only constraints that
896 can be correctly discovered using a branching factor of 2. Indeed, the
897 third constraint with 2 branches in the original model is *Response*($a,\{b,c\}$),
898 which is entailed by *AlternateResponse*($a,\{b,c\}$). The constraints correctly dis-
899 covered with branching factor equal to 3 and support equal to 1 are, again,
900 *ChainPrecedence*($\{a,b\},c$) and *AlternateResponse*($a,\{b,c\}$). However, in this
901 case, also *Precedence*($\{a,b,d\},e$), restriction of *Precedence*($\{a,b,c,d\},e$), is dis-
902 covered. This result improves the original models that contains a redundancy.
903 Indeed, in all cases in which *e* is preceded by *c*, it is also preceded by *a* or by
904 *b* due to *ChainPrecedence*($\{a,b\},c$). Starting from a branching factor of 3 up to
905 a branching factor of 8, these 3 constraints are always part of the discovered
906 models. This confirms the effectiveness of the proposed approach since this set
907 of constraints corresponds to the original set after removing redundancies.

908 *Efficiency.* Figure 5 shows the efficiency of our approach by plotting the compu-
909 tation time as a function of log alphabet size, branching factor, log size, and
910 average trace size. Figure 5a shows the trend of the computation time (in log-
911 arithmic scale) by varying the log alphabet size. Different curves refer to the
912 computation time for (i) the knowledge base construction, (ii) the querying on
913 the knowledge base, and (iii) to the total computation time. Notice that there
914 is a break point, when the log alphabet is composed of 12 activities: there the
915 query time becomes higher than the knowledge base construction time. In Fig-
916 ure 5b, we can see that the computation time (here displayed in logarithmic
917 scale) does not depend on the branching factor. It is approximately constant
918 and higher for querying the knowledge base. Figure 5c shows the trend of the
919 computation time by varying the log size, whereas Figure 5d depicts the trend
920 of the computation time by varying the average trace size (both displayed in
921 linear scale). In both cases, the query time clearly outperforms the knowledge

³Note that there is also the possibility, in some cases, that the discovered model contains constraints that are entailed by one of the constraints of the original model. However, this happens very rarely using randomly generated logs and it never occurred in our experiments.

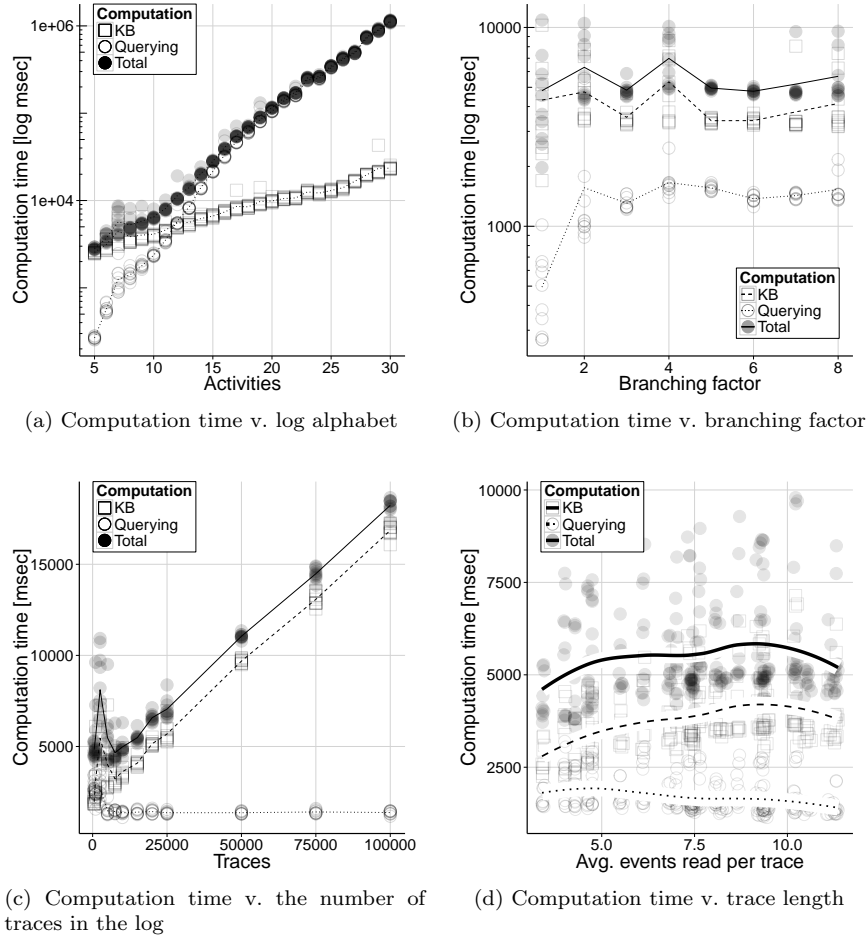
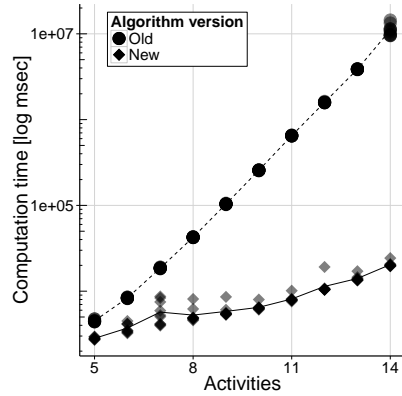


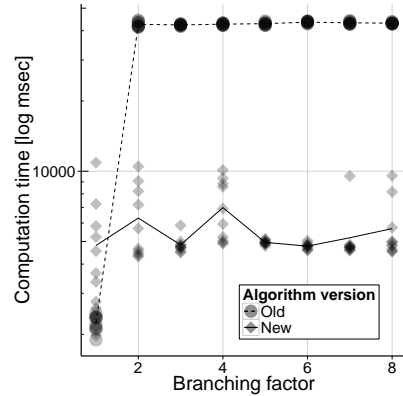
Figure 5: Efficiency tests performed on synthetic logs, comparing the computation time needed for building the knowledge base (“KB” time), and for deriving the constraints (“querying” time).

922 base construction time. Generally speaking, the only factor that makes queries
 923 less efficient than the knowledge base construction is the size of the alphabet.

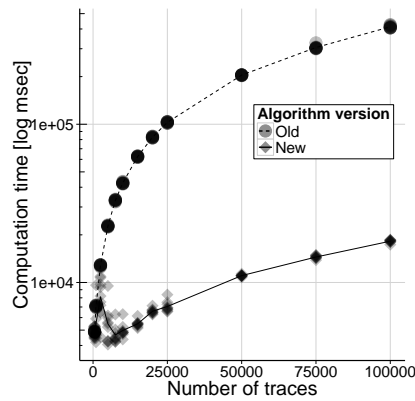
924 In Figure 6, we compare the time performances of the new version of the
 925 discovery algorithm, w.r.t. the version presented in [8]. In particular, we plot
 926 the computation time as a function of log alphabet size, branching factor, log
 927 size, and average trace size, in logarithmic scale. For all these parameters,
 928 the plots highlight the dramatic reduction of the computation time when using
 929 the new proposed approach. The main factor that contributed to the perfor-
 930 mance improvement is the new algorithm adopted for the computation of



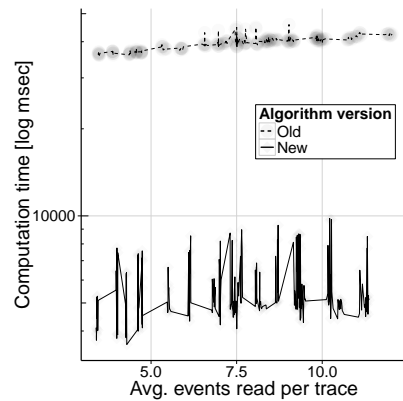
(a) Computation time v. log alphabet



(b) Computation time v. branching factor



(c) Computation time v. the number of traces in the log



(d) Computation time v. trace length

Figure 6: Efficiency tests performed on synthetic logs, comparing the time performances of the new version of the discovery algorithm versus the version of [8].

931 *AlternateResponse* and *AlternatePrecedence* constraints.

932 5.2. Evaluation Based on Real Data

933 In this section, we validate our approach using real-life logs. The results are
 934 described in the following sections. The first log we use has been provided by a
 935 Dutch financial institute. The second log has been provided by the Rabobank
 936 Netherlands Group ICT.

937 *5.2.1. A Dutch Financial Institution*

938 We have evaluated the applicability of our approach using a real-life event
939 log provided for the BPI challenge 2012 [29]. The event log pertains to an ap-
940 plication process for personal loans or overdrafts of a Dutch financial institution.
941 It contains 262 200 events distributed across 24 different possible activities and
942 includes 13 087 cases.

943 In this case, it is possible to prune the list of discovered constraints in
944 order to obtain a compact set of constraint, which is understandable for
945 human analysts. By applying the miner with a support threshold equal to 1,
946 confidence threshold set to 0.8, and branching factor 5, we obtain the following
947 11 constraints:

948 *ChainResponse*(*A.SUBMITTED*, *A.PARTLYSUBMITTED*)
949 *AlternateResponse*(*A.SUBMITTED*, {*A.PREACCEPTED*,*A.DECLINED*,*A.CANCELLED*})
950 *AlternateResponse*(*A.SUBMITTED*, {*A.PREACCEPTED*,*A.DECLINED*,*W.Afhandelen leads*})
951 *AlternateResponse*(*A.SUBMITTED*, {*W.Completeren aanvraag*,*A.DECLINED*,*A.CANCELLED*})
952 *AlternateResponse*(*A.SUBMITTED*, {*W.Completeren aanvraag*,*A.DECLINED*,*W.Afhandelen leads*})
953 *ChainPrecedence*(*A.SUBMITTED*, *A.PARTLYSUBMITTED*)
954 *AlternateResponse*(*A.PARTLYSUBMITTED*, {*A.PREACCEPTED*,*A.DECLINED*,*A.CANCELLED*})
955 *AlternateResponse*(*A.PARTLYSUBMITTED*, {*A.PREACCEPTED*,*A.DECLINED*,*W.Afhandelen leads*})
956 *ChainResponse*(*A.PARTLYSUBMITTED*, {*A.PREACCEPTED*,*A.DECLINED*,*W.Afhandelen leads*,*W.Beoordelen fraude*})
957 *AlternateResponse*(*A.PARTLYSUBMITTED*, {*W.Completeren aanvraag*,*A.DECLINED*,*A.CANCELLED*})
958 *AlternateResponse*(*A.PARTLYSUBMITTED*, {*W.Completeren aanvraag*,*A.DECLINED*,*W.Afhandelen leads*})
959

960 This results are in line with what described in the report published by the
961 winners of the BPI challenge 2012 [30]. For example, one of the results discussed
962 in this report is that each case starts with an application of a customer where
963 an application is first submitted and, immediately after, partly submitted. In
964 addition, over 13 087 cases in the log, in 4 852 cases, an application partly sub-
965 mitted is immediately pre-accepted, in 3 429 cases it is immediately declined
966 and in the remaining cases is followed up (through activities corresponding to
967 events *A.Afhandelenleads* or *A.Beoordelenfraude*). This is in line with the
968 *ChainResponse* constraints discovered.

969 *5.2.2. Rabobank*

970 The case study we illustrate in this section has been provided for the BPI
971 challenge 2014 by the Rabobank Netherlands Group ICT [9]. The log we use
972 pertains to the management of calls or mails from customers to the Service
973 Desk concerning disruptions of ICT-services. The log contains 46 616 cases,
974 466 737 events referring to 39 different activities. There are 242 originators
975 and domain specific event attributes like *KM number*, *Interaction ID* and
976 *IncidentActivity_Number*.

977 By applying the miner with a support threshold equal to 1, confidence
978 threshold set to 0.8, and branching factor 5, we obtain the following 18
979 constraints:

980 *Precedence*({*Reassignment*, *Operator Update*, *Update from customer*, *Open*},*Assignment*)

981 *RespondedExistence*(*Assignment*,{*Reassignment*, *Closed*, *Pending vendor*})
982 *RespondedExistence*(*Assignment*,{*Reassignment*, *Open*})
983 *RespondedExistence*(*Assignment*,{*Operator Update*, *Update from customer*, *Open*})
984 *RespondedExistence*(*Assignment*,{*Update from customer*, *Caused By CI*, *Open*})
985 *RespondedExistence*(*Assignment*,{*Update from customer*, *Description Update*, *Open*})
986 *RespondedExistence*(*Assignment*,{*Update from customer*, *Update*, *Open*})
987 *RespondedExistence*(*Assignment*,{*Update from customer*, *OO Response*, *Open*})
988 *RespondedExistence*(*Assignment*,{*Closed*, *Status Change*})
989 *RespondedExistence*(*Assignment*,{*Closed*, *External Vendor Assignment*})
990 *RespondedExistence*(*Assignment*,{*Closed*, *Pending vendor*, *Vendor Reference*})
991 *RespondedExistence*(*Assignment*,{*Closed*, *Open*})
992 *RespondedExistence*(*Assignment*,{*Caused By CI*, *Resolved*, *Open*})
993 *Response*(*Open*,{*Reassignment*, *Closed*, *Pending vendor*})
994 *Response*(*Open*,{*Assignment*, *Closed*, *Pending vendor*})
995 *Response*(*Open*,{*Closed*, *Status Change*})
996 *Response*(*Open*,{*Closed*, *External Vendor Assignment*})
997 *Response*(*Open*,{*Closed*, *Pending vendor*, *Vendor Reference*})
998

999 From further analysis of the log (see also <http://www.win.tue.nl/bpi/2014/challenge>), it is possible to verify that these results reflect the reality.
1000 For example, over 46 616 cases in the log, only in 449 cases an opened incident
1001 is not eventually closed. These 449 cases always contain a status change and an
1002 external vendor assignment. Only 447 of them contain a pending vendor and
1003 the remaining 2 as well as a status change and an external vendor assignment
1004 both contain an assignment, a reassignment and a vendor reference. This is in
1005 line with the list of *Response* constraints discovered.
1006

1007 6. Related Work

1008 *Process Mining* [31] is the set of techniques for the extraction of process de-
1009 scriptions, stemming from a set of recorded real executions (*event logs*). ProM
1010 [32] is one of the most used plug-in based software environments for implement-
1011 ing process mining techniques. Process Mining mainly covers three different
1012 aspects: process discovery, conformance checking and operational support. The
1013 first aims at discovering the process model from logs. Control-flow mining in
1014 particular focuses on the causal and sequential relations among activities. The
1015 second focuses on the assessment of the compliance of a given process model
1016 with event logs, and the possible enhancement of the process model in this re-
1017 gard. The third is finally meant to assist the enactment of processes at run-time,
1018 based on given process models.

1019 From [33] onwards, many techniques have been proposed for the control-flow
1020 mining: pure algorithmic (e.g., α algorithm, drawn in [34] and its evolution α^{++}
1021 [35]), heuristic (e.g., [36]), genetic (e.g., [37]), etc. A very smart extension to the
1022 previous research work was achieved by the two-steps algorithm proposed in [38].
1023 Differently from the former approaches, which typically provide a single process
1024 mining step, it splits the computation in two phases: (*i*) the configurable mining

1025 of a Transition System (TS) representing the process behavior and (ii) the
1026 automated construction of a Petri net bisimilar to the TS [39, 40]. In the
1027 field of conformance checking, [41, 42, 43] have proposed techniques capable of
1028 realigning procedural process models to logs.

1029 The need for flexibility in the definition of some types of process, such as
1030 the knowledge-intensive processes [44], has led to an alternative to the clas-
1031 sical “procedural” approach: the “declarative” approach. Rather than using a
1032 procedural language for expressing the allowed sequences of activities (“closed”
1033 models), it is based on the description of workflows through the usage of con-
1034 straints: the idea is that every task can be performed, except what does not
1035 respect such constraints (“open” models). The work of van der Aalst et al.
1036 [27] showed how the declarative approach (such as the one adopted by Declare
1037 [45]) could help in obtaining a fair trade-off between flexibility in managing
1038 collaborative processes and support in controlling and assisting the enactment
1039 of workflows. The original semantics of Declare used in these works is based
1040 on LTL_f . Other semantics for Declare have been proposed in [46] (based on
1041 the Event Calculus) and in [47, 48, 49] (based on Dynamic Condition Response
1042 Graphs). Very recent investigations have compared the procedural and the de-
1043 clarative paradigms and discussed the possibility of adopting hybrid approaches
1044 based on both procedural and declarative models [50, 51, 11, 52, 53, 54, 55].

1045 Our work contributes to the area of declarative process mining. In this
1046 context, Maggi et al. [5] first proposed an unsupervised algorithm for mining
1047 Declare processes. They based the discovery of constraints on the replay of the
1048 log on specific automata, each accepting only those traces that are compliant
1049 to one constraint. Candidate constraints are generated considering all the in-
1050 stantiations of Declare templates with activities that occur in the log. Each
1051 constraint among the candidates becomes part of the discovered process only
1052 if the percentage of traces accepted by the related automaton exceeds a user-
1053 defined threshold. In order to remove irrelevant constraints from the output set,
1054 the authors apply vacuity detection techniques [56]. Constraints are considered
1055 as vacuously satisfied when no trace in the log violates them, yet no trace shows
1056 the effect of their application either. A vacuously satisfied constraint is, e.g.,
1057 that every request is eventually acknowledged, in a process instance that does
1058 not contain requests.

1059 [6] describes an evolution of [5], with the adoption of a two-phase approach.
1060 The first phase is based on the Apriori algorithm, developed by Agrawal and
1061 Srikant for mining association rules [20]. During this preliminary phase, the fre-
1062 quent sets of correlated activities are identified. The candidate constraints are
1063 computed on the basis of the correlated activity sets only. During the second
1064 phase, the candidate constraints are checked as in [5]. Therefore, the search
1065 space for the second phase is reduced. In output, constraints constituting the
1066 discovered process are weighted according to their support, i.e., the probability
1067 of such constraints to hold in the mined process. To filter out irrelevant con-
1068 straints, more metrics are introduced, such as confidence and interest factor.
1069 Both the concepts of support and confidence have been adopted in this paper.

1070 In [28], Maggi et al. refined the technique of [6] by pruning returned con-

1071 straints on the basis of three main methods: (i) the removal of weaker con-
1072 straints entailed by stronger constraints; (ii) the reparation of predefined basic
1073 Declare models; (iii) an ontology-guided search for constraints, linking activities
1074 that either belong to different groups of interest, or to the same group. The last
1075 two require the user input, whereas the first does not. A technique for pruning
1076 an existing Declare model based on event correlations has been presented in
1077 [57].

1078 All the aforementioned “automata based” methods have been implemented
1079 in [58]. Unfortunately, none of these methods turned out to be practical in our
1080 context. This is mainly due to their checking algorithm, based on the replay
1081 of the log on one automaton for each candidate constraint. In TBDeclare, the
1082 search space of candidate constraints would be much too vast to make this
1083 approach feasible.

1084 [59, 60, 61] describe the usage of inductive logic programming techniques to
1085 mine models expressed as a SCIFF [62] first-order logic theory, consisting of a
1086 set of implication rules named Social Integrity Constraints (IC’s for short). To
1087 complete the Declare discovery, the learned theory is automatically translated
1088 into Declare notation. [63, 64] extend this technique by weighting in a second
1089 phase the constraints with a probabilistic estimation. The learned IC’s are
1090 indeed translated from SCIFF, discovered by DPML, into Markov Logic formu-
1091 lae [65]. Their probabilistic-based weighting is computed by the Alchemy tool
1092 [64]. Both the techniques in [59] and [64] rely on the availability of compliant
1093 and non-compliant traces of execution, w.r.t. the process to mine. As in the
1094 aforementioned “logic-based” approaches, we preferred to elaborate a technique
1095 which avoided the replay of every trace on automata in the log. On the other
1096 hand, we had to deal with traces which were not labeled in advance. Therefore,
1097 our technique does not require the user’s specification of positive and negative
1098 past executions.

1099 The third branch of Declare mining algorithms, alternative to the automata-
1100 and logic-based, is the one that started with [3]. It is based on a two-step
1101 approach. The first step computes statistic data describing the occurrences of
1102 activities and their interplay in the log. The second one checks the validity
1103 of Declare constraints by querying such a statistic data structure (knowledge
1104 base). [4] extends such an approach by weighing each constraint with reliability
1105 and interest metrics, such as support and confidence. [21] shows the boost in
1106 performance that such algorithm allowed, w.r.t. the automata-based approaches
1107 and [66] reports on its application in the context of highly flexible processes
1108 [44]. Although fast, these algorithms do not broaden the spectrum of returned
1109 constraints to TBDeclare. Therefore, we extended these works with a wider
1110 range of constraints and an efficient implementation algorithm.

1111 Recently, [67] have proposed a framework for discovering general LTL_f rules
1112 in event logs. Though more flexible than other approaches, it reveals not suit-
1113 able for TBDeclare, due to a deep increase of computation time, as soon as
1114 disjunction among variables are introduced. An efficient approach for the dis-
1115 covery of Declare models at runtime has been presented in [68]. However, this
1116 technique only allows for the discovery of standard Declare constraints.

1117 Various conceptual extensions of Declare have been proposed in the literat-
1118 ure, partially with accompanying mining algorithms. In [69], the authors define
1119 *Timed Declare*, an extension of Declare based on a metric temporal logic se-
1120 mantics allowing for the specification of required delays and deadlines. The ap-
1121 proach relies on timed automata to monitor metric dynamic constraints. In [70],
1122 such semantics is used for the discovery of metric temporal Declare constraints.
1123 [71] presents an approach for the discovery of Declare rules characterizing the
1124 lifecycle of non-atomic activities in a log. In [72], the authors propose an ap-
1125 proach for monitoring data-aware Declare constraints at run-time, based on the
1126 data-aware semantics for Declare presented in [46, 73]. In the work proposed
1127 in [7], an alternative data-aware semantics for Declare has been introduced by
1128 using a first-order variant of LTL to specify data-aware patterns. Such ex-
1129 tended patterns are used in [7] as the target language for a process discovery
1130 algorithm, which produces data-aware Declare constraints from raw event logs.
1131 The data-aware semantics for Declare has been further extended in [74]. In
1132 [75], a semantics for Declare based on metric first order temporal logics allows
1133 for combining data and temporal perspectives. Our work is complementary to
1134 these works. It is an avenue of future research to integrate TBDeclare with
1135 these perspectives.

1136 7. Conclusion

1137 In this paper, we have defined the class of Target-Branched Declare, which
1138 exhibits interesting properties in terms of set-dominance. We exploit these
1139 properties for the definition of an efficient mining approach. Furthermore, we
1140 specify pruning rules in order to arrive at a compact rule set. Our technique is
1141 evaluated for efficiency and effectiveness using simulated data and the case of the
1142 BPI Challenges of 2012 and 2014. In future research, we aim to further study
1143 broader classes of branched Declare. At this stage, we have focused on target-
1144 branched constraints. It is an open question how our results can be translated
1145 to the class of activation-branched constraints. Furthermore, we also plan to
1146 extend our technique towards the coverage of the entire Declare language.

1147 Acknowledgements

1148 The research work of Claudio Di Ciccio and Jan Mendling has received
1149 funding from the European Union’s Seventh Framework Programme (FP7/2007-
1150 2013) under grant agreement 318275 (GET Service).

1151 References

- 1152 [1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, *Fundamentals of Busi-*
1153 *ness Process Management*, Springer, 2013.

- 1154 [2] M. Pesic, Constraint-based workflow management systems: Shifting control
1155 to users, Ph.D. thesis, Technische Universiteit Eindhoven (10 2008).
1156 URL <http://repository.tue.nl/638413>
- 1157 [3] C. Di Ciccio, M. Mecella, Mining constraints for artful processes, in:
1158 W. Abramowicz, D. Kriksciuniene, V. Sakalauskas (Eds.), International
1159 Conference on Business Information Systems, Vol. 117 of Lecture Notes
1160 in Business Information Processing, Springer, 2012, pp. 11–23. doi:
1161 10.1007/978-3-642-30359-3_2.
1162 URL <http://dx.doi.org/10.1007/978-3-642-30359-3>
- 1163 [4] C. Di Ciccio, M. Mecella, A two-step fast algorithm for the auto-
1164 mated discovery of declarative workflows, in: Symposium on Com-
1165 putational Intelligence and Data Mining, IEEE, 2013, pp. 135–142.
1166 doi:10.1109/CIDM.2013.6597228.
1167 URL [http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?](http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6588692)
1168 [punumber=6588692](http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6588692)
- 1169 [5] F. M. Maggi, A. J. Mooij, W. M. P. van der Aalst, User-guided discovery
1170 of declarative process models, in: CIDM, IEEE, 2011, pp. 192–199.
1171 URL <http://dx.doi.org/10.1109/CIDM.2011.5949297>
- 1172 [6] F. M. Maggi, R. P. J. C. Bose, W. M. P. van der Aalst, Efficient discovery
1173 of understandable declarative process models from event logs, in: J. Ralyté,
1174 X. Franch, S. Brinkkemper, S. Wrycza (Eds.), CAiSE, Vol. 7328 of Lecture
1175 Notes in Computer Science, Springer, 2012, pp. 270–285.
1176 URL http://dx.doi.org/10.1007/978-3-642-31095-9_18
- 1177 [7] F. M. Maggi, M. Dumas, L. García-Bañuelos, M. Montali, Discovering
1178 data-aware declarative process models from event logs, in: Daniel et al.
1179 [76], pp. 81–96. doi:10.1007/978-3-642-40176-3_8.
- 1180 [8] C. Di Ciccio, F. M. Maggi, J. Mendling, Discovering target-branched de-
1181 clare constraints, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), Business
1182 Process Management, Vol. 8659 of Lecture Notes in Computer Science,
1183 Springer, 2014, pp. 34–50. doi:10.1007/978-3-319-10172-9_3.
- 1184 [9] B. F. van Dongen, Real-life event logs – logs from itil pro-
1185 cesses of rabobank group ict, Fourth International Business Pro-
1186 cess Intelligence Challenge (BPIC’14) (2014). doi:10.4121/uuid:
1187 c3e5d162-0cfd-4bb0-bd82-af5268819c35.
1188 URL [http://dx.doi.org/10.4121/uuid:](http://dx.doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35)
1189 [c3e5d162-0cfd-4bb0-bd82-af5268819c35](http://dx.doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35)
- 1190 [10] D. Fahland, D. Lübke, J. Mendling, H. A. Reijers, B. Weber, M. Weid-
1191 lich, S. Zugald, Declarative versus imperative process modeling languages:
1192 The issue of understandability, in: T. A. Halpin, J. Krogstie, S. Nurcan,
1193 E. Proper, R. Schmidt, P. Soffer, R. Ukor (Eds.), Enterprise, Business-
1194 Process and Information Systems Modeling, 10th International Workshop,

- 1195 BPMDs 2009, and 14th International Conference, EMMSAD 2009, held at
1196 CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings,
1197 Vol. 29 of Lecture Notes in Business Information Processing, Springer, 2009,
1198 pp. 353–366.
- 1199 [11] H. A. Reijers, T. Slaats, C. Stahl, Declarative modeling-an academic dream
1200 or the future for bpm?, in: Daniel et al. [76], pp. 307–322. doi:10.1007/
1201 978-3-642-40176-3_26.
1202 URL http://dx.doi.org/10.1007/978-3-642-40176-3_26
- 1203 [12] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium
1204 on Foundations of Software Technology and Theoretical Computer Science
1205 (FSTTCS), IEEE, 1977, pp. 46–57.
- 1206 [13] E. M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 2001.
- 1207 [14] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic
1208 logic on finite traces, in: F. Rossi (Ed.), IJCAI, IJCAI/AAAI, 2013, pp.
1209 854–860.
1210 URL [http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/
1211 view/6997](http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997)
- 1212 [15] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on LTL on finite
1213 traces: Insensitivity to infiniteness, in: C. E. Brodley, P. Stone (Eds.), Pro-
1214 ceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence,
1215 July 27 -31, 2014, Québec City, Québec, Canada., AAAI Press, 2014, pp.
1216 1027–1033.
1217 URL [http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/
1218 8575](http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8575)
- 1219 [16] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, Sym-
1220 bolic model checking: 10^{20} states and beyond, Inf. Comput. 98 (2) (1992)
1221 142–170. doi:10.1016/0890-5401(92)90017-A.
1222 URL [http://dx.doi.org/10.1016/0890-5401\(92\)90017-A](http://dx.doi.org/10.1016/0890-5401(92)90017-A)
- 1223 [17] D. Gries, Monotonicity in calculational proofs, in: E. Olderog, B. Steffen
1224 (Eds.), Correct System Design, Recent Insight and Advances, (to Hans
1225 Langmaack on the occasion of his retirement from his professorship at
1226 the University of Kiel), Vol. 1710 of Lecture Notes in Computer Science,
1227 Springer, 1999, pp. 79–85. doi:10.1007/3-540-48092-7_4.
1228 URL http://dx.doi.org/10.1007/3-540-48092-7_4
- 1229 [18] A. Burattin, F. M. Maggi, W. M. P. van der Aalst, A. Sperduti, Techniques
1230 for a posteriori analysis of declarative processes, in: C. Chi, D. Gasevic,
1231 W. van den Heuvel (Eds.), 16th IEEE International Enterprise Distributed
1232 Object Computing Conference, EDOC 2012, Beijing, China, September
1233 10-14, 2012, IEEE, 2012, pp. 41–50. doi:10.1109/EDOC.2012.15.
1234 URL <http://doi.ieeecomputersociety.org/10.1109/EDOC.2012.15>

- 1235 [19] C. Di Ciccio, M. Mecella, J. Mendling, The effect of noise on mined declarative constraints, in: P. Ceravolo, R. Accorsi, P. Cudre-Mauroux (Eds.),
1236 Data-Driven Process Discovery and Analysis, Vol. 203 of Lecture Notes in
1237 Business Information Processing, Springer Berlin Heidelberg, 2015, pp. 1–
1238 24. doi:10.1007/978-3-662-46436-6_1.
1239 URL http://dx.doi.org/10.1007/978-3-662-46436-6_1
1240
- 1241 [20] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large
1242 databases, in: J. B. Bocca, M. Jarke, C. Zaniolo (Eds.), VLDB, Morgan
1243 Kaufmann, 1994, pp. 487–499.
1244 URL <http://www.vldb.org/conf/1994/P487.PDF>
- 1245 [21] C. Di Ciccio, M. Mecella, On the discovery of declarative control flows for
1246 artful processes, ACM Trans. Manage. Inf. Syst. 5 (4) (2015) 24:1–24:37.
1247 doi:10.1145/2629447.
- 1248 [22] D. M. M. Schunselaar, F. M. Maggi, N. Sidorova, Patterns for a log-
1249 based strengthening of declarative compliance models, in: J. Derrick,
1250 S. Gnesi, D. Latella, H. Treharne (Eds.), IFM, Vol. 7321 of Lecture
1251 Notes in Computer Science, Springer, 2012, pp. 327–342. doi:10.1007/
1252 978-3-642-30729-4_23.
- 1253 [23] B. A. Hausmann, O. Ore, Theory of quasi-groups, American Journal of
1254 Mathematics 59 (4) (1937) 983–1004.
1255 URL <http://www.jstor.org/stable/2371362>
- 1256 [24] G. Birkhoff, Lattice theory, in: Colloquium Publications, 3rd Edition,
1257 Vol. 25, Amer. Math. Soc., 1967.
- 1258 [25] H. Mitsch, A natural partial order for semigroups, Proceedings of the Amer-
1259 ican Mathematical Society 97 (3) (1986) 384–388.
1260 URL <http://www.jstor.org/stable/2046222>
- 1261 [26] J. Mendling, M. Strembeck, J. Recker, Factors of process model compre-
1262 hension - findings from a series of experiments, Decision Support Systems
1263 53 (1) (2012) 195–206.
- 1264 [27] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows:
1265 Balancing between flexibility and support, Computer Science - R&D 23 (2)
1266 (2009) 99–113. doi:10.1007/s00450-009-0057-9.
1267 URL <http://dx.doi.org/10.1007/s00450-009-0057-9>
- 1268 [28] F. M. Maggi, R. P. J. C. Bose, W. M. P. van der Aalst, A knowledge-
1269 based integrated approach for discovering and repairing declare maps, in:
1270 C. Salinesi, M. C. Norrie, O. Pastor (Eds.), CAiSE, Vol. 7908 of Lecture
1271 Notes in Computer Science, Springer, 2013, pp. 433–448. doi:10.1007/
1272 978-3-642-38709-8_28.

- 1273 [29] B. F. van Dongen, Real-life event logs – a loan application process, Second
1274 International Business Process Intelligence Challenge (BPIC'12) (2012).
1275 doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
1276 URL [http://dx.doi.org/10.4121/uuid:
1277 3926db30-f712-4394-aebc-75976070e91f](http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f)
- 1278 [30] A. D. Bautista, L. Wangikar, S. M. K. Akbar, Process mining-driven optim-
1279 ization of a consumer loan approvals process - the BPIC 2012 challenge case
1280 study, in: Business Process Management Workshops - BPM 2012 Interna-
1281 tional Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers,
1282 2012, pp. 219–220.
- 1283 [31] W. M. P. van der Aalst, Process Mining: Discovery, Conformance
1284 and Enhancement of Business Processes, Springer, 2011. doi:10.1007/
1285 978-3-642-19345-3.
- 1286 [32] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat,
1287 E. Verbeek, T. Weijters, ProM: The process mining toolkit, in: A. K. A.
1288 de Medeiros, B. Weber (Eds.), BPM (Demos), Vol. 489 of CEUR Workshop
1289 Proceedings, CEUR-WS.org, 2009.
1290 URL <http://ceur-ws.org/Vol1-489/paper3.pdf>
- 1291 [33] R. Agrawal, D. Gunopulos, F. Leymann, Mining process models from work-
1292 flow logs, in: H.-J. Schek, G. Alonso, F. Saltor, I. Ramos (Eds.), Ad-
1293 vances in Database Technology – EDBT'98, Vol. 1377 of Lecture Notes
1294 in Computer Science, Springer Berlin / Heidelberg, 1998, pp. 467–483,
1295 10.1007/BFb0101003.
1296 URL <http://dx.doi.org/10.1007/BFb0101003>
- 1297 [34] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Dis-
1298 covering process models from event logs, IEEE Trans. Knowl. Data Eng.
1299 16 (9) (2004) 1128–1142.
1300 URL [http://csdl.computer.org/comp/trans/tk/2004/09/k1143abs.
1301 htm](http://csdl.computer.org/comp/trans/tk/2004/09/k1143abs.htm)
- 1302 [35] L. Wen, W. M. P. van der Aalst, J. Wang, J. Sun, Mining process models
1303 with non-free-choice constructs, Data Min. Knowl. Discov. 15 (2) (2007)
1304 145–180.
1305 URL <http://dx.doi.org/10.1007/s10618-007-0065-y>
- 1306 [36] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering workflow
1307 models from event-based data using little thumb, Integrated Computer-
1308 Aided Engineering 10 (2) (2003) 151–162.
1309 URL <http://iospress.metapress.com/content/8puq22eumrva7vyp/>
- 1310 [37] A. K. A. de Medeiros, A. J. M. M. Weijters, W. M. P. van der Aalst,
1311 Genetic process mining: an experimental evaluation, Data Min. Knowl.
1312 Discov. 14 (2) (2007) 245–304. doi:10.1007/s10618-006-0061-7.
1313 URL <http://dx.doi.org/10.1007/s10618-006-0061-7>

- 1314 [38] W. M. P. van der Aalst, V. Rubin, E. Verbeek, B. F. van Dongen,
1315 E. Kindler, C. W. Günther, Process mining: a two-step approach to bal-
1316 ance between underfitting and overfitting, *Software and Systems Modeling*
1317 9 (2010) 87–111, 10.1007/s10270-008-0106-z.
1318 URL <http://dx.doi.org/10.1007/s10270-008-0106-z>
- 1319 [39] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving petri
1320 nets from finite transition systems, *IEEE Trans. Comput.* 47 (8) (1998)
1321 859–882. doi:10.1109/12.707587.
1322 URL <http://dx.doi.org/10.1109/12.707587>
- 1323 [40] J. Desel, W. Reisig, The synthesis problem of petri nets, *Acta Informatica*
1324 33 (1996) 297–315, 10.1007/s002360050046.
1325 URL <http://dx.doi.org/10.1007/s002360050046>
- 1326 [41] D. Fahland, W. M. P. van der Aalst, Repairing process models to reflect
1327 reality, in: A. P. Barros, A. Gal, E. Kindler (Eds.), *BPM*, Vol. 7481 of
1328 *Lecture Notes in Computer Science*, Springer, 2012, pp. 229–245. doi:
1329 10.1007/978-3-642-32885-5_19.
1330 URL <http://dx.doi.org/10.1007/978-3-642-32885-5>
- 1331 [42] D. Fahland, W. M. P. van der Aalst, Model repair - aligning process models
1332 to reality, *Inf. Syst.* 47 (2015) 220–243. doi:10.1016/j.is.2013.12.007.
- 1333 [43] F. M. Maggi, D. Corapi, A. Russo, E. Lupu, G. Visaggio, Revising pro-
1334 cess models through inductive learning, in: *Business Process Management*
1335 *Workshops - BPM 2010 International Workshops and Education Track*,
1336 Hoboken, NJ, USA, September 13-15, 2010, Revised Selected Papers, 2010,
1337 pp. 182–193.
- 1338 [44] C. Di Ciccio, A. Marrella, A. Russo, Knowledge-intensive Processes: Char-
1339 acteristics, requirements and analysis of contemporary approaches, *J. Data*
1340 *Semantics* 4 (1) (2015) 29–57. doi:10.1007/s13740-014-0038-4.
- 1341 [45] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, Declare: Full support
1342 for loosely-structured processes, in: *EDOC*, IEEE Computer Society, 2007,
1343 pp. 287–300.
1344 URL <http://doi.ieeecomputersociety.org/10.1109/EDOC.2007.25>
- 1345 [46] F. Chesani, P. Mello, M. Montali, P. Torroni, Verification of choreographies
1346 during execution using the reactive event calculus, in: *WS-FM*, 2008, pp.
1347 55–72.
- 1348 [47] T. T. Hildebrandt, R. R. Mukkamala, T. Slaats, Designing a cross-
1349 organizational case management system using dynamic condition response
1350 graphs, in: *Proceedings of the 15th IEEE International Enterprise Dis-*
1351 *tributed Object Computing Conference*, *EDOC 2011*, Helsinki, Finland,
1352 August 29 - September 2, 2011, 2011, pp. 161–170.

- 1353 [48] T. T. Hildebrandt, R. R. Mukkamala, T. Slaats, F. Zanitti, Contracts
1354 for cross-organizational workflows as timed dynamic condition response
1355 graphs, *J. Log. Algebr. Program.* 82 (5-7) (2013) 164–185.
- 1356 [49] T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as
1357 distributed dynamic condition response graphs, in: K. Honda, A. Mycroft
1358 (Eds.), *PLACES*, Vol. 69 of *EPTCS*, 2010, pp. 59–73.
1359 URL <http://dx.doi.org/10.4204/EPTCS.69.5>
- 1360 [50] G. Greco, A. Guzzo, D. Saccà, Simulations on workflow management sys-
1361 tems: A framework based on event choice datalog, *Intelligenza Artificiale*
1362 5 (2) (2011) 189–206. doi:10.3233/IA-2011-0022.
1363 URL <http://dx.doi.org/10.3233/IA-2011-0022>
- 1364 [51] C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Ping-
1365 gera, B. Weber, Understanding declare models: strategies, pitfalls, em-
1366 pirical results, *Software & Systems Modeling* (2014) 1–28doi:10.1007/
1367 s10270-014-0435-z.
1368 URL <http://dx.doi.org/10.1007/s10270-014-0435-z>
- 1369 [52] F. M. Maggi, T. Slaats, H. A. Reijers, The automated discovery of hybrid
1370 processes, in: S. W. Sadiq, P. Soffer, H. Völzer (Eds.), *Business Process*
1371 *Management - 12th International Conference, BPM 2014, Haifa, Israel,*
1372 *September 7-11, 2014. Proceedings*, Vol. 8659 of *Lecture Notes in Computer*
1373 *Science*, Springer, 2014, pp. 392–399. doi:10.1007/978-3-319-10172-9_
1374 27.
1375 URL http://dx.doi.org/10.1007/978-3-319-10172-9_27
- 1376 [53] N. C. Silva, C. A. L. de Oliveira, F. A. L. A. Albino, R. M. F. Lima, De-
1377 clarative versus imperative business process languages - A controlled exper-
1378 iment, in: S. Hammoudi, L. A. Maciaszek, J. Cordeiro (Eds.), *ICEIS 2014*
1379 *- Proceedings of the 16th International Conference on Enterprise Informa-*
1380 *tion Systems*, Volume 3, Lisbon, Portugal, 27-30 April, 2014, *SciTePress*,
1381 2014, pp. 394–401. doi:10.5220/0004896203940401.
1382 URL <http://dx.doi.org/10.5220/0004896203940401>
- 1383 [54] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H. A. Reijers,
1384 Imperative versus declarative process modeling languages: An empirical
1385 investigation, in: *Business Process Management Workshops - BPM 2011*
1386 *International Workshops, Clermont-Ferrand, France, August 29, 2011, Re-*
1387 *vised Selected Papers, Part I*, 2011, pp. 383–394.
- 1388 [55] C. Haisjackl, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera,
1389 B. Weber, Making sense of declarative process models: Common strategies
1390 and typical pitfalls, in: *Enterprise, Business-Process and Information Sys-*
1391 *tems Modeling - 14th International Conference, BPMDS 2013, 18th Inter-*
1392 *national Conference, EMMSAD 2013, Held at CAiSE 2013, Valencia,*
1393 *Spain, June 17-18, 2013. Proceedings*, 2013, pp. 2–17.

- 1394 [56] O. Kupferman, M. Y. Vardi, Vacuity detection in temporal model checking,
1395 STTT 4 (2) (2003) 224–233. doi:10.1007/s100090100062.
1396 URL dx.doi.org/10.1007/s100090100062
- 1397 [57] R. P. J. C. Bose, F. M. Maggi, W. M. P. van der Aalst, Enhancing declare
1398 maps based on event correlations, in: Business Process Management - 11th
1399 International Conference, BPM 2013, Beijing, China, August 26-30, 2013.
1400 Proceedings, 2013, pp. 97–112.
- 1401 [58] F. M. Maggi, Declarative process mining with the declare component of
1402 ProM, in: M.-C. Fauvet, B. F. van Dongen (Eds.), BPM (Demos), Vol.
1403 1021 of CEUR Workshop Proceedings, CEUR-WS.org, 2013.
1404 URL http://ceur-ws.org/Vol1-1021/paper_8.pdf
- 1405 [59] E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Inducing declarative
1406 logic-based models from labeled traces, in: G. Alonso, P. Dadam,
1407 M. Rosemann (Eds.), BPM, Vol. 4714 of Lecture Notes in Computer Science,
1408 Springer, 2007, pp. 344–359. doi:10.1007/978-3-540-75183-0_25.
1409 URL http://dx.doi.org/10.1007/978-3-540-75183-0_25
- 1410 [60] E. Lamma, P. Mello, F. Riguzzi, S. Storari, Applying inductive logic programming
1411 to process mining, in: H. Blockeel, J. Ramon, J. W. Shavlik,
1412 P. Tadepalli (Eds.), ILP, Vol. 4894 of Lecture Notes in Computer Science,
1413 Springer, 2007, pp. 132–146. doi:10.1007/978-3-540-78469-2_16.
1414 URL http://dx.doi.org/10.1007/978-3-540-78469-2_16
- 1415 [61] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Exploiting
1416 inductive logic programming techniques for declarative process
1417 mining, T. Petri Nets and Other Models of Concurrency 2 (2009) 278–
1418 295.
1419 URL http://dx.doi.org/10.1007/978-3-642-00899-3_16
- 1420 [62] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni,
1421 Verifiable agent interaction in abductive logic programming: The sciff
1422 framework, ACM Trans. Comput. Log. 9 (4) (2008) 29:1–29:43. doi:
1423 10.1145/1380572.1380578.
- 1424 [63] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic logic-based process mining,
1425 in: W. Faber, N. Leone (Eds.), CILC, Vol. 598 of CEUR Workshop
1426 Proceedings, CEUR-WS.org, 2010.
1427 URL <http://ceur-ws.org/Vol1-598/paper17.pdf>
- 1428 [64] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic declarative process mining,
1429 in: Y. Bi, M.-A. Williams (Eds.), KSEM, Vol. 6291 of Lecture
1430 Notes in Computer Science, Springer, 2010, pp. 292–303. doi:10.1007/
1431 978-3-642-15280-1_28.
1432 URL http://dx.doi.org/10.1007/978-3-642-15280-1_28

- 1433 [65] M. Richardson, P. Domingos, Markov logic networks, *Machine Learning*
1434 62 (1-2) (2006) 107–136. doi:10.1007/s10994-006-5833-1.
1435 URL <http://dx.doi.org/10.1007/s10994-006-5833-1>
- 1436 [66] C. Di Ciccio, M. Mecella, Mining artful processes from knowledge workers’
1437 emails, *IEEE Internet Computing* 17 (5) (2013) 10–20. doi:10.1109/MIC.
1438 2013.60.
- 1439 [67] M. Răim, C. Di Ciccio, F. M. Maggi, M. Mecella, J. Mendling, Log-based
1440 understanding of business processes through temporal logic query checking,
1441 in: R. Meersman, H. Panetto, T. S. Dillon, M. Missikoff, L. Liu, O. Pastor,
1442 A. Cuzzocrea, T. Sellis (Eds.), *International Conference on Cooperative In-*
1443 *formation Systems, On the Move to Meaningful Internet Systems Confed-*
1444 *erated International Conferences*, Vol. 8841 of *Lecture Notes in Computer*
1445 *Science*, Springer, 2014, pp. 75–92. doi:10.1007/978-3-662-45563-0_5.
1446 URL <http://dx.doi.org/10.1007/978-3-662-45563-0>
- 1447 [68] F. M. Maggi, A. Burattin, M. Cimitile, A. Sperduti, Online process dis-
1448 covery to detect concept drifts in ltl-based declarative process models, in:
1449 *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*
1450 *- Confederated International Conferences: CoopIS, DOA-Trusted Cloud,*
1451 *and ODBASE 2013*, Graz, Austria, September 9-13, 2013. *Proceedings,*
1452 *2013*, pp. 94–111.
- 1453 [69] M. Westergaard, F. M. Maggi, Looking into the future. using timed auto-
1454 mata to provide a priori advice about timed declarative process models,
1455 in: R. Meersman, H. Panetto, T. S. Dillon, S. Rinderle-Ma, P. Dadam,
1456 X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, I. F. Cruz (Eds.),
1457 *On the Move to Meaningful Internet Systems: OTM 2012, Confeder-*
1458 *ated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012,*
1459 *Rome, Italy, September 10-14, 2012. Proceedings, Part I, Vol. 7565 of*
1460 *Lecture Notes in Computer Science*, Springer, 2012, pp. 250–267. doi:
1461 10.1007/978-3-642-33606-5_16.
1462 URL http://dx.doi.org/10.1007/978-3-642-33606-5_16
- 1463 [70] F. M. Maggi, Discovering metric temporal business constraints from event
1464 logs, in: *Perspectives in Business Informatics Research - 13th International*
1465 *Conference, BIR 2014*, Lund, Sweden, September 22-24, 2014. *Proceedings,*
1466 *2014*, pp. 261–275.
- 1467 [71] M. L. Bernardi, M. Cimitile, C. D. Francescomarino, F. M. Maggi, Using
1468 discriminative rule mining to discover declarative process models with non-
1469 atomic activities, in: *Rules on the Web. From Theory to Applications*
1470 *- 8th International Symposium, RuleML 2014, Co-located with the 21st*
1471 *European Conference on Artificial Intelligence, ECAI 2014*, Prague, Czech
1472 Republic, August 18-20, 2014. *Proceedings, 2014*, pp. 281–295.

- 1473 [72] M. Montali, F. M. Maggi, F. Chesani, P. Mello, W. M. P. van der Aalst,
1474 Monitoring business constraints with the event calculus, *ACM TIST* 5 (1)
1475 (2013) 17.
- 1476 [73] M. Montali, F. Chesani, F. M. Maggi, P. Mello, Towards data-aware con-
1477 straints in declare, in: *SAC*, ACM Press and Addison Wesley, 2013, pp.
1478 1391–1396.
- 1479 [74] R. D. Masellis, F. M. Maggi, M. Montali, Monitoring data-aware business
1480 constraints with finite state automata, in: *International Conference on*
1481 *Software and Systems Process 2014, ICSSP '14*, Nanjing, China - May 26
1482 - 28, 2014, 2014, pp. 134–143.
- 1483 [75] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on
1484 multi-perspective declarative process models, *CoRR* abs/1503.04957.
- 1485 [76] F. Daniel, J. Wang, B. Weber (Eds.), *Business Process Management - 11th*
1486 *International Conference, BPM 2013*, Beijing, China, August 26-30, 2013.
1487 *Proceedings*, Vol. 8094 of *Lecture Notes in Computer Science*, Springer,
1488 2013. doi:10.1007/978-3-642-40176-3.

This document is a pre-print copy of the manuscript
(Di Ciccio, Maggi, and Mendling 2016)
published by Elsevier.

The final version of the paper is identified by DOI: [10.1016/j.is.2015.06.009](https://doi.org/10.1016/j.is.2015.06.009)

References

Di Ciccio, Claudio, Fabrizio Maria Maggi, and Jan Mendling (2016). “Efficient discovery of Target-Branched Declare constraints”. In: *Information Systems* 56, pp. 258–283. ISSN: 0306-4379. DOI: [10.1016/j.is.2015.06.009](https://doi.org/10.1016/j.is.2015.06.009).

BibTeX

```
@Article{
  DiCiccio.etal/IS2016:EfficientDiscoveryTarget,
  author      = {Di Ciccio, Claudio and Fabrizio Maria Maggi and Jan
                Mendling},
  journal     = {Information Systems},
  title       = {Efficient discovery of {T}arget-{B}ranched {D}eclare
                constraints},
  year        = {2016},
  issn        = {0306-4379},
  pages       = {258-283},
  volume      = {56},
  doi         = {10.1016/j.is.2015.06.009},
  keywords    = {Process mining},
  publisher   = {Elsevier}
}
```