# Using Discriminative Rule Mining to Discover Declarative Process Models with Non-atomic Activities

Mario Luca Bernardi[1], Marta Cimitile[2],
Chiara Di Francescomarino[3], and Fabrizio Maria Maggi[4]

[1] University of Sannio, Benevento, Italy
mlbernar@unisannio.it
[2] Unitelma Sapienza University, Rome, Italy
marta.cimitile@unitelma.it
[3] FBK-IRST, Trento, Italy
dfmchiara@fbk.eu
[4] University of Tartu, Tartu, Estonia
f.m.maggi@ut.ee

**Abstract.** Process discovery techniques try to generate process models from execution logs. Declarative process modeling languages are more suitable than procedural notations for representing the discovery results deriving from logs of processes working in dynamic and low-predictable environments. However, existing declarative discovery approaches aim at mining declarative specifications considering each activity in a business process as an atomic/instantaneous event. In spite of this, often, in realistic environments, process activities are not instantaneous; rather, their execution spans across a time interval and is characterized by a sequence of states of a transactional lifecycle. In this paper, we investigate how to use discriminative rule mining in the discovery task, to characterize lifecycles that determine constraint violations and lifecycles that ensure constraint fulfillments. The approach has been implemented as a plug-in of the process mining tool ProM and validated on synthetic logs and on a real-life log recorded by an incident and problem management system called VINST in use at Volvo IT Belgium.

**Keywords:** Process Discovery, Rule Mining, Discriminative Mining, Non-Atomic Activities, Activity lifecycle, Linear Temporal Logic.

## 1 Introduction

Process discovery techniques are widely considered as critical for successful business process management and monitoring. In particular, the discovery of declarative models can be used in complex environments where process executions involve multiple alternatives and high flexibility is needed [2] [9]. Consider, for example, a business process for handling natural disasters. This type of process is totally unpredictable and should be adapted every time to specific conditions characteristic of specific cases. Using declarative models for describing processes like this, allows analysts to define generic constraints to be followed during the process execution instead of explicitly representing the flows of events allowed. At runtime, anything that does not violate these constraints

A. Bikakis et al. (Eds.): RuleML 2014, LNCS 8620, pp. 281–295, 2014.
© Springer International Publishing Switzerland 2014

is possible. In this way, process participants are free to adapt their tasks to the environment characteristics as long as these general rules are respected. At the same time, models remain under-specified and easy to understand for humans.

Existing process discovery techniques for generating declarative specifications, do not take activity lifecycles and their characteristics into consideration, even if, in many practical cases, activities are non-atomic. In the reality, activities have a duration spanning over time intervals in which transactional states (a.k.a. *event types*) of the activity can occur. The sequences of event types that occur when an activity is executed, characterize the lifecycle of that activity. For example, when an activity $a$ is executed, the lifecycle $\langle a_{assign}, a_{start}, a_{complete}\rangle$ can take place, including event types *assign*, *start*, and *complete*. If available, this information is very relevant to be considered when mining an event log, since it allows analysts to understand not only the constraints between activities but also the ones that relate event types appearing inside the lifecycle of one single activity.

In 2010, the IEEE Task Force on Process Mining has adopted XES (eXtensible Event Stream) [19] as the standard for storing data in event logs. XES supports a specific extension (Lifecycle Extension) to keep track of information related to the lifecycle of an activity in a log. In addition, XES defines a standard transactional model for activity lifecycles in a log through a state machine describing the allowed sequences of event types for an activity.

Starting from this definition, in this paper, we present a novel approach to discover declarative specifications from logs with a strong focus on the activity lifecycles. For describing declarative models, we use Declare, a declarative process modeling language, first introduced in [15], that combines a formal semantics grounded in Linear Temporal Logic (LTL) on finite traces[1] with a graphical representation. Here, we slightly modify the original semantics of Declare constraints to adapt it to the non-atomic case. In addition, the proposed approach relies on the notion of *constraint activation* [5]. For the constraint "every request is eventually acknowledged" each request is an activation. This activation becomes a fulfillment or a violation depending on whether the request is followed by an acknowledgement or not.

In our approach, in a first phase, starting from a log, we try to group together events belonging to the same lifecycle of the same activity (discharging the "malformed" lifecycles according to an input transactional model for activity lifecycles like, e.g., XES). The lifecycle identification can be done using (i) a FIFO-based approach [6], or (ii) event correlations [3]. The FIFO -based approach is a typical "conservative approach" first in-first out, in which if a new upcoming event can be connected to two events occurred in the past and belonging to two different lifecycles, the priority is given to the one that occurred before. With event correlations, events are connected as part of the same lifecycle whenever they share common values for some data attributes. This approach can only be applied if events in the log carry data. In our implemented prototype, we have used the FIFO-based approach, but the tool can be easily extended to support lifecycle identification through event correlations.

In a second phase of our approach, we generate a set of candidate Declare constraints (over non-atomic activities) considering the constraints that are most frequently

[1] For compactness, we will use the LTL acronym to denote LTL on finite traces.

activated in the log. In this way, we discover information about the inter-relations between lifecycles of different activities.

In a third phase, we use discriminative rule mining to retrieve characteristics of the lifecycle of an activation of a constraint that discriminate between cases in which that activation is a fulfillment and the cases in which the activation leads to a violation for that constraint. For example, we want to find rules like "if a registration ends with an abortion the user cannot be notified via e-mail", or "if an analysis is suspended more than twice then eventually a check should be executed". Using discriminative rule mining we can discover intra-relations between transactional states inside the lifecycle of a constraint activation that discriminate between cases in which the activation is a fulfillment and cases in which the activation is a violation. In particular, we use a decision tree to learn from contrasting training data discriminative rules related to the lifecycle control flow. Lifecycles are encoded, in the form of Declare constraints, as features of the tree and the resulting decision rules are used to express the lifecycle characteristics able to discriminate between fulfillments and violations.

The approach presented in this paper has been implemented as a plug-in of the ProM[2] process mining toolset. This prototype has been used to validate our technique on both synthetic logs and a real life log recorded by an incident and problem management system called VINST in use at Volvo IT Belgium.

The paper is structured as follows. Section 2 provides a preliminary background about the Declare language, introduces the concepts of non-atomic activities and activity lifecycles and provides an overview on discriminative rule mining. Next, Section 3 illustrates the approach, based on the combination of declarative process mining algorithms (extended to the non-atomic logs) and discriminative mining approaches. In Section 4, the experimentation is discussed. Section 5 reports some conclusion and future work.

## 2 Background

In this section, we introduce some preliminary knowledge needed to understand the techniques presented in this paper. In particular, in Section 2.1, we give an overview of the Declare language. In Section 2.2, we describe the transactional model for activity lifecycles defined in the XES standard. Finally, in Section 2.3, we give some background about discriminative rule mining.

### 2.1 Declare: The Language

Declare is a language for describing declarative process models first introduced in [15]. A Declare model consists of a set of constraints applied to (atomic) activities. Constraints, in turn, are based on templates. Templates are abstract parameterized patterns and constraints are their concrete instantiations on real activities. Templates have a user-friendly graphical representation understandable to the user and their semantics can be formalized using different logics [14], the main one being Linear Temporal Logic

[2] `www.processmining.org`

**Table 1.** Graphical notation and LTL formalization of some Declare templates

| TEMPLATE | FORMALIZATION | NOTATION |
|---|---|---|
| responded existence(A,B) | $\Diamond A \rightarrow \Diamond B$ | A B |
| response(A,B) | $\Box(A \rightarrow \Diamond B)$ | A B |
| precedence(A,B) | $\neg B\, \mathcal{W}\, A$ | A B |
| alternate response(A,B) | $\Box(A \rightarrow \bigcirc(\neg A\, \mathcal{U}\, B))$ | A B |
| alternate precedence(A,B) | $(\neg B\, \mathcal{W}\, A) \wedge \Box(B \rightarrow \bigcirc(\neg B\, \mathcal{W}\, A))$ | A B |
| chain response(A,B) | $\Box(A \rightarrow \bigcirc B)$ | A B |
| chain precedence(A,B) | $\Box(\bigcirc B \rightarrow A)$ | A B |
| not resp. existence(A,B) | $\Diamond A \rightarrow \neg\Diamond B$ | A B |
| not response(A,B) | $\Box(A \rightarrow \neg\Diamond B)$ | A B |
| not precedence(A,B) | $\Box(A \rightarrow \neg\Diamond B)$ | A B |
| not chain response(A,B) | $\Box(A \rightarrow \neg\bigcirc B)$ | A B |
| not chain precedence(A,B) | $\Box(A \rightarrow \neg\bigcirc B)$ | A B |

over finite traces, making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its templates. The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. Table 1 summarizes some Declare templates. The reader can refer to [1] for a full description of the language. Here, we indicate template parameters with capital letters (see Table 1) and real activities in their instantiations with lower case letters (e.g., constraint $\Box(a \rightarrow \Diamond b)$).

Consider, for example, the *response* constraint $\Box(a \rightarrow \Diamond b)$. This constraint indicates that if *a occurs*, *b* must eventually *follow*. Therefore, this constraint is satisfied for traces such as $\langle a, a, b, c\rangle$, $\langle b, b, c, d\rangle$, and $\langle a, b, c, b\rangle$, but not for $\langle a, b, a, c\rangle$ because, in this case, the second instance of $a$ is not followed by a $b$. Note that, in trace $\langle b, b, c, d\rangle$, the considered response constraint is satisfied in a trivial way because $a$ never occurs. In this case, we say that the constraint is *vacuously satisfied* [11]. An *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events in the same trace. For example, $a$ is an activation for the *response* constraint $\Box(a \rightarrow \Diamond b)$, because the execution of $a$ forces $b$ to be executed eventually.

An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the response constraint $\Box(a \rightarrow \Diamond b)$. In trace $\langle a, a, b, c\rangle$, the constraint is activated and fulfilled twice, whereas, in trace $\langle a, b, c, b\rangle$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\langle a, b, a, c\rangle$, for example, the response constraint $\Box(a \rightarrow \Diamond b)$ is activated twice, but the first activation leads to a fulfillment (eventually $b$ occurs) and the second activation leads to a violation ($b$ does not occur subsequently).

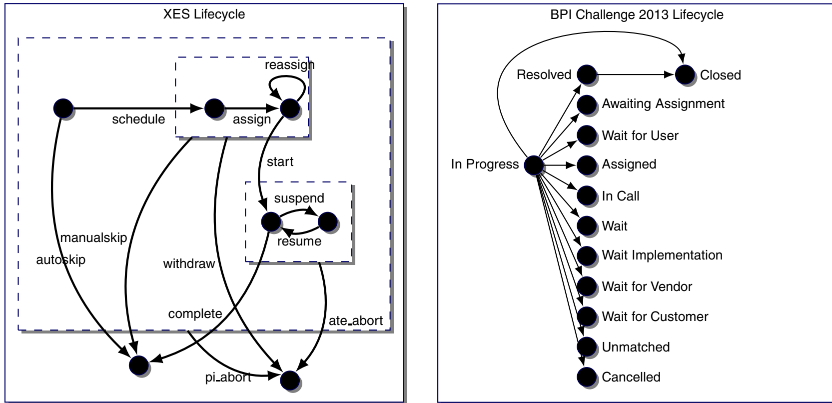


**Fig. 1.** XES Standard and BPI Challenge 2013 transactional models for activity lifecycles

## 2.2 Activity Lifecycle

In real business applications, activities cannot be considered as atomic/instantaneous events, but they traverse different states in their lifecycle. Consider, for example, activity *Check*. This activity, during its execution can traverse different transactional states like, e.g., (i) *schedule* ($Check_{schedule}$), meaning that a check has been scheduled, (ii) *start* ($Check_{start}$), indicating that the check activity has started its execution, and (iii) *complete* ($Check_{complete}$), meaning that the check has been completed.

A transactional model for activity lifecycles represents the set of the admissible sequences of states that an activity can assume during its lifecycle. Figure 1 depicts the XES standard transactional model [10], which allows keeping track in the log of information related to activity lifecycles. In particular, the model is provided as a state machine describing the admissible flows of the different states an activity can assume. In addition, using the Lifecycle Extension provided in XES, users can customize the transactional model allowed in an event log.

Figure 2 reports an example that we will use as running example throughout the paper. In the figure, $\mathbf{t}_0$ represents an execution trace containing only atomic activities (*a*, *b*, and *c*). $\mathbf{t}_1$ includes non-atomic activities and different event types assumed by activities *a*, *b* and *c* (based on the XES standard transactional model). Activity *a* appears with three event types (once with *assign*, and twice with *start* and *complete*) *b* occurs with two event types (twice with *start* and *complete*), and activity *c* appears only with event type *start*. From this example, it is clear that we need a mechanism to connect events belonging to the same activity lifecycle together. In this paper, we use a conservative, FIFO-based approach. Using this approach, events at positions 1, 3 and 6 for activity *a* are grouped together, and, also, events at positions 5 and 7. Events at positions 2 and 4, and events at positions 8 and 10 for activity *b* are grouped together. Notice that, executions of activities can overlap. For example, the first lifecycle of activity *a* in the figure completes after that the first lifecycle of activity *b* has started. Also notice that some lifecycles are malformed, like the one of activity *c* that does not have a completion.

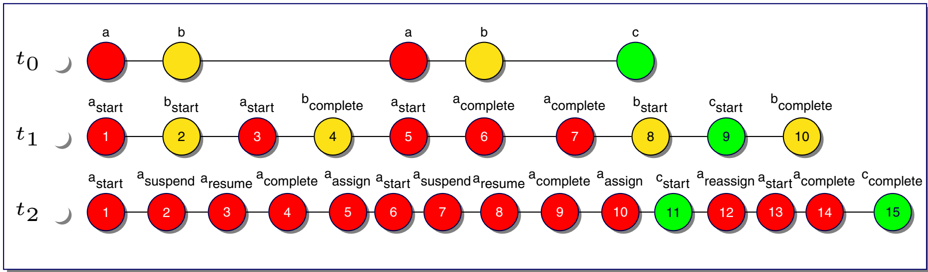


**Fig. 2.** An example of a trace including atomic activities ($t_0$) and traces composed of non-atomic activities ($t_1$ and $t_2$)

### 2.3 Discriminative Mining

Discriminative mining aims at extracting, from an existing set of data, patterns that are discriminative with respect to a given criterion. In the literature, several branches of works, differing with each other based on the type of mined patterns (e.g., sequences or rules), fall under this umbrella, like, for example, discriminative pattern mining [8], discriminative sequence mining [12] or classification rule approaches [18]. All these approaches are usually based on supervised or non-supervised learning techniques. In this work, we exploit decision tree supervised learning [7] in order to mine a set of declarative rules that discriminate between fulfillments and violations of a given constraint. Decision trees have been applied in the context of discriminative rule mining (also on top of other techniques) for their capability to construct readable rules [18]. Decision tree learning uses a decision tree as a model to predict the value of a target variable based on input variables (features). Decision trees are built from a set of training dataset. Each internal node of the tree is labeled with an input feature. Arcs stemming from a node labeled with a feature are labeled with possible values or value ranges of the feature. Each leaf of the decision tree is labeled with a class, i.e., a value of the target variable given the values of the input variables represented by the path from the root to the leaf. Moreover, each leaf of the decision tree is associated with a support (*support*) and a probability distribution (*class probability*). Support represents the number of examples in the training dataset that follow the path from the root to the leaf and that are correctly classified; class probability is the percentage of examples correctly classified with respect to all examples following that specific path.

In this work we rely on the Weka J48 implementation of one of the most known decision tree algorithms, the C4.5 algorithm [16], which exploits the normalized information gain to choose, for each node of the tree, the feature to be used for splitting the set of examples.

## 3 Approach

Figure 3 illustrates our proposed approach. Given a log containing non-atomic activities and a transactional model for activity lifecycles as a reference, it returns as output (i) a set of Declare constraints with non-atomic activities, and (ii) for each constraint, a set of characteristics of the involved activity lifecycles, which are possibly related to the

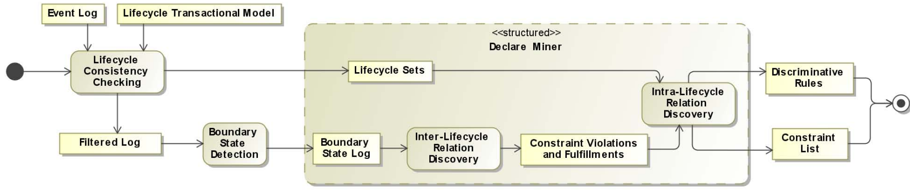


**Fig. 3.** An overview of the proposed approach

fulfillment or to the violation of the constraint. Roughly speaking, the approach takes into consideration a list of candidate Declare constraints (involving non-atomic activities), and it uses standard discovery techniques [13] for identifying fulfillments and violations for each of them. Then, a discriminative rule mining approach is used to find the characteristics of the lifecycle of an activation of a candidate constraint that allows us to discriminate between cases in which that activation was a fulfillment, and cases in which the activation was a violation for the considered constraint. In the following sections, we describe the main steps of the approach in detail.

## 3.1 Lifecycle Consistency Checking

The first step of our proposed technique aims at processing the input log to (i) connect together activity states belonging to the same lifecycle, and (ii) remove all the "malformed" lifecycles that are not consistent with the input transactional model. As already mentioned in Section 2, there are two possibilities for grouping event types of an activity belonging to the same lifecycle. One way to do it is by using event correlations as explained in [3]. If the event log contains (event) data attributes, it is possible to connect activity states that share some data values, e.g., an event id. However, this approach is applicable only if attributes that can be used to connect events exist. Therefore, for our experimentation, we decided to implement a conservative approach that is less precise, but applicable also in cases of logs with no data attributes or with data attributes that cannot be used for event correlation. In particular, we use a FIFO-based algorithm. We explain this approach using our running example in Fig. 2, and using the XES standard transactional model for activity lifecycles as reference. Based on this transactional model, events in trace $\mathbf{t}_1$ of the example can be grouped in separate lifecycles in different ways. For example, $a_{complete}$ at position 6 can be connected with $a_{start}$ at position 3, or with $a_{start}$ at position 5. Applying our FIFO-based algorithm, we can disambiguate the correlation using a conservative approach. This means that $a_{complete}$ at position 6 is connected with the $a_{start}$ event that occurred first, i.e., the one at position 3. Following this approach, we can identify, in trace $\mathbf{t}_1$, the following lifecycles:

- $a_1 = \langle a_{assign}(1), a_{start}(3), a_{complete}(6)\rangle$,
- $a_2 = \langle a_{start}(5), a_{complete}(7)\rangle$,
- $b_1 = \langle b_{start}(2), b_{complete}(4)\rangle$,
- $b_2 = \langle b_{start}(8), b_{complete}(10)\rangle$,
- $c_1 = \langle c_{start}(9)\rangle$,

where numbers between brackets indicate the position of the event in the trace.

As already mentioned, when we have all the lifecycles grouped together we can filter out the ones that are inconsistent with respect to the input transactional model. For

**Table 2.** LTL formalization of some Declare templates with non-atomic activities

| TEMPLATE | FORMALIZATION |
|---|---|
| responded existence(A,B) | $\Diamond A_i \rightarrow \Diamond B_i$ |
| response(A,B) | $\Box(A_f \rightarrow \Diamond B_i)$ |
| precedence(A,B) | $\neg B_i \,\mathcal{W}\, A_f$ |
| alternate response(A,B) | $\Box(A_f \rightarrow \bigcirc(\neg A_f \,\mathcal{U}\, B_i))$ |
| alternate precedence(A,B) | $(\neg B_i \,\mathcal{W}\, A_f) \wedge \Box(B_i \rightarrow \bigcirc(\neg B_i \,\mathcal{W}\, A_f))$ |
| chain response(A,B) | $\Box(A_f \rightarrow \bigcirc B_i)$ |
| chain precedence(A,B) | $\Box(\bigcirc B_i \rightarrow A_f)$ |
| not responded existence(A,B) | $\Diamond A_i \rightarrow \neg\Diamond B_i$ |
| not response(A,B) | $\Box(A_f \rightarrow \neg\Diamond B_i)$ |
| not precedence(A,B) | $\Box(A_f \rightarrow \neg\Diamond B_i)$ |
| not chain response(A,B) | $\Box(A_f \rightarrow \neg\bigcirc B_i)$ |
| not chain precedence(A,B) | $\Box(A_f \rightarrow \neg\bigcirc B_i)$ |

example, in our case, lifecycle $c_1$ is not allowed in the XES standard model, since a completion is missing. For this reason, this lifecycle will be filtered out and not considered for further analysis. The outputs of this step of our proposed approach are, respectively, the filtered log and the list of all the activities lifecycles (described by sequences of event types) that are contained in the log.

### 3.2 Boundary State Detection

The aim of this step is to abstract the activity lifecycles in the log by replacing them with placeholders marking the start and the end of each lifecycle in the log. This transformation is needed to discover Declare constraints with semantics for non-atomic activities described in Table 2. The formulas are straightforward and directly follow by the corresponding formulas in standard Declare. The idea is that, for verifying constraints involving non-atomic activities, it is sufficient to take into account the boundary states of lifecycles (in the table the initial state is indicated with "*i*" and the final one with "*f*"), abstracting away from the lifecycle details. For example, for the *response* template, it is enough to verify that the final state of activity $A$ ($A_f$) is followed by the initial state $B$ ($B_i$). In general, we consider most of the constraints valid for non-overlapping lifecycles. The only exceptions are the semantics of templates *responded existence* and *not responded existence* that we consider valid also for overlapping lifecycles of parallel activities.

In this step of the approach, we take as input the filtered log obtained by the previous step and produce a new log in which internal states of each lifecycle (i.e., states that are neither initial nor final) are filtered out. Trace $\mathbf{t}_1$ of our running example would be transformed into trace $\langle a_{assign}, b_{start}, b_{complete}, a_{start}, a_{complete}, a_{complete}, b_{start}, b_{complete}\rangle$. Event $a_{start}(3)$ is filtered out because it is not a boundary event in lifecycle $a_1 = (a_{assign}(1), a_{start}(3), a_{complete}(6))$. Events in the log are further transformed by abstracting away from the specific event type that corresponds to the first or the last

event of each lifecycle: each starting state will be indicated with "*i*" and each final state with "*f*". For instance, $\mathbf{t}_1$ would become $\langle a_i, b_i, b_f, a_i, a_f, a_f, b_i, b_f \rangle$.

### 3.3 Discovering Inter-Lifecycle Relations

In this step, the boundary state log derived from the previous step is mined using the approach presented in [3] to discover Declare constraints with semantics for non-atomic activities described in Table 2. As shown in the table, this semantics take into consideration only the boundary events of the activity lifecycles. The outcome of this step of the approach is a set of candidate Declare constraints connecting elements of different lifecycles (inter-lifecycle relations). For each candidate, we extract the set of fulfillments and the set of violations in the log. These sets will be input of the supervised learning problem defined in the next step of our proposed technique needed to find intra-lifecycle relations discriminating between lifecycles of constraint activations that eventually lead to a fulfillment and lifecycles of activations that lead to a constraint violation.

### 3.4 Discovering Intra-Lifecycle Relations

In the previous step of our proposed approach, we extract fulfillments and violations for a list of candidate constraints describing inter-lifecycle relations between activities in the log. For example, in trace $\mathbf{t}_1$ of our running example, $b_{complete}(4)$ is a fulfillment for constraint $\Box(b_f \rightarrow \Diamond a_i)$ (in this case $b_f$ is eventually followed by $a_i$), whereas $b_{complete}(10)$ is a violation for the same constraint. Note that, at this point of our approach, we take again into consideration the entire lifecycles connected to the boundary events in the log and we analyze their control flow characteristics to discriminate between lifecycles of activations that are fulfillments and the ones that are violations for each candidate constraint. These characteristics will be expressed, in turn, using Declare. In this case, the Declare constraints express intra-lifecycle relations between event types of the same activity.

This problem can be reformulated in terms of a supervised learning problem. For each candidate constraint $constr$, defined over a pair $(a, b)$, with $a$ activation of $constr$, the features of the lifecycle of $a$ discriminating with respect to fullfillments/violations of $constr$, are learned from a set $L$ of sequences representing all the lifecycles of $a$ in the log. These sequences are classified in two sets $L_{ful}$ and $L_{viol}$ according to whether the lifecycle corresponds to a fulfillment or to a violation of $constr$.

For example, consider traces $\mathbf{t}_1$ and $\mathbf{t}_2$ in our running example and constraint $\Box(a_f \rightarrow \Diamond b_i)$. Lifecycles $a_1 = \langle a_{start}, a_{complete} \rangle$ and $a_2 = \langle a_{assign}, a_{start}, a_{complete} \rangle$ in trace $\mathbf{t}_1$ correspond to fulfillments for the considered constraint since they are followed by an occurrence of $b_i$. Lifecycles $a_3 = \{\langle a_{start}, a_{suspend}, a_{resume}, a_{complete} \rangle$, $a_4 = \langle a_{assign}, a_{start}, a_{suspend}, a_{resume}, a_{complete} \rangle$, and $a_5 = \langle a_{assign}, a_{reassign}, a_{start}, a_{complete} \rangle$ in trace $\mathbf{t}_2$ correspond to violations. Therefore, in this case, we have $L_{ful} = \{\langle a_{start}, a_{complete} \rangle, \langle a_{assign}, a_{start}, a_{complete} \rangle\}$ and $L_{viol} = \{\langle a_{start}, a_{suspend}, a_{resume}, a_{complete} \rangle, \langle a_{assign}, a_{start}, a_{suspend}, a_{resume}, a_{complete} \rangle, \langle a_{assign}, a_{reassign}, a_{start}, a_{complete} \rangle\}$.

From these sets, we could learn, for example, that it is likely that $\Box(a_f \rightarrow \Diamond b_i)$ is verified when $a_{start}$ is immediately followed by $a_{complete}$ but is not immediately

preceded by $a_{reassign}$ in the lifecycle of $a$. In particular, we exploit decision tree learning in order to identify the conditions (*decision rules*) on the lifecycle of an activation given the training sets $L_{ful}$ and $L_{viol}$.

Each lifecycle of an activation $a$ is encoded in terms of a set of Declare constraints (over the states of the lifecycle). In particular, each lifecycle is encoded as a vector of (boolean) values representing whether all these (intra-lifecycle) constraints are satisfied or not on that lifecycle. The decision tree is trained using the intra-lifecycle conditions as features and the classification of the lifecycle as part of $L_{ful}$ or $L_{viol}$. The analysis of the tree allows us to retrieve the set of conditions on the features which possibly make the constraint under consideration fulfilled or violated. Figure 4 shows a possible decision tree generated over the lifecycle of the running example described above. In order to have the inter-lifecycle constraint $\Box(a_f \rightarrow \Diamond b_i)$ fulfilled, the intra-lifecycle conditions $\Box(a_{start} \rightarrow \bigcirc a_{complete})$ (chain response between $a_{start}$ and $a_{complete}$) and $\neg\Box(\bigcirc a_{reassign} \rightarrow a_{start})$ (chain precedence between $a_{reassign}$ and $a_{start}$) should hold. In summary, the result of the decision tree learning will be a set of Declare constraints over the states of the lifecycles of $a$ which would bring to fulfillments or to violations for $\Box(a_f \rightarrow \Diamond b_i)$.

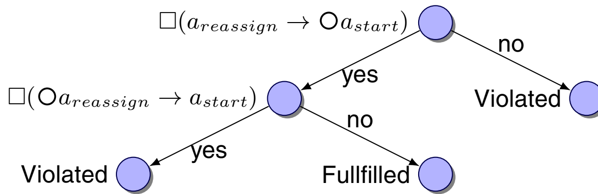


**Fig. 4.** A decision tree generated over the lifecycle of the running example

# 4 Experiments

In order to evaluate the proposed approach, we have implemented it as a plug-in of the process mining tool ProM. Then, the implemented plug-in has been applied to (i) a set of execution logs synthetically generated (to verify its capability to capture known discriminating behaviors); (ii) a real-life log (to check the scalability and the applicability of the approach to real-life settings). All the experiments have been conducted on a machine with an Intel i7 processor (limiting the execution to just one core), 8 GB of RAM and the Oracle Java virtual machine installed on a GNU/Linux Ubuntu operating system and are discussed in this section.

## 4.1 Synthetic Log Analysis

The purpose of the synthetic log analysis is to verify whether the discriminative rules discovered reflect the actual discriminating behaviors (with respect to constraint fulfillments/violations) of the execution logs under examination. To this aim, four synthetic logs have been generated, taking inspiration from the insurance claim process presented in [4]. The process describes the handling of health insurance claims in a travel agency,

starting from the registration up to the claim archiving. The approach has been applied to each synthetic log and discriminative rules have been extracted. In the following subsections, we illustrate how we have generated the logs and the results obtained.

**Synthetic Log 1.** Synthetic Log 1 contains 1000 traces in which $Register$ occurs with one of the following possible lifecycles:

- $\langle\, Register_{start}, Register_{complete}\,\rangle$
- $\langle\, Register_{abort}\,\rangle$
- $\langle\, Register_{assign}, Register_{start}, Register_{complete}\,\rangle$
- $\langle\, Register_{start}, Register_{abort}\,\rangle$
- $\langle\, Register_{start}, Register_{suspend}, Register_{abort}\,\rangle$

Whenever $Register$ is aborted (see second, fourth and fifth lifecycle in the list), the claimer is notified via phone; on the other hand, if the registration completes normally (see first and third lifecycle in the list), the e-mail notification is required. Therefore, in the log, whenever *Register* is aborted, the response constraint $\Box(Register \rightarrow \Diamond Notify_by_phone)$ is verified, otherwise this constraint does not hold.

**Synthetic Log 2.** Synthetic Log 2 contains 1000 traces in which the non-atomic activity *Send_questionnaire* occurs with one of the following possible lifecycles:

- $\langle\, Send_questionnaire_{start}, Send_questionnaire_{complete}\rangle$
- $\langle\, Send_questionnaire_{withdraw}\,\rangle$
- $\langle\, Send_questionnaire_{assign}, Send_questionnaire_{withdraw}\,\rangle$
- $\langle\, Send_questionnaire_{start}, Send_questionnaire_{suspend}, Send_questionnaire_{abort}\,\rangle$
- $\langle\, Send_questionnaire_{complete}\,\rangle$

When *Send_questionnaire* is withdrawn or aborted (see second, third and fourth lifecycle in the list), *Skip_response* for skipping the response is executed. On the other hand, whenever *Send_questionnaire* completes normally (see first and fifth lifecycle in the list), *Skip_response* is not executed. Therefore, in the log, the response constraint $\Box(Send_questionnaire \rightarrow \Diamond(Skip_response))$ is verified only if *Send_questionnaire* does not complete normally (i.e., with withdraw or abort).

**Synthetic Log 3.** Synthetic Log 3 contains 1000 traces in which the non-atomic activity *High_medical_history_check* occurs with one of the following possible lifecycles:

- $\langle\, High_medical_history_check_{withdraw}\,\rangle$
- $\langle\, High_medical_history_check_{start}, High_medical_history_check_{complete}\rangle$
- $\langle\, High_medical_history_check_{start}, High_medical_history_check_{abort}\rangle$
- $\langle\, High_medical_history_check_{assign}, High_medical_history_check_{autoskip}\rangle$
- $\langle\, High_medical_history_check_{assign}, High_medical_history_check_{start}, High_medical_history_check_{complete}\,\rangle$

When *High_medical_history_check* does not complete normally (see first, third and fourth in the list), *Contact_hospital* is executed eventually. On the other hand, in cases in which the verification procedure completes normally (see second and fifth lifecycle in the list), there is no need to contact the hospital. Therefore, in the log, the response constraint $\Box(High_medical_history_check \rightarrow \Diamond Contact_hospital)$ holds if and only if *High_medical_history_check* fails.

**Table 3.** Synthetic Log Results

| LOG | INTER-LIFECYCLE RELATION | INTRA-LIFECYCLE RELATION |
|---|---|---|
| 1 | response($Register$,$Notify_by_phone$) | exactly (1,$Register_{abort}$) |
| 2 | response ($Send_questionnaire$, $Skip_response$) | exclusive choice ($Send_questionnaire_{abort}$, $Send_questionnaire_{withdraw}$) |
| 3 | response ($High_medical_history_check$, $Contact_hospital$) | $\neg$ alternate response ($High_medical_history_check_{start}$, $High_medical_history_check_{complete}$) |
| 4 | response($Register$, $Notify_by_email$) | alternate succession ($Register_{resume}$,$Register_{complete}$) |

**Synthetic Log 4.** Synthetic Log 4 contains 1000 traces in which the non-atomic activity *Register* occurs with one of the following possible lifecycles:

- $\langle\ Register_{start}, Register_{suspend}, Register_{resume}, Register_{complete}\ \rangle$
- $\langle\ Register_{start}, Register_{suspend}, Register_{abort}\ \rangle$
- $\langle\ Register_{start}, Register_{suspend}, Register_{resume}, Register_{suspend}, Register_{resume}, Register_{complete} \rangle$
- $\langle\ Register_{start}, Register_{suspend}, Register_{resume}, Register_{suspend}, Register_{withdraw}\ \rangle$

When *Register* is suspended and ends with an abort or a withdraw (see second and fourth lifecycle in the list), or *Register* is suspended (and resumed) more than once (see third lifecycle in the list), the claimer has to be notified via phone; if there is only one suspension correctly resumed, i.e., a single cycle suspend/resume and, eventually, a normal completion (see first lifecycle in the list), the claimer can be notified via e-mail. Therefore, in the log, whenever $Register$ is suspended and not resumed, or suspended more than once, the response constraint $\Box(Register \rightarrow \Diamond Notify_by_phone)$ is verified, otherwise this constraint does not hold.

**Discussion of the Results.** Table 3 shows some of the constraints discovered. For Synthetic Log 1 and the response constraint between *Register* and *Notify_by_phone* the discriminative rule discovered is exactly(1, $Register_{abort}$). Whenever $Register_{abort}$ occurs (exactly once, since it cannot occur more than once based on the XES transactional model), the claimer has to be notified via phone. This result confirms the rationale behind the construction of Synthetic Log 1: whenever a registration is aborted, the claimer has to be notified via phone, otherwise *Notify_by_phone* is not executed.

Concerning Synthetic Log 2 and the response constraint between *Send_questionnaire* and *Skip_response*, the exclusive choice between $Send_questionnaire_{abort}$ and $Send_questionnaire_{withdraw}$ is discovered as discriminative rule. If and only if the lifecycle of *Send_questionnaire* contains either $Send_questionnaire_{abort}$ or $Send_questionnaire_{withdraw}$, the questionnaire response is skipped. This perfectly fits with the behavior characterizing the log: whenever *Send_questionnaire* cannot complete normally, the questionnaire response is skipped.

The response constraint between *High_medical_history_check* and *Contact_hospital* in Synthetic Log 3 is valid when the intra-lifecycle relation $\neg$ alternate response between $High_medical_history_check_{start}$ and $High_medical_history_check_{complete}$ holds. If and only if $High_medical_history_check_{start}$ is not followed by

**Table 4.** BPI 2013 Results

| | INTER-LIFECYCLE RELATION | INTRA-LIFECYCLE RELATION | CLASS PROB. | SUPPORT |
|---|---|---|---|---|
| (1) | precedence ($Accepted$,$Completed$) | not responded existence ($Completed_{Cancelled}$,$Completed_{Closed}$) | 0.65 | 3711 |
| (2) | precedence ($Queued$,$Accepted$) | init($Accepted_{Assigned}$) $\vee$ init ($Accepted_{Wait_implementation}$) | 0.75 | 92 |
| (3) | precedence ($Queued$, $Completed$) | co-existence ($Completed_{In_call}$,$Completed_{Cancelled}$) | 0.8 | 4551 |
| (4) | response ($Completed$, $Queued$) | $\neg$ responded existence ($Completed_{Resolved}$,$Completed_{Closed}$) | 0.98 | 7570 |
| (5) | responded existence ($Completed$, $Accepted$) | not responded existence ($Completed_{Cancelled}$,$Completed_{Closed}$) | 0.66 | 3771 |
| (6) | responded existence ($Completed$, $Queued$) | co-existence ($Completed_{In_call}$,$Completed_{Cancelled}$) | 0.8 | 4595 |

*High_medical_history_check*$_{complete}$, the hospital has to be contacted. The result is in line with what described in Synthetic Log 3: whenever, in the lifecycle of *High_medical_history_check*, there is a *High_medical_history_check*$_{start}$ that is not followed by *High_medical_history_check*$_{complete}$, the hospital has to be contacted.

Finally, for Synthetic Log 4 and the response constraint between $Register$ and $Notify_by_email$, the discriminative rule discovered for the verification of this constraint is the alternate succession between $Register_{resume}$ and $Register_{complete}$. If and only if $Register_{resume}$ is followed by $Register_{complete}$ and not more than one $Register_{resume}$ is executed before $Register_{complete}$ (at most one suspend/resume cycle occurs), then the response constraint between *Register* and *Notify_by_email* is verified. This result is in line with the behavior injected into Synthetic Log 4: whenever $Register$ is suspended and and correctly resumed at most once, the customer is notified via e-mail.

### 4.2 BPI Challenge 2013

The proposed approach has also been applied to a real-life log. The log, which was provided for the BPI Challenge 2013 [17], has been taken from an incident and problem management system called VINST. The VINST system includes the activities required to diagnose the root causes of incidents and to secure the resolution of those problems ensuring high levels of service quality and availability of services operated by Volvo IT. The log contains 7,554 cases and 65,533 events and is characterized by four different activities (*Accepted*, *Completed*, *Queued* and *Unmatched*) and 14 event types like *In_Call*, *Assigned*, *Cancelled*, *Resolved* and *Closed*. The transactional model for activity lifecycles followed in the log is reported in Figure 1 on the right hand side.

Table 4 shows a list of discovered constraints and, for each of them, the (intra-lifecycle) rules to discriminate between fulfillments and violations. The last two columns of the table also report class probability and support associated to each discovered discriminative rule.

The first row of the table (1) suggests that a possible discriminating behavior for which *Completed* is preceded by *Accepted* is that either *Completed*$_{Cancelled}$ or *Completed*$_{Closed}$ occurs in the lifecycle of *Completed*. The same rule is also discriminating

for fulfillments/violations of the responded existence between *Completed* and *Accepted* (i.e., the constraint assessing that whenever $Completed$ occurs, then, *Accepted* has to occur in the future or has already occurred before). In the second row of Table 4 (2), we can see that whenever the lifecycle of *Accepted* starts with $Accepted_{Assigned}$ or with $Accepted_{Wait_Implementation}$, *Accepted* is preceded by *Queued*. These results are the ones with the lowest class probability and support. For example, the class probability of the rule in (1) (0.65) indicates that only in 65% of the cases in which the constraint is actually verified, the corresponding discriminative rule also holds. Similarly, the support of (2) indicates that the cases in which the constraint and the corresponding discovered discriminative rule are both verified are 92.

On the other hand, the remaining rules present both a reasonable class probability ($> 0.8$) and a good support ($> 4500$). In particular, the co-occurrence of $Completed_{In_Call}$ and $Completed_{Cancelled}$ discriminates both on the precedence (3) and on the responded existence (6) between *Queued* and *Completed*, i.e., whenever both *In_Call* and *Cancelled* occur in the lifecycle of *Completed*, it means that *Completed* is preceded by *Queued* or, more in general (with a slightly higher support), that *Queued* occurs at least once before or after $Completed$.

Finally, the discovered discriminating behavior with the highest class probability (almost 1) and support (more than 7000) is the one related to the response constraint between *Completed* and *Queued* (6): *Queued* eventually follows $Completed$ if and only if only one among $Completed_{Resolved}$ and $Completed_{Closed}$ occurs in the lifecycle of *Completed*.

## 5 Conclusion and Future Work

This paper presents a novel approach for the discovery of declarative process models from logs containing non-atomic activities. Discriminative rule mining is used to characterize the lifecycle of each constraint activation and discriminate between lifecycles that ensure that the activation is a fulfillment and lifecycles that correspond to violations of the constraint under examination.

In order to assess the applicability of the proposed approach, we applied it to four synthetic logs and a real-life log recorded by an incident and problem management system in use at Volvo IT Belgium. Our experiments show the effectiveness of the approach and its applicability in real-life scenarios. As future work, we will conduct a wider experimentation of the proposed framework on several case studies in real-life scenarios and different transactional models for activity lifecycles. In addition, we will implement the identifications of lifecycles through event correlations and compare this approach with the FIFO-based approach presented in this paper.

**Acknowledgment.** This research has been carried out with the valuable comments and support of Andrea Burattin from University of Padua.

## References

1. Van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D, 99–113 (2009)

2. Bernardi, M.L., Cimitile, M., Lucca, G.A.D., Maggi, F.M.: Using declarative workflow languages to develop process-centric web applications. In: 16th IEEE International Enterprise Distributed Object Computing Conference Workshops, EDOC Workshops, Beijing, China, September 10-14, pp. 56–65 (2012)
3. Bose, R.P.J.C., Maggi, F.M., van der Aalst, W.M.P.: Enhancing declare maps based on event correlations. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 97–112. Springer, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40176-3_9
4. Bose, R.J.C.: Process Mining in the Large: Preprocessing, Discovery, and Diagnostics. Ph.D. thesis, Eindhoven University of Technology (2012)
5. Burattin, A., Maggi, F., van der Aalst, W., Sperduti, A.: Techniques for a Posteriori Analysis of Declarative Processes. In: EDOC, pp. 41–50 (2012)
6. Burattin, A., Sperduti, A.: Heuristics Miner for Time Intervals. In: European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium (2010)
7. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, pp. 161–168. ACM, New York (2006), http://doi.acm.org/10.1145/1143844.1143865
8. Cheng, H., Yan, X., Han, J., Yu, P.S.: Direct discriminative pattern mining for effective classification. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE 2008, pp. 169–178. IEEE Computer Society, Washington, DC (2008), http://dx.doi.org/10.1109/ICDE.2008.4497425
9. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems 2, 278–295 (2009)
10. Günther, C.W.: XES Standard Definition (2009), www.xes-standard.org, http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf
11. Kupferman, O., Vardi, M.: Vacuity Detection in Temporal Model Checking. Int. Journal on Software Tools for Technology Transfer, 224–233 (2003)
12. Lo, D., Cheng, H.: Lucia: Mining closed discriminative dyadic sequential patterns. In: Proc. of the International Conference on Extending Database Technology (EDBT), pp. 21–32. Springer (2011)
13. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
14. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1) (2010)
15. Pesic, M., Schonenberg, M.H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC, pp. 287–300 (2007)
16. Quinlan, J.R.: C4.5: Programs for Machine Learning. M. Kaufmann Publishers Inc. (1993)
17. Steeman, W.: Bpi challenge 2013, incidents (2013), http://dx.doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee
18. Sun, C., Du, J., Chen, N., Khoo, S.C., Yang, Y.: Mining explicit rules for software process evaluation. In: Proceedings of the 2013 International Conference on Software and System Process, ICSSP 2013, pp. 118–125. ACM, New York (2013), http://doi.acm.org/10.1145/2486046.2486067
19. Verbeek, E.H.M.W., Buijs, J., van Dongen, B., van der Aalst, W.M.P.: ProM 6: The Process Mining Toolkit. In: BPM 2010 Demo, pp. 34–39 (2010)