Noname manuscript No.
(will be inserted by the editor)

Generating synthetic positive and negative business process traces through abduction

Daniela Loreti · Federico Chesani ·
Anna Ciampolini · Paola Mello

Received: date / Accepted: date

Abstract As recent years have seen the rise of a new discipline commonly addressed as *process mining*, focused on the management of business processes, two tasks have gained increasing attention in research: *process discovery* and *compliance monitoring*. In both these fields, the demand for event log benchmarks with predefined characteristics has determined the design of various methodologies and tools for synthetic log generation. However, artificially created as well as real-life logs often contain *positive* examples only (i.e., process instances deemed as compliant w.r.t. the model), while the presence of *negative* process instances (i.e., non-compliant traces) can be crucial to correctly evaluate the performance and robustness of a novel process discovery or conformance checking technique.

In this work, we investigate positive and negative trace generation in case of both declarative and procedural model specifications and we present our abduction-based approach to log synthesis. The theoretical study is concretely applied in a software prototype for log generation, which takes as input a declarative or structured workflow model, and emits logs containing positive and negative traces. The approach provides both a highly expressive notation for the description of the business model, and the ability to generate logs with various customizable features. The final comparative study of other existing log generators reveals several advantages of the proposed approach and draws the direction of future improvements.

D. Loreti
CIRI - Health Sciences & Technologies. Via Tolara di Sopra, 41/E,
Ozzano dell'Emilia (BO), Italy
Tel.: +39-051-2093275
E-mail: daniela.loreti@unibo.it

F. Chesani · A. Ciampolini · P. Mello
DISI - Department of Computer Science and Engineering, University of Bologna.
Viale del Risorgimento 2, Bologna, Italy
E-mail: federico.chesani@unibo.it, anna.ciampolini@unibo.it, paola.mello@unibo.it

Keywords Synthetic Log Generation · Declarative and Procedural Business Model · Artificially Positive and Negative Events · Abductive Reasoning

1 Introduction

Over the last years, we have assisted to the rising of a novel research field called *process mining* [1], dealing with the analysis of business processes starting from a log. The increasing interest in process mining drops into the wide-ranging framework of Business Process Management (BPM), a discipline dealing with the study and control of process execution. Especially, the tasks of learning a model from a log (commonly addressed as *process discovery*) and verifying the conformance of a log w.r.t. a model (i.e., *compliance monitoring* or *conformance checking*) have brought significant advantages to business processes, thus gaining increasing attention in industrial and academic research.

For both process discovery and compliance monitoring, the availability of the input log is a fundamental requirement. Unfortunately, in a number of real contexts such log might not be present: for example, a newly designed process might have a model, but it might lack the execution logs (due to its novelty). Even when the log is available, its quality might be a major concern, as incompleteness or noise can heavily affect both mining and conformance checking results. Furthermore, in order to accurately compare novel process discovery and conformance checking approaches with existing techniques, it is extremely important to relay on a solid benchmark suite composed of logs with specific, known characteristics. For these reasons, a common approach to evaluate process mining techniques is based on the use of synthetic logs created via process simulation. Log generators are software tools that take as input a model and a set of desired features and emit artificial logs. The generators differ in the supported model language and in the log features that can be requested. A small number of these software [33, 16, 17] can also produce negative examples, i.e., sequences of activities representing process instances that diverge from the expected behavior. For example, executions that are not compliant with specific time or resource constraints or lack some events in a sequence. Indeed, the availability of negative examples in the log is very important when evaluating the performance of different process mining techniques and their robustness to noise. Real life logs might contain also negative examples, but the information about *which* among all the instances are positive/negative is often unavailable. Obviously, manually annotating logs is viable only for small data sets.

The cited approaches [33, 16, 17] to log generation with the ability to produce negative examples are intrinsically procedural, and therefore not suitable for the evaluation of process discovery techniques based on declarative process models. As pointed out in [5, 47], differently from procedural models (which work in a closed world assumption where all the allowed behaviors need to be explicitly specified), declarative models are open, thus enjoying a flexibility that makes them more suitable to describe highly variable behaviors in a compact way.

In previous works [22, 24], we adopted an Artificial Intelligence technique, namely abduction, to complete partial logs, and introduced a (novel) notion of *weak compliance* of incomplete process instances w.r.t. a procedural model. The focus was on determining if incomplete logs (i.e., reporting process instances with missing or partially specified events) might be considered compliant with a model, and under which conditions/hypotheses. To that end, we exploited Social Constrained IFF (SCIFF) [12], an Abductive Logic Programming (ALP) framework, and in particular the hypotheses-making reasoning capabilities typical of abduction.

In this work, we investigate the link between abduction and process mining by exploiting the SCIFF framework for a different purpose: the generation of synthetic logs according to predefined custom-selectable properties. Thanks to its highly expressive notation, SCIFF is able to operate with both procedural and declarative formalisms to express the behavioral model. The resulting tool provides logs with several interesting features. In particular, synthetic negative examples can be generated, which are not compliant with the original input model according to user-defined properties, such as the absence of some events, the presence of events in non-compliant sequences, with wrong timing, duration or resource values, etc. While the previous paper [21] contains some preliminary ideas about the design of a synthetic log generator for positive examples only, the contributions of this work are manifold:

– a study of the theoretical aspects of positive and negative log generation in both open declarative and closed procedural environments;
– a detailed description of our log generation methodology as well as underlying foundations of the proposed technique;
– a standalone log generation prototype available for download together with an evaluation of its performance on an average hardware architecture;
– a study of the existing approaches to log generation and a discussion on the advantages and shortcoming of each one when compared with the proposed methodology.

2 Preliminaries

A key concept in the field of BPM is that of *event log*: a collection of observed (i.e., logged) executions of a business process, in terms of all the occurred *events*. Each event in the log refers to an *activity* i.e., a well-defined step in the business process, and is related to a specific process *instance*. The logged description of a process instance in terms of all its constituent activities is addressed as *trace* or *case*. In this regard, a storing standard has also been developed to clearly define a common way to exchange and analyse event logs: eXtensible Event Stream (XES) [55]. In order to reason upon the business execution, a *model* of the process is often employed. It is intended as the set of relevant activities that may occur in the observed environment and the constraints on them. This model must be specified through some shared language in order to be understood by different experts from the business domain. The languages to express the business model are often classified according to their procedural or declarative nature. Petri nets [36] for instance, are a widely used example of a *procedural* modelling language. As such, they tend to represent the model as a flow of activities that can be executed sequentially, alternatively or in parallel from a beginning to an end. *Declarative* languages on the other hand, specify the constraints that should be satisfied by all the traces, without focusing on the exact paths to be followed. One of the most famous examples of declarative languages is Declare [50,4]. In the following we use the terms *positive* trace to address a logged process instance that fulfil the requirements of the business model, whereas *negative* traces diverge from the expected behaviour, thus being non-compliant w.r.t. the model.

Our approach to log generation involves the concept of *abduction* [40]: a general, logic-based technique for making hypotheses, originally thought as a mechanism for providing explanations given some observations and some rules/constraints (which capture the domain knowledge). The set of hypotheses that explain the observations is usually called *abductive explanation.* This form of reasoning is the basis of the SCIFF framework [12] employed here. SCIFF provides a language based on ALP for expressing the domain knowledge, together with a proof procedure supporting the abductive reasoning process. It was initially developed with the goal of run-time checking the compliance of the observed system behaviour w.r.t. a given model. To this end, SCIFF extends the abductive reasoning with a few fundamental compliance-related concepts, such as observed events, expected behaviour (in terms of expected events) and prohibitions, and the formal notions of compliance and violation of traces against expectations.

A relevant point for this work, is the SCIFF's ability to deal with both *ground* traces and *templates* (or *non-ground* traces). A trace is indeed a container for various information. For example, consider an activity addressing the measurement of the temperature D in a room. The trace containing such event is likely to report the observed value for D and the timestamp T of the measurement. These data can be seen as variables assuming different values. A trace is said to be ground if all the data in it are bounded to a value. A template is instead a trace containing some variables e.g., generic indications of D and T instead of their effective values. Templates can possibly contain constraints restricting the variable domains. As such, they are a powerful abstraction to represent sets of traces in a compact way.

3 Motivations for positive and negative log generation

The availability of an event log and the presence of positive and negative examples in it is very important for the validation of conformance checking and process discovery techniques. Indeed, as conformance checking algorithms aim to identify the traces that are not compliant with a predefined model, the availability of a log containing both positive and negative examples allows the developer to evaluate the performance of a certain technique, for example, in terms of the number of false positive and false negative results obtained or required time for computation. Since in real-life logs the information about which traces diverge from the expected behaviour is usually unavailable, the main process discovery techniques are limited to the harder setting of unsupervised learning with positive examples only. The generation of artificial logs with both compliant and non-compliant traces could foster the adoption of more effective supervised techniques. For example, some machine learning schemes based on Inductive Logic Programming (ILP) [49] relay on a set of negative as well as positive examples in order to extract a formal description of the business process model from an event log.

Furthermore, in the field of process discovery, the presence of negative events allows the developer to verify the robustness of its approach. For example, suppose to have a log generator able to take a process model as input and produce a log with a mixed composition, reporting both positive and negative traces in predefined percentages, according to configurable parameters. The emitted event log can be used as input to the process discovery algorithm to test its performance by comparing the extracted process model with the original one. Varying the percentage of negative examples in the log can give an evaluation of the robustness of the process discovery algorithm to noise and incompleteness.

In the field of business process, a strong concern is related to the accuracy of the model i.e., its ability to capture allowed behaviours and to highlight unwanted actions. For this purpose, a domain expert is often involved but, when the model is composed of several actions, constraints, and possibly concurrent paths, the analysis of accuracy is far from straightforward. For this reason, when the process model is given, the generation of synthetic traces can be useful to provide domain experts with a collection of examples to clearly explain which process instances are allowed or not allowed by the input model. Thus, to obtain a quicker feedback about accuracy.

Synthetic logs can be employed also for the conversion of models from a declarative to a procedural formalization (and vice-versa). For example, if the business model of a process is provided by means of declarative constraints – e.g., through a Declare formalization –, the synthetic traces produced through log generation can be used as input to a process miner, like for example the α-algorithm [6], to extract a procedural version of the original model (in the mentioned miner, a Petri-net model).

Another interesting application is related to the planning problem. Indeed, when the model is understood, the generated positive traces are examples of process executions compliant with a certain set of constraints and can therefore be used for production purposes. For example, consider a line of production that must complete its job in 10 minutes. If the execution model is known, positive trace generation can suggest which are the execution path of the production line that can fulfil the requirement on the total production time.

Furthermore, being able to emit both ground and non-ground traces, abduction is a perfect candidate to suggest which are the sets of cases that fulfil the constraints, thus to tackle the planning problem. It is interesting to note that abduction has been investigated in the past for supporting the planning task [52], and that in turn planning itself has been recently proposed for generation of trace templates and instances [46].

Finally, when the business model is complex, involving several different actions, and including constraints on time and resources, even the evaluation of model consistency (i.e., determine if at least one trace can satisfy all the constraints), which may appear a simple operation in general, can be very difficult for a human being. Log generation allows to suddenly identify situations in which no trace can be compliant with the model. In that cases indeed, as the intersection of the sets of traces satisfying each constraint is empty, the generation would not produce any output.

4 On different modelling approaches and how they affect log generation

Within the BPM research field, a number of different formalisms and languages have been proposed for the definition of business processes. Different aspects of the business process itself have been researched, thus leading to the emergence of several proposals, each one with merits and limits. Providing a complete and detailed view of all these approaches is out of the scope of this work. However, in the following we introduce the few aspects that we deem fundamental to comprehend the issues related to positive and negative trace generation.

4.1 Procedural vs. declarative and open vs. closed modelling approaches

A first important aspect of log generation is how to define the *flow-related* aspects of a business process. On one end of the spectrum, there are the *procedural* approaches, where the process flow is defined as a number of steps from a beginning to an end. Usually, two artificial activities, the start and the stop are employed. They are not meant to address real events, but just to support the concepts of beginning and conclusion of a process instance. Typical constructs of procedural languages provide notions such as *sequence* of activities, *or–split/join* (alternative, non-exclusive choices of different flow paths), *xor–split/join*, and *and–split/join* (parallel execution of different flow paths). Fig. 1 shows a simple example of a procedural workflow defined using the YAWL graphical language [3], where after the start, activity a should be executed, then b or {c, d} should be executed (exclusive choice between the two paths), and finally activity e should be executed.

On the other end of the spectrum, there are *declarative* approaches, which focus more on the constraints that should be satisfied by all the process executions, rather than fixing exact flows/paths. In Fig. 2a a Declare [50] constraint

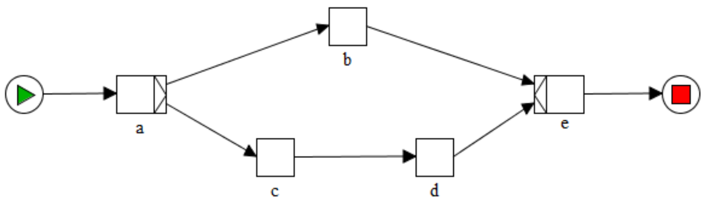


Fig. 1: A simple business process defined through the procedural language YAWL.

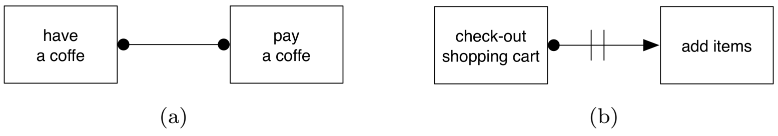


Fig. 2: Two simple examples of Declare constraints.

is placed over the two activities of having a coffee and paying a coffee: the intended meaning is that it does not matter the order (notice the absence of arrows) of execution, but if you have a coffee then you must pay for it, and vice-versa, if you pay for a coffee you must have it. In Fig. 2b another constraint is shown: after the check-out of a shopping cart, the user is not allowed to add more items. The two small vertical lines in the connection between the two activities stand for a prohibition, while the presence of an arrow sets also a temporal ordering.

Another important aspect regards the degree of openness allowed by the modelling approach. *Closed* models are characterized by the fact that only the specified activities, at the specified time instant can be executed. For example, let us consider again the model depicted in Fig. 1: a closed modelling approach would implies that only the envisaged activities (namely, {a,b,c,d,e}) are allowed to be executed, and any other activity is prohibited. Such approaches are typical of a number of applicative domains, like for example, security communication protocols or bank protocols for financial transactions, where even the smallest bit of information that is not envisaged should trigger alarms and invalidate the interaction.

At the opposite side, *open* approaches are characterized by the fact that they specify both activities that should be executed, and activities that should not be executed (i.e., *prohibitions*). When nothing is specified about an activity, the usual intended meaning is that the appearance of such activity in a case does not influence the trace compliance (or non-compliance) w.r.t. the model. As an example, let us consider the process depicted in Fig. 2a: within having a coffee and paying a coffee, any other activity such as chat with the barman is allowed as well. Theoretically, open approaches allow for the execution of *any* activity for which no prohibition has been expressed. However, for practical reasons, many approaches allow for the execution of *any (non-forbidden)* *activity within a specified set*: in other words, the set of allowed activities is defined a-priori and *finite*. This paper adopts the same view: we will generate only traces whose activities belong to a user-defined finite set $\mathbb{A}$.

4.1.1 Relation between two orthogonal dimensions

A noteworthy consideration regards the relation between these orthogonal dimensions: procedural vs. declarative models and open vs. closed approaches. A number of works have identified procedural models as closed, and declarative models as open. This association is by no means mandatory. Indeed, a declarative approach can be used to model a closed process if we provide a set of additional constraints to ensure that only the specified activities are allowed, and any other is prohibited. As well as a procedural formalism can be used to express an open model by relaxing some constraints.

Although our approach remains general and can be applied to all the cases, for convenience's sake, in this paper we focus on the most popular combination of the orthogonal dimensions i.e., procedural closed models and declarative open ones. The interested reader can refer to [47] for an in-depth discussion on closed vs. open models.

4.2 Positive and negative traces w.r.t. declarative open process models

The open/closed nature of the model affects the reasoning on *positive* and *negative* traces. In an open model, if nothing is said about a certain activity X, this means, for example, that any positive trace, containing or not containing X is compliant anyway with the process model. This observation leads to two further considerations:

1. one might be tempted to ignore traces containing activities like X. However, if X is not in the model (i.e., it does not appear in any constraint), but a process designer addresses it as an activity that could happen, he might be interested in observing both traces containing and not containing X;
2. the number of traces that are compliant with an open declarative process model is potentially *infinite* w.r.t. activities like X.

To address the issues above, we restrict the generation to traces containing only events belonging to a specified finite set $\mathbb{A}$ of activities. For example, let us consider the process model shown in Fig. 2a. For the sake of the example, we could say:

$$\mathbb{A} = \{\mathsf{have\ a\ coffee}, \mathsf{pay\ a\ coffee}, \mathsf{chat\ with\ the\ barman}\}$$

where the activity chat with the barman is not subject to any constraint. In this case, a subset of all the positive traces would be the following one[1]:

$$
\begin{aligned}
&\tau_1 = \emptyset\\
&\tau_2 = [\textsf{have a coffee}, \textsf{pay a coffee}]\\
&\tau_3 = [\textsf{chat with the barman}, \textsf{have a coffee}, \textsf{pay a coffee}]\\
&\tau_4 = [\textsf{have a coffee}, \textsf{chat with the barman}, \textsf{pay a coffee}]\\
&\tau_5 = [\textsf{have a coffee}, \textsf{pay a coffee}, \textsf{chat with the barman}]\\
&\tau_6 = [\textsf{pay a coffee}, \textsf{have a coffee}]\\
&\ldots
\end{aligned}
$$

Negative traces are those ones that violate one or more constraints of the process model. If we consider again the example shown in Fig. 2a, negative traces are those that violate the only constraint present: they should contain pay a coffee without have a coffee or should contain have a coffee without pay a coffee. A subset of all the negative traces would be the following one:

$$
\begin{aligned}
\overline{\tau_7} &= [\textsf{have a coffee}]\\
\overline{\tau_8} &= [\textsf{pay a coffee}]\\
\overline{\tau_9} &= [\textsf{chat with the barman, have a coffee}]\\
\overline{\tau_{10}} &= [\textsf{have a coffee, chat with the barman}]\\
\overline{\tau_{11}} &= [\textsf{chat with the barman}, \textsf{pay a coffee}]\\
\overline{\tau_{12}} &= [\textsf{pay a coffee, chat with the barman}]\\
&\ldots
\end{aligned}
$$

where the overline on the trace symbol τ_i indicates that the trace violates the process model.

4.3 Positive and negative traces w.r.t. procedural closed process models

Positive traces w.r.t. procedural models are only those that are allowed and explicitly considered by the model. Intuitively, it suffices to walk the structure (the graph) of the process model, to get instances of positive traces. For example, if we consider the process model shown in Fig. 1, only two positive traces are allowed:

$$
\begin{aligned}
\tau_{13} &= [\textsf{a}, \textsf{b}, \textsf{e}]\\
\tau_{14} &= [\textsf{a}, \textsf{c}, \textsf{d}, \textsf{e}]
\end{aligned}
$$

Similarly to the case of open declarative models, negative traces w.r.t. closed procedural models are those traces that violates one or more constraints.

[1] For the sake of readability, we omit in the following traces the timestamps: the order in which the events are written determines which event comes first.

More in detail, procedural constraints are the flow constructs explicitly defined in the process model, plus the constraint (implicit in the closeness flavour) that anything not considered by the model is strictly prohibited. Thus, negative traces can be grouped into two sets: (*i*) traces that contain only (possibly, a subset of) events explicitly mentioned in the model, and that violate one or more flow structures; and (*ii*) traces that contain unknown events. Referring to Fig. 1, two examples of negative traces that belongs to group (*i*) would be:

$$\begin{aligned}\overline{\tau_{15}} &= [\mathsf{a},\ \mathsf{e}]\\ \overline{\tau_{16}} &= [\mathsf{a},\ \mathsf{d},\ \mathsf{c},\ \mathsf{e}]\\ &\dots\end{aligned}$$

In $\overline{\tau_{15}}$ the activity b is missing, in $\overline{\tau_{16}}$ instead activities c and d are executed in the wrong order. The set of negative traces belonging to group (*ii*) instead is possibly infinite, as possibly infinite is the set of activities not explicitly envisaged by the procedural model. To cope with this aspect, we restrict our approach by considering again a finite set $\mathbb{A}$ of activities that will be used for generating the traces. Referring again to Fig. 1, such a set could be:

$$\mathbb{A} = \{\mathsf{a},\ \mathsf{b},\ \mathsf{c},\ \mathsf{d},\ \mathsf{e},\ \mathsf{f},\ \mathsf{g}\}$$

The process model does not envisages activities $\{\mathsf{f}, \mathsf{g}\}$: any trace containing one or more of these activities is a negative trace. For example:

$$\begin{aligned}\overline{\tau_{17}} &= [\mathsf{a},\ \mathsf{f},\ \mathsf{b},\ \mathsf{e}]\\ \overline{\tau_{18}} &= [\mathsf{a},\ \mathsf{c},\ \mathsf{d},\ \mathsf{e},\ \mathsf{g}]\\ &\dots\end{aligned}$$

5 The SCIFF abductive capabilities

In order to clearly explain our generation approach, we first briefly recall the main concepts behind the SCIFF abductive capability. The interested reader can refer to the work [12] for a complete dissertation on this topic. Formally, a SCIFF specification is a triple $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where:

- $\mathcal{KB}$ is a knowledge base (i.e., a Logic Program as for [44]);
- $\mathcal{A}$ is a set of abducible predicates with functor **ABD**, **E**, or **EN**;
- $\mathcal{IC}$ is a set of Integrity Constraints (ICs).

Among the elements of $\mathcal{A}$, predicates with functor **ABD** correspond to usual abducibles as in ALP [40] i.e., predicates that can be hypothesized. **E** and **EN** are particular abducibles respectively used for modelling *positive expectations* – i.e. expectations about the happening of certain events – and *negative expectations* – i.e. prohibitions about the happening of certain events.

Happened events are represented through predicates with functor **H**. Hence, in SCIFF a trace is a set of predicates $\mathbf{H}(EvDesc, T)$, where each predicate stands for the observation of the happening of an event described by $EvDesc$

at timestamp T. For example, trace τ_2 introduced in Section 4.2 would be represented in SCIFF as:

$$\tau_2 = [\mathbf{H}(\mathsf{have\_a\_coffee}, 5), \mathbf{H}(\mathsf{pay\_a\_coffee}, 8)]$$

meaning that at time instant 5 the event $\mathsf{have\_a\_coffee}$ has been observed, and then, at time instant 8, the event $\mathsf{pay\_a\_coffee}$ has been observed too.

$\mathcal{IC}$ is a set of forward rules of the form *body* → *head*. They are used to dynamically link the observation of the happening of events with positive and negative expectations. Roughly speaking, when *body* becomes true, also *head* must be true. The *body* contains conjunctions of special terms with functors **H**, **E**/**EN** or **ABD**, while the *head* is made of disjunctions of conjunctions of terms with functors **E** or **ABD**. For example, the following IC

$$\mathbf{H}(\mathsf{have\_a\_coffee}, T_a) \rightarrow \mathbf{E}(\mathsf{pay\_a\_coffee}, T_b). \tag{1}$$

states that every time the event $\mathsf{have\_a\_coffee}$ is observed at a timestamp T_a, then an event $\mathsf{pay\_a\_coffee}$ is expected to happen (to be observed) at a timestamp T_b.

The IC shown in (1) already provides a powerful hint of how the SCIFF proof procedure determines if a trace is compliant w.r.t. to a model. Let us suppose a process model is described in terms of (1). Also, let us suppose to observe the event $\mathbf{H}(\mathsf{have\_a\_coffee}, 12)$. The event triggers the IC (1): as a consequence, the positive expectation $\mathbf{E}(\mathsf{pay\_a\_coffee}, T_b)$ is abduced. The SCIFF Proof Procedure then waits for further events. If an event happens, such that it matches with the positive expectation, we say that such expectation is *fulfilled.* If no event matching the expectation happens, the expectation is *violated.* We do not report here the definitions of *compliance* and *violation* of a trace w.r.t. a model: the interested reader can refer to [12]. Rather, we highlight the following: given a process model described with a SCIFF specification, a trace is compliant with (or violates) that specification, if it is compliant with (violates) the positive and negative expectations generated by the ICs.

In this paper, the goal is to generate traces that are compliant to (or violate) a given model. To this end, we recur to a technique similar to that of previous works: we start from a process model represented in terms of ICs. However, instead of using positive and negative expectations, ICs are defined in terms of abducibles only, each abducible representing the hypothesis that an event happens. For example, let us model the procedural process shown in Fig. 1. First of all, the sequence between the start and the execution of activity a would be modelled as:

$$ic_1 : \quad \mathbf{ABD}(\mathsf{start}, T_s) \rightarrow \mathbf{ABD}(\mathsf{a}, T_a) \wedge T_a > T_s.$$

Then, the xor disjunction between b and c would be[2]:

$$ic_2 : \mathbf{ABD}(\mathsf{a}, T_a) \rightarrow \mathbf{ABD}(\mathsf{b}, T_b) \wedge T_b > T_a \vee \mathbf{ABD}(\mathsf{c}, T_c) \wedge T_c > T_a.$$

[2] Although represented with ∨, SCIFF actually performs an exploration of two alternatives only (after a, either b or c is abduced), thus realizing the semantics of a xor gate[12].

Finally, further sequence constraints are needed:

$$\begin{aligned}
ic_3 :&\quad \mathbf{ABD}(\mathsf{b}, T_b) \rightarrow \mathbf{ABD}(\mathsf{e}, T_e) \wedge T_e > T_b.\\
ic_4 :&\quad \mathbf{ABD}(\mathsf{c}, T_c) \rightarrow \mathbf{ABD}(\mathsf{d}, T_d) \wedge T_d > T_c.\\
ic_5 :&\quad \mathbf{ABD}(\mathsf{d}, T_d) \rightarrow \mathbf{ABD}(\mathsf{e}, T_e) \wedge T_e > T_d.\\
ic_6 :&\quad \mathbf{ABD}(\mathsf{e}, T_e) \rightarrow \mathbf{ABD}(\mathsf{stop}, T_s) \wedge T_s > T_e.
\end{aligned}$$

Summing up, the process model in Fig. 1 would be represented by the set $\{ic_1, ic_2, ic_3, ic_4, ic_5, ic_6\}$. Given the start symbol as input, the SCIFF Proof Procedure would produce the following two *trace templates*:

$$\begin{aligned}
\tau_\alpha &= [\mathbf{ABD}(\mathsf{a}, T_a), \mathbf{ABD}(\mathsf{b}, T_b), \mathbf{ABD}(\mathsf{e}, T_e), T_a < T_b < T_e]\\
\tau_\beta &= [\mathbf{ABD}(\mathsf{a}, T_a), \mathbf{ABD}(\mathsf{c}, T_c), \mathbf{ABD}(\mathsf{d}, T_d), \mathbf{ABD}(\mathsf{e}, T_e), T_a < T_c < T_d < T_e]
\end{aligned}$$

where τ_α matches τ_{13} and τ_β matches τ_{14}, both introduced in Section 4.3.

Finally, notice that τ_α and τ_β are what we call trace templates, since the timestamps are not grounded to specific values, but are rather variables that can be assigned to any set of values respecting the inequalities.

6 Generation of a synthetic log through SCIFF

In order to simplify the comprehension of our approach, we first introduce the reader to the generation of positive traces from a high-level perspective. The process consists of tree steps:

1. the process model is properly translated into a SCIFF specification;
2. the SCIFF proof procedure takes as input the process model expressed as a SCIFF specification, and generates the trace templates;
3. trace templates are grounded with values respecting the constraints imposed by the model.

Translation into SCIFF. The translation of the process model into a SCIFF specification depends on the language adopted for the model definition. In previous works we investigated how closed procedural approaches can be easily translated into SCIFF [23,22], even with the support to activity data and temporal constraints [24,26]. In these previous works, the focus was on establishing if a partial trace could be possibly considered as compliant. If not compliant because of missing events, we showed how hypothetical reasoning and abduction could be exploited to determine a possible set of further activities that, once merged with the initial trace, would make it compliant w.r.t. the model. However, what if the partial trace is empty? The approach presented here is an extension of the previous ones, where the starting point is an empty trace, and the SCIFF is queried about the existence of a trace. Differently from [23, 22], here there is no need to verify the compliance, since the initial trace is empty.

Generation of trace templates. Starting from a SCIFF model specification, the proof procedure is queried about the existence of a compliant trace. As also highlighted in [26], thanks to its abductive nature, the same generation process can also be carried out starting from partially specified traces. The SCIFF generates each trace through hypothetical reasoning, i.e. by abducing the events as of the rules given in the specification. At this stage, timestamps and activity data (if present) are indicated as variables thus realizing trace templates. Once a solution (i.e., a first template) is found, the SCIFF is asked to look for another one: iteratively, all the templates are emitted. To ensure the termination of this iterative process, we ask the user to provide in input a *meta*-information about the process: we require that a maximum number of events for each trace is specified. This allows to avoid problems due, for example, to the presence of (indeterministic) loops in the process model.

Grounding the trace templates. Finally, SCIFF substitutes the variables in the templates with all their possible values. Of course, it can happen that a variable have an infinite set of possible values. For example, consider a timestamp of an event whose only constraint is to be greater than zero: there are infinite values that can satisfy such constraint. For this reason, we ask the user to provide in input a *meta*-information about the process: the user must specify the maximum allowed timestamp (the minimum is automatically assumed to be zero). Together with the assumption that timestamps are represented with integer numbers, the maximum timestamp limit ensures the termination of the grounding process.

As regards the generation of negative examples, since in our approach models are represented by means of SCIFF's ICs, a negative trace can be seen as a trace that violates *one or more* ICs. Hence, to generate a negative trace, it suffices to take the initial SCIFF specification, obtain a new specification by negating one or more ICs, and use it to generate traces. The output will be compliant with the model containing negated ICs, thus it will violate the initial model.

First of all, it is important to understand what it means to negate an IC. In SCIFF, ICs are (forward) implications: they are violated when the premises are true, and the consequences are false. In SCIFF, the consequences (named *head*) are disjunction of conjunction of literals and abducibles. Let us focus to a simple example of one IC made of two different conjuncts:

$$\mathbf{ABD}(a, T_a) \rightarrow \mathbf{ABD}(b, T_b) \wedge T_b > T_a. \tag{2}$$

The intended meaning is that the execution of activity a should be followed by the execution of activity b. In other words, every time we observe a, we should observe also b *after*. The head of the ICs contains two conjuncts: the first states that whenever a happens, also b should happen; and the second requires that b should happen after. So, (2) can be violated in two ways:

i) by having a trace containing activity a, and not containing activity b;
ii) by having a trace containing a, and containing b happened *before* a;[3]

[3] Actually, there is another way to violate IC(2): by negating both the conjuncts i.e., having a trace containing a, and not containing b, and b happening *before* a. Clearly, this third option is inconsistent, since it asks for both the happening and the non-happening of b. Hence, in this case no trace can be generated.

The two cases lead to two different trace templates, each one violating the original IC (2) in its own way.

Given this explanation of what IC negation implies, the generation process takes place as follows. The approach starts from a SCIFF specification $\mathcal{S} = \langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$ of a process model, where $\mathcal{IC}$ is the set of one or more ICs ic_i:

$$\mathcal{IC} \equiv \{ic_1, ic_2, \ldots, ic_n\} \tag{3}$$

New sets $\overline{\mathcal{IC}_j}$ are obtained by negating one or more $ic_i, 1 \leq i \leq n$. Notice however that – as explained for (2) – negating a single ic_i can lead to different ICs. Hence for each ic_i we could obtain $\overline{ic_{i,1}}, \ldots, \overline{ic_{i,m_i}}$. For example, let us suppose to negate ic_1, and that this leads to three different ICs. We would get:

$$\begin{aligned}\overline{\mathcal{IC}_1} \equiv& \{\overline{ic_{1,1}}, ic_2, \ldots, ic_n\}\\ \overline{\mathcal{IC}_2} \equiv& \{\overline{ic_{1,2}}, ic_2, \ldots, ic_n\}\\ \overline{\mathcal{IC}_3} \equiv& \{\overline{ic_{1,3}}, ic_2, \ldots, ic_n\}\end{aligned}$$

Then, we could negate ic_2 (suppose this is possible in two different ways):

$$\begin{aligned}\overline{\mathcal{IC}_4} \equiv& \{ic_1, \overline{ic_{2,1}}, \ldots, ic_n\}\\ \overline{\mathcal{IC}_5} \equiv& \{ic_1, \overline{ic_{2,2}}, \ldots, ic_n\}\\ \ldots\end{aligned}$$

The number of models $\overline{\mathcal{IC}_j}$ that can be obtained is given by the cardinality of the power set of $\mathcal{IC}$, multiplied for the cardinality of the power set of all the possible ways of negating one or more ICs. SCIFF specifications $\overline{\mathcal{S}_j} = \langle \mathcal{KB}, \overline{\mathcal{IC}_j}, \mathcal{A} \rangle$ are obtained consequently, and each one is used to generate traces. The number of $\overline{\mathcal{S}_j}$ provided in this way is definitely huge. However, not all those specifications will allow for the existence of traces: indeed, it might happen that by negating two different ICs, an inconsistent model is obtained, thus not allowing the existence of any trace compliant with that model. It might also happen that a trace generated from a $\overline{\mathcal{IC}_j}$ is again compliant with the initial $\mathcal{IC}$. For example, if the initial model admits two alternative paths (i.e., $\mathcal{IC}$ includes a xor constraint) and the specification derived from a $\overline{\mathcal{IC}_j}$ negates the constraints of only one path, the traces generated following the other, unmodified path are again compliant with the initial model. To remove these actually positive traces, after the generation, each trace is again checked for its non-compliance with the original specification $\mathcal{S}$. Algorithm 1 summarizes the steps of negative trace generation.

6.1 Positive and negative trace generation for open declarative processes

Among the several languages proposed for modelling open declarative approaches, we focus on Declare [50]. In previous works [48, 47], we showed how Declare constructs can be mapped in SCIFF while preserving the notion of compliance. Here, the shift of perspective towards log generation motivates

Algorithm 1 Generation of all the traces that are non-compliant with a given process model expressed through a SCIFF specification $\mathcal{S}$.

Input: $\mathcal{S} = \langle \mathcal{KB}, \mathcal{IC}, \mathcal{A} \rangle$, a SCIFF specification.
Output: $\overline{T}$ a set of traces non-compliant with $\mathcal{S}$.

```
1: procedure negativeTraceGeneration($\mathcal{S} = \langle \mathcal{KB}, \mathcal{IC}, \mathcal{A} \rangle$)
2:     $\mathbb{P}(\mathcal{IC})$ = generate the power set of $\mathcal{IC}$
3:     for each $p \in \mathbb{P}(\mathcal{IC})$ do
4:         $neg(p)$ = generate the set of all possible negations of $p$
5:         for each $\overline{p}_j \in neg(p)$ do
6:             $\overline{\mathcal{IC}}_j = \overline{p}_j \cup \{\mathcal{IC} \backslash p\}$
7:             $\overline{\mathcal{S}}_j = \langle \mathcal{KB}, \mathcal{IC}_j, \mathcal{A} \rangle$
8:             $\overline{T}_j$ = generate traces compliant with $\overline{\mathcal{S}}_j$
9:             for each $\overline{t}_j \in \overline{T}_j$ do
10:                if ! $compliant(\overline{t}_j, \mathcal{S})$ then
11:                    $\overline{T} = \overline{T} \cup \overline{t}_j$
12:                end if
13:            end for
14:        end for
15:    end for
16:    return $\overline{T}$
17: end procedure
```

some changes in the way we represent the Declare constructs. Moreover, a further issue is given by the fact that, as explained in Section 4.1, in open declarative models there can be activities that are part of the process but are not part of any constraint. When generating traces, also these activities are interesting and should appear in the log.

6.1.1 Generating positive traces

As regards the first step of log generation i.e., the translation of the Declare model into a SCIFF specification, it stems from the one proposed in [48], which was oriented to compliance monitoring. Differently from [48], here the focus is restricted to the problem of generating traces, thus two variations are present: (*i*) there is no more need for happened events and positive/negative expectations, but every SCIFF IC is expressed in terms of abducibles only; (*ii*) prohibitions can be expressed in terms of a particular type of IC, named *denials*, like for instance in the *neg_response(A,B)* constraint, where the *fail* constant is used with the special purpose of addressing an inconsistency or a failure. Table 1 reports a few examples of how the Declare constructs can be easily mapped.

The translation of the Declare constraints alone is not enough: activities that are not subjected to any existence constraint, or activities that are not mentioned at all by any constraint (but are allowed due to the openness of the

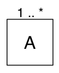
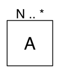
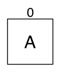
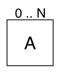
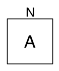
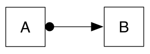
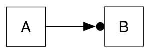
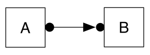
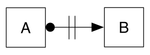
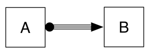
Name	Integrity Constraint	Graphical
existence(A)	$true \rightarrow \mathbf{ABD}(\mathsf{A}, T)$	
existence_N(A)	$true \rightarrow \bigwedge_{i=1}^{N} \Big(\mathbf{ABD}(\mathsf{A}, T_i) \wedge T_i > T_{i-1}\Big)$	
absence(A)	$\mathbf{ABD}(\mathsf{A}, T) \rightarrow fail$	
absence_N+1(A)	$\bigwedge_{i=1}^{N+1} \Big(\mathbf{ABD}(\mathsf{A}, T_i) \wedge T_i > T_{i-1}\Big) \rightarrow fail$	
exactly_N(A)	*existence_N(A)* $\wedge$ *absence_N+1(A)*	
response(A,B)	$\mathbf{ABD}(\mathsf{A}, T_A) \rightarrow \mathbf{ABD}(\mathsf{B}, T_B) \wedge T_B > T_A$	
precedence(A,B)	$\mathbf{ABD}(\mathsf{A}, T_A) \rightarrow \mathbf{ABD}(\mathsf{B}, T_B) \wedge T_A > T_B$	
succession(A,B)	*response(A,B)* $\wedge$ *precedence(A,B)*	
neg_response(A,B)	$\mathbf{ABD}(\mathsf{A}, T_A) \wedge \mathbf{ABD}(\mathsf{B}, T_B) \wedge T_A < T_B \rightarrow fail$	
chain_response(A,B)	*response*(A,B) $\wedge \mathbf{ABD}(\mathsf{A}, T_A) \wedge \mathbf{ABD}(\mathsf{B}, T_B) \wedge T_A > T_B$ $\wedge \mathbf{ABD}(\mathsf{X}, T_X) \wedge T_X > T_A \wedge T_X < T_B \rightarrow fail$	

Table 1: Mapping of some relevant Declare formulas onto SCIFF.

approach) will not appear in any trace. To address this issue, in the SCIFF specification of a Declare model we add a further set of ICs. In particular, we assume a finite set $\mathbb{A}$ of all activities that have to be considered. For each activity $\mathsf{x} \in \mathbb{A}$, we add the following:

$$true \rightarrow true \vee \mathbf{ABD}(\mathsf{x}, T_x) \tag{4}$$

$$\mathbf{ABD}(\mathsf{x}, T_1) \rightarrow true \vee \mathbf{ABD}(\mathsf{x}, T_2) \wedge T_2 > T_1 \tag{5}$$

The IC (4) ensures that either an activity x is not in a trace, or it is included in at least a trace. The IC (5) instead ensures that, once an activity x has been included in a trace, there is also the possibility of generating traces with two, three, ..., many instances of x. Notice that to avoid termination issues, we exploit an implementation feature of the SCIFF proof procedure: it explores the disjoints following the syntactical order in which they have been defined. By placing as a first disjoint always the *true* option in both IC(4) and (5), we ensure that the procedure terminates, and only if requested for further solutions it explores the other disjoints.

6.1.2 Generating negative traces

The generation of negative traces follows exactly the schema presented in Algorithm 1. For each Declare constraint, one or more ICs are generated, corresponding to the possible ways of negating the constraint. For example, let us consider again the IC (2), that corresponds indeed to a *response(A,B)* Declare constraint (i.e., the execution of an activity A must be followed by the execution of an activity B). We already discussed that there are two significant ways (out of three) for negating the IC. Formally, they are:

$$\begin{array}{r} true \rightarrow \mathbf{ABD}(a, T_a) \\ \mathbf{ABD}(a, T_a) \wedge \mathbf{ABD}(b, T_b) \rightarrow fail. \end{array} \tag{6}$$

and

$$\begin{array}{r} true \rightarrow \mathbf{ABD}(a, T_a) \\ \mathbf{ABD}(a, T_a) \wedge \mathbf{ABD}(b, T_b) \wedge T_b > T_a \rightarrow fail. \\ \mathbf{ABD}(a, T_a) \rightarrow \mathbf{ABD}(b, T_b) \wedge T_b < T_a. \end{array} \tag{7}$$

IC (6) states the prohibition of the happening of b, if a has happened. The reader might wonder why the denial is needed: if we simply resorted to not generate any activity b, we would have already achieved the goal of a trace with a and not b. However, other Declare constraints might call for the happening of b. The denial guarantees that the traces will not contain b, thus ensuring the violation of the IC (2).

A final consideration regards inconsistent models. When negating one or more ICs, inconsistent models will be generated: for example, a model with an IC asking for the presence of activity a, and also an IC prohibiting a. From the perspective of the SCIFF proof procedure inconsistent models are not an issue: simply, they will not produce any trace.

6.2 Positive and negative trace generation for closed procedural processes

A huge number of specification languages for closed procedural processes have been proposed. The number of approaches is so vast that their listing would be out of the scope of this work. We restrict ourselves to a well-known class of processes, often referred as structured process models [41] with unique tasks [6]. Although our approach could in theory deal with unstructured models, we prefer to adopt such restriction since structured models have a clearer semantics, in particular when dealing with loop patterns and termination conditions. Such a choice comes at the price that not all real processes can be easily represented, and more complex models would be needed to capture them. The choice of unique tasks instead is due to a simpler management of the process model translation into our framework: indeed, any process model with repeated activities can be always translated into another with unique tasks by applying a simple renaming of the repeated activities.

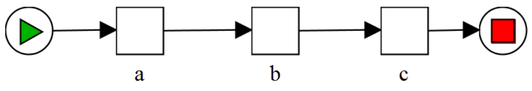


Fig. 3: A closed, procedural process defined in YAWL.

6.2.1 Generating positive traces

The generation of positive traces from procedural models follows the general three-steps procedure. As regards the translation of structured process models into SCIFF, we refer to the previous works [26,21,23]. As for template generation, it is worth noting that, differently from open approaches, in procedural closed models only those activities that are explicitly mentioned by the model are allowed in the trace. Hence, the specification of the procedural model in terms if SCIFF's constraints is sufficient to ensure the generation of the traces. There is no need for additional constraints such as IC (4) and (5).

6.2.2 Generating negative traces

When negating flow constructs of a closed procedural model, it is important to consider that the correct chaining of events might be broken, and certain negative traces might be not generated. Let us explain the issue through the workflow example shown in Fig. 3, where a very simple sequence of activities a, b, c is proposed. Let us focus, for the sake of comprehension, on generating those traces that violate the sequence constraint between activities a and b because they do not contain b. These traces are generated through IC (6). Intuitively, they would be:

$$\overline{\tau_{19}} = [\mathsf{a}]$$
$$\overline{\tau_{20}} = [\mathsf{a}, \mathsf{c}]$$

Since the sequence construct between b and c is represented as:

$$\mathbf{ABD}(b, T_b) \rightarrow \mathbf{ABD}(c, T_c) \wedge T_c > T_b.$$

in our formalism the generation of activity c is triggered by the presence of b. Its absence would clearly lead to not generating c. Practically, only trace $\overline{\tau_{19}}$ would be generated, while we would miss trace $\overline{\tau_{20}}$.

We overcome the issue through the same approach discussed in Section 6.1.1: for each activity $\mathsf{x} \in \mathbb{A}$ (thus also for c), we add the two ICs (4) and (5). Such a choice would lead to generate a number of non-compliant traces due to the presence of events at the wrong position in the trace and, eventually in the example, to the generation of $\overline{\tau_{20}}$.

It is worth to underline that this is not the only way in which we could have dealt with the issue. For example, a simple alternative could have been to generate positive traces and then randomly insert further interesting activities from $\mathbb{A}$. Nonetheless, we believe that our approach comes with the advantage of using the same logic framework for dealing with both negative and positives traces. Indeed, constraints (4) and (5) are used: (*i*) in case of open models - to insert additional, allowed activities not mentioned by any constraints in the trace, thus to generate other positive traces; (*ii*) in case of closed models to generate negative traces by inserting additional activities not envisaged by the model (or activities that do belong to the model but result placed in the wrong position inside the trace). In this sense, our approach uniformly deals with the generation of both negative and positives traces. Furthermore, the random insertion of activities in the trace once it is generated could leave some cases uncovered.

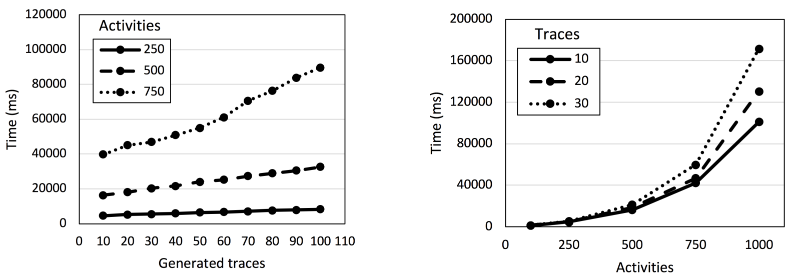
Another point is related to the strategy of identifying all possible ways to negate each constraint. It is indeed crucial when we deal with alternative flows in the model because it allows the generation of negative traces mixing activities from both paths. When a *xor* gate is in the model, as for example in Fig. 1 (see IC ic_2 of Section 5 for its SCIFF specification), there are several ways to negate such constraint. As previously discussed for IC (2), there are two significant ways to negate each of the two alternative sequences [a, b] and [a, c] entailed by the xor constraints. Furthermore, there are two ways to negate the disjunction: (*i*) after a, neither b, nor c are executed; (*ii*) after a, both b and c are executed. The latter negated form of the xor constraint generates negative traces containing both b and c, which originally belonged to alternative paths.

7 Evaluation of the prototype

Aiming to empirically verify our approach, we developed a first software prototype which employs the SCIFF abduction capabilities to generate positive and negative traces starting from a given business model.

The software, which is publicly available for download [51], allows to specify the input model in terms of the ICs that must be fulfilled and a set $\mathbb{A}$ of possible activities that can occur in the traces. As discussed in the previous sections, $\mathbb{A}$ can include further activities that do not appear in any constraint. The user can also specify various options, useful to control the generation process and the characteristics of the emitted traces. It is indeed possible to specify:

- if the model is *declarative* open or *procedural* closed;
- if *positives*, *negatives* or both types of traces must be generated;
- the *time limit* for each generated trace i.e., its maximum time length in seconds;
- the *maximum length* of the generated traces intended as the maximum number of events in each trace – particularly relevant to guarantee the termination of the generation procedure when loops are in the model;
- the number of *instances for each path* i.e., in case the model includes some alternative choices of different flow paths (as it is in Fig. 1), the maximum number of traces that must be generated for each alternative path.



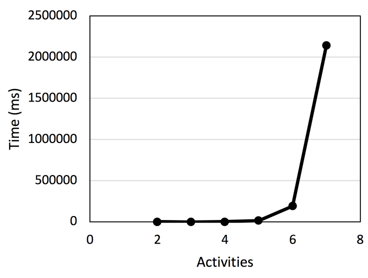
(a) (b)

Fig. 4: Evaluation of the times to generate positive traces for an arbitrarily long procedural model.

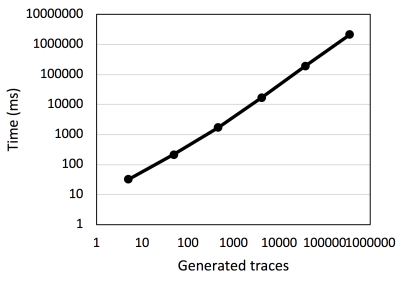
The emitted traces are provided in XES format [55], each one reporting a trace identifier and the list of the events with the corresponding timestamp. The download package includes a codification of the two models in Fig. 1 and 2a – as examples of procedural and declarative processes, respectively – that can be used to verify the generation abilities of the prototype as regards both positive and negative traces. The prototype originates 10 positive and 20 negative traces from the declarative model in Fig. 2a when the options of 5 seconds time limit, 5 activities as maximum length and 1 instance for each path are provided. With the same options the procedural model in Fig. 1 originates 4 positive and 4450 negative traces (as a consequence of its closeness flavour).

As discussed in [30], the complexity of the main abductive decision problem (i.e., to determine whether an explanation exists) is located at the second level of the polynomial hierarchy (Σ_2^P-complete). In order to empirically evaluate the generation performance of the proposed tool, we employ an arbitrarily big procedural model composed of a long sequence of activities. Thus, there is only one path allowed by the model from the start to the stop activity. We first estimate the time to generate an increasing number of only positive traces on an average hardware architecture (a quad-core Intel i7 2,9 GHz CPU and 16 GB RAM), when the SCIFF is given three different models composed of 250, 500 and 750 activities (see the graph in Fig. 4a). In order to force the generation of a certain number N of traces in each experiment, we set the *time limit* and *maximum length* to high values (so that the generation process is not influenced by them) and we ask to generate N *instances for each path*. Since there is only one possible path in the given model, we obtain exactly N traces in the output. As highlighted in Fig. 4a, the generation time increases linearly with the number of emitted traces.

Fig. 4b reports the performance of the tool when we fix to 10, 20 and 30 the number of generated positive traces, but we progressively increase the number of activities in the model. In this case, as a bigger model corresponds to more ICs to be considered, the generation time shows a superlinear trend in the number of activities. This is indeed expected, since a bigger model entails



(a)



(b)

Fig. 5: Evaluation of the times to generate positive and negative traces for an arbitrarily long procedural model. The performance decreases rapidly (superlinearly) with the number of activities in the process model (Fig. 5a), but linearly with the number of generated traces (Fig. 5b). Fig. 5b reports the data in logarithmic scale on both the axis to better appreciate the trend.

more activities for each instance of the process execution and requires more time to generate the corresponding trace.

The same test is repeated while asking the tool to generate both positive and negative traces. As the model is closed procedural, there are far more forbidden process instances than the allowed ones. Therefore, as shown in Fig. 5a, a slight increase in the number of model activities corresponds to a huge increase in the space of negative traces to be explored and, consequently, to a higher generation time. This is further confirmed by the graph in Fig. 5b, where we relate the number of traces generated in each experiment of Fig. 5a with the required time (with logarithmic scale on both axes to better appreciate the trend): the two dimensions are still linearly proportional.

8 Related work

Automated discovery of process models from event logs is one of the main research areas of process mining. As it focuses on extracting knowledge from business process logs, the evaluation of process discovery techniques inevitably requires the availability of event logs [28]. For this purpose, some real-life logs have been made publicly available [15] and are often used as benchmarks for testing process discovery tools. Given the real-life origin of these datasets, they may contain imperfections (i.e., non-compliant traces) or show incompleteness (i.e., miss some examples of traces that should be considered compliant), thus causing alterations in the discovered business model. Indeed, real-life logs usually reports all the observed traces without any indication of which cases are non-compliant or which execution paths are missing. For this reason, a widespread approach in this field is based on process simulation to artificially generate event logs with predefined characteristics. This allows the researchers to perform a finer tuning of the developed algorithms and better control the experimental evaluation. Some model simulators and log generators have been developed for this purpose [38,37].

In this section we propose a classification of some relevant works on log generation according to the following main directives:

- the ability to generate process execution examples with the employment of a *procedural* approach;
- the ability to support *declarative* constraints;
- the possibility to generate *positive* compliant traces;
- the possibility to generate *negative* non-compliant traces;
- the availability of generation mechanisms from *partially specified traces*;
- the ability to deal with *time* constraints;
- the ability to deal with *data* constraints;
- the possibility to generate traces with a user-defined *probability distribution of workflow execution paths and values*;
- the online *availability* of the proposed generation *tool*[4].

The approaches described in the following are classified according to these directives in Table 2.

The work [35] introduces a framework for the automated generation of Petri nets representing processes, according to user-defined rules. In particular, they suggest to gradually refine Workflow nets [2] in a top-down approach, in order to generate all possible process models belonging to the class of Jackson nets. A similar approach has been proposed in [14], where the authors describe a technique to generate Petri nets according to a different set of refinement rules. In both cases, the generated process models are intended to be used as benchmarks for process discovery algorithms, but the proposed approaches do not address the problem of generating traces from the developed Petri nets, hence we exclude the works [35] and [14] from the classification in Table 2.

In the works [18,19], the authors present a tool, the Processes Logs Generator (PLG), developed for the specific purpose of generating process discovery benchmarks. The software allows the user to randomly develop business models according to some predefined parameters and then "execute" the generated model while recording each activity in a log file. In [32], the authors present an approach based on CPN Tools [39] to simulate business process models. The key component of the simulator performs a template-oriented transformation from BPMN process models into CPNs. Another approach based on CPN Tools is described in [13], where the authors generate XML event logs by the simulation of a CPN. The work [57] uses simulation to evaluate the impact in terms of performance of re-engineering business processes. This approach creates simulation models from workflow processes and uses simulation results as a feedback to calibrate the model.

[4] We checked the availability on April 8th, 2019 starting from the URLs reported in the original papers. Note that some tools have been updated and moved to different web sites. In any case, the interested reader can contact the authors of the works to request the source code when not available online.

Table 2: Classification of log/model generators approaches according their most relevant features. Since providing a survey of all available works on log generation is beyond the scope of this paper, this list has not to be intend exhaustive.

Approach	Procedural	Declarative	Positive trace gen.	Negative trace gen.	Partially specified traces	Time constraints	Data constraints	Probability distribution of paths and values	Tool availability
Burattin et al. [19,18]	+	-	+	-	-	-	-	+	+
García-Bañuelos et al. [32]	+	-	+[*]	-	-	-	-[**]	+	+[a]
Alves de Medeiros et al. [13]	+	-	+[*]	-	-	+	+	+	-
Wynn et al. [57]	+	-	+[*]	-	-	+	+	+	-
Westergaard et al. [56]	+	+	+	-	-	-	-	-	+
Di Ciccio et al. [28]	-	+	+	-	-	-	-	-	+
Ackermann et al. [10,9]	-	+	+	-	-	+	+	-	+[c]
Accorsi et al. [8,53,54]	+	-	+	+	-	+	-	+	+
Goedertier et al. [33]	+	-	+	+	-	+	+	+	+
Broucke et al. [16,17]	+	-	+	+	-	+	+	+	+[b]
SCIFF approach	+	+	+	+	+	+	+	-	+

[*] Log generation is provided through Colored Petri Net (CPN) Tools.
[**] Currently not explored although CPN Tools allows it.
[a] Following an exchange of correspondence with the authors, we understand that the tool has been replaced by BIMP [7], available at http://bimp.cs.ut.ee/ on April 8th, 2019.
[b] Available at http://processmining.be/neconformance/ on April 8th, 2019.
[c] Available at https://www.ai4.uni-bayreuth.de/en/research/ToolsAndResources/index.html #tab_68762287 on April 8th, 2019.

All the approaches described so far support only procedural business process models plus additional information like, e.g., mean execution time of activities, probability of choices, etc. Eventually, a number of approaches support also the simulation of limited resources, their allocation, queues, etc. Nevertheless, these procedural approaches show some limitations when the process is characterized by high variability and allows for many alternative execution paths. In these cases, declarative process models are proven to perform better. For this reason, CPN Tools has presented an extension [56] of the traditional procedural-based modeller to graphically add Declare constraints [4]. The simulation of such hybrid models can be carried out by the user or executed in a random way. The work [28] presents a synthetic log generator that allows the user to define the process model by listing a number of declarative constraints expressed in Declare. More recently, another log generator based on a declarative approach has been presented in [10,9,11]. Here, the authors employ Declarative Process Intermediate Language (DPIL), a declarative process modelling language, to create a simulation model, and then generate XES event logs composed of only positive traces through a second component called Multi-perspective Declarative Process Simulation (MuDePS). Each generated log describes an exhaustive, distinct set of traces of the desired length.

The SCIFF approach can inherently give support to the definition of both declarative as well as procedural models and naturally exploit abduction to enable the generation. Furthermore, differently from SCIFF, all these solutions to log generation ignore the necessity of negative examples in the benchmark event log to determine how robust to noise a certain process discovery technique is [29]. Since information about state transitions that were prevented from taking place is often unavailable in real-life logs, it cannot be exploited in order to guide the learning task. For this reason, as also underlined in [33], in most cases process discovery techniques are limited to the harder setting of unsupervised learning. Our notion of negative traces may recall the concept of *syntactical noise* introduced by Günther in [34]. However, this similarity is only limited to the appearance of the trace (i.e., showing missing, out of order or unexpected events), whereas the hypothesized use of these instances remains different: Günther's noisy traces are intended as a consequence of distortions in transmitting or recording the log (and as such, should be identified and discarded during process discovery), whereas in this work, negative traces are intended as meaningful aspects of the process. Their non-compliance can be itself a source of knowledge, for example, to instruct an inductive mining tool.

In [31], the authors assume to have a set of negative events collected from domain experts who suggests whether a proposed execution plan is feasible or not. Given this knowledge, the authors combine ILP and partial-order planning techniques to process discovery. Similarly, other approaches [42,43,12,25] envisage the presence of "non-compliant" traces as a fundamental requirement to enable process mining tasks through supervised learning techniques. In [8, 53,54], the authors present a tool that takes a series of business process specifications as input and generates an event log. Based upon other user-defined security and compliance requirements, deviations from the defined control flow are generated. In particular, specific trace properties can be either enforced or violated on a random basis in the resulting event log. This approach shares with SCIFF the possibility to generate negative examples but is limited to support the definition of procedural business process models only. The work [33,16,17] propose a process discovery learning technique, which starts by generating artificially induced negative events and later uses these examples to enforce the learning process. Differently from our approach, in all these works the negative traces are not generated by modifying the model's constraints, but rather replacing the positive events of each trace and then checking if a state transition of interest (corresponding to a candidate negative event) could occur. Specifically, the authors assume that a generated transition can be considered as a negative example if it is not present in any trace with a similar history in the log. Although this assumption has the great advantage of allowing the user to deal with negative traces before having extracted the process model, it can lead to incorrect results. Indeed, the initial hypothesis of having all the positive examples in the log cannot be always guaranteed, causing the allowed state transitions that are not reported in the log to be classified as negative examples. On the contrary, if the log contains some not allowed trace, the generator will erroneously consider them as positive examples. Differently from these works, the SCIFF approach is able to ensure the validity of the generated positive and negative traces w.r.t. a model. This feature could be useful for example, in conjunction with an inductive miner (employing both positive and negative instances) to iteratively refine a business process model through subsequent steps of generation and mining.

9 Conclusions and Future Works

In the recent years, as the interest in BPM and process mining techniques has increased, the need for event log benchmarks with predefined characteristics has become more and more popular.

This work presents a novel approach for generating synthetic logs starting from a declarative or procedural business process model. We analysed the theoretical aspects of positive and negative trace generation and evaluate the feasibility of our method by providing a standalone prototype able to generate trace templates as well as completely grounded traces. Also, custom constraints on data and time can be specified to meet the application-dependent requirements. The performance evaluation of the proposed tool is promising although a superlinear trend in the generation time for increasing model dimension is clearly highlighted. The proposed approach is then put through a deep qualitative comparison with the existing literature on business process simulation, revealing the numerous advantages of the employment of the SCIFF framework in this field. Namely, the possibility to deal with both open declarative and closed procedural model specifications while producing positive and negative traces, and the ability to deal with partially specified traces.

This analysis of the state of the art has also highlighted some shortcomings of our approach, which will be addressed in the future. In particular, the generation of traces and their grounding does not consider any probability information, while some paths in the process model might be more frequent than others, as well as some data values and activity durations might be more probable than others. Furthermore, as data constraints are currently supported by directly modifying the SCIFF constraints, for the future, we plan to introduce some higher-level mechanism aiming to meet the needs of a non-technical user.

Since positive/negative log generation explores all the possible allowed and non-allowed traces given a business model, the process can require a significant amount of time, depending on the length of the model, the number of possible paths and the quantity and quality of constraints to be checked. When the SCIFF framework is used for compliance monitoring purposes, previous works [20,45] have proven the possibility to significantly speed up the checking process through the employment of programming models for distributed computation like MapReduce [27]. A similar approach could be useful to accelerate positive/negative log generation when a collection of computing nodes is available. For the future, we plan to explore this possibility in order to partition the generation process into smaller tasks that can be easily concurrently executed with the support of a large-scale data processing engine.

References

1. van der Aalst, W.M.P., Adriansyah, A., Alves de Medeiros, A.K., et alt.: Process mining manifesto. In: Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I, *Lecture Notes in Business Information Processing*, vol. 99, pp. 169–194. Springer (2011). URL `https://doi.org/10.1007/978-3-642-28108-2_19`
2. van der Aalst, W.M.P., van Hee, K.M.: Workflow management: models, methods, and systems. MIT Press (2002)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005). URL `https://doi.org/10.1016/j.is.2004.02.002`
4. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: towards a truly declarative service flow language. In: M. Bravetti, M. Núñez, G. Zavattaro (eds.) Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4184, pp. 1–23. Springer (2006). URL `https://doi.org/10.1007/11841197_1`
5. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: balancing between flexibility and support. Computer Science - R&D **23**(2), 99–113 (2009). URL `https://doi.org/10.1007/s00450-009-0057-9`
6. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004). URL `https://doi.org/10.1109/TKDE.2004.47`
7. Abel, M.: Lightning fast business process simulator. Master's thesis, University of Tartu, Estonia (2011)
8. Accorsi, R., Stocker, T.: SecSy: Synthesizing smart process event logs. In: R. Jung, M. Reichert (eds.) Enterprise Modelling and Information Systems Architectures: Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, EMISA 2013, St. Gallen, Switzerland, September 5-6, 2013, *LNI*, vol. 222, pp. 71–84. GI (2013). URL `https://dl.gi.de/20.500.12116/17247`
9. Ackermann, L., Schönig, S.: MuDePS: Multi-perspective Declarative Process Simulation. In: L. Azevedo, C. Cabanillas (eds.) BPM Demo Track 2016, *CEUR Workshop Proceedings*, vol. 1789, pp. 12–16. CEUR-WS.org (2016). URL `http://ceur-ws.org/Vol-1789/bpm-demo-2016-paper3.pdf`
10. Ackermann, L., Schönig, S., Jablonski, S.: Simulation of multi-perspective declarative process models. In: BPM 2016 Workshops, *LNBIP*, vol. 281, pp. 61–73 (2016). URL `https://doi.org/10.1007/978-3-319-58457-7_5`
11. Ackermann, L., Schönig, S., Jablonski, S.: Towards simulation- and mining-based translation of resource-aware process models. In: Business Process Management Workshops - BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers, *LNBIP*, vol. 281, pp. 359–371 (2016). URL `https://doi.org/10.1007/978-3-319-58457-7_26`
12. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. **9**(4), 29:1–29:43 (2008). DOI 10.1145/1380572.1380578
13. Alves de Medeiros, A.K., Günther, C.W.: Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. Procs. of the 6th works. on practical use of coloured petri nets and the CPN tools pp. 177–190 (2005)

14. Bergmann, G., Horváth, Á., Ráth, I., Varró, D.: A benchmark evaluation of incremental pattern matching in graph transformation. In: H. Ehrig, R. Heckel, G. Rozenberg, G. Taentzer (eds.) Graph Transformations, 4th International Conference, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 5214, pp. 396–410. Springer (2008). URL `https://doi.org/10.1007/978-3-540-87405-8_27`
15. BPI Challenge Real life Event Logs (2017). URL `https://data.4tu.nl/repository/collection:event_logs_real`
16. vanden Broucke, S.: Advances in process mining: Artificial negative events and othertechniques. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium (2014). URL `https://lirias.kuleuven.be/handle/123456789/459143`
17. vanden Broucke, S., Weerdt, J.D., Baesens, B., Vanthienen, J.: Improved artificial negative event generation to enhance process event logs. In: J. Ralyté, X. Franch, S. Brinkkemper, S. Wrycza (eds.) Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings, *Lecture Notes in Computer Science*, vol. 7328, pp. 254–269. Springer (2012). URL `https://doi.org/10.1007/978-3-642-31095-9_17`
18. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: L. Azevedo, C. Cabanillas (eds.) Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016., *CEUR Workshop Proceedings*, vol. 1789, pp. 1–6. CEUR-WS.org (2016). URL `http://ceur-ws.org/Vol-1789/bpm-demo-2016-paper1.pdf`
19. Burattin, A., Sperduti, A.: PLG: A framework for the generation of business process models and their execution logs. In: M. zur Muehlen, J. Su (eds.) BPM 2010 Workshops, *LNBIP*, vol. 66, pp. 214–219. Springer (2010). URL `http://doi.org/10.1007/978-3-642-20511-8_20`
20. Chesani, F., Ciampolini, A., Loreti, D., Mello, P.: Process mining monitoring for map reduce applications in the cloud. Proceedings of the 6th International Conference on Cloud Computing and Services Science (2016). URL `http://doi.org/10.5220/0005864000950105`
21. Chesani, F., Ciampolini, A., Loreti, D., Mello, P.: Abduction for generating synthetic traces. In: E. Teniente, M. Weidlich (eds.) Business Process Management Workshops, pp. 151–159. Springer International Publishing, Cham (2018)
22. Chesani, F., De Masellis, R., Francescomarino, C.D., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Abducing compliance of incomplete event logs. In: AI*IA 2016, Procs., *LNCS*, vol. 10037, pp. 208–222. Springer (2016). URL `https://doi.org/10.1007/978-3-319-49130-1_16`
23. Chesani, F., De Masellis, R., Francescomarino, C.D., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Abducing compliance of incomplete event logs. CoRR **abs/1606.05446** (2016). URL `http://arxiv.org/abs/1606.05446`
24. Chesani, F., De Masellis, R., Francescomarino, C.D., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Abducing workflow traces: A general framework to manage incompleteness in business processes. In: ECAI 2016, *Frontiers in Artificial Intelligence and Applications*, vol. 285, pp. 1734–1735. IOS Press (2016). URL `https://doi.org/10.3233/978-1-61499-672-9-1734`
25. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. Trans. Petri Nets and Other Models of Concurrency **2**, 278–295 (2009). URL `https://doi.org/10.1007/978-3-642-00899-3_16`
26. Chesani, F., Mello, P., Masellis, R.D., Francescomarino, C.D., Ghidini, C., Montali, M., Tessaris, S.: Compliance in business processes with incomplete information and time constraints: a general framework based on abductive reasoning. Fundam. Inform. **159**(1-2), 35–63 (2018). URL `https://doi.org/10.3233/FI-2018-1657`
27. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107 (2008). DOI 10.1145/1327452.1327492
28. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: EOMAS 2015, Held at CAiSE 2015, *LNBIP*, vol. 231, pp. 20–36. Springer (2015). URL `http://doi.org/10.1007/978-3-319-24626-0_2`

29. Di Ciccio, C., Mecella, M., Mendling, J.: The effect of noise on mined declarative constraints. In: Data-Driven Process Discovery and Analysis - Third IFIP WG 2.6, 2.12 International Symposium, SIMPDA 2013, Riva del Garda, Italy, August 30, 2013, Revised Selected Papers, *LNBIP*, vol. 203, pp. 1–24. Springer (2013). URL `https://doi.org/10.1007/978-3-662-46436-6_1`
30. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. J. ACM **42**(1), 3–42 (1995). URL `http://doi.acm.org/10.1145/200836.200838`
31. Ferreira, H.M., Ferreira, D.R.: An integrated life cycle for workflow management based on learning and planning. Int. J. Cooperative Inf. Syst. **15**(4), 485–505 (2006). URL `https://doi.org/10.1142/S0218843006001463`
32. Garcıa-Banuelos, L., Dumas, M.: Towards an open and extensible business process simulation engine. In: CPN Workshop 2009, pp. 199–208 (2009)
33. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. Journal of Machine Learning Research **10**, 1305–1340 (2009). URL `http://doi.acm.org/10.1145/1577069.1577113`
34. Günther, C.: Process mining in flexible environments. Ph.D. thesis, Department of Industrial Engineering & Innovation Sciences (2009). DOI 10.6100/IR644335
35. van Hee, K.M., Liu, Z.: Generating benchmarks by random stepwise refinement of Petri nets. In: S. Donatelli, J. Kleijn, R.J. Machado, J.M. Fernandes (eds.) Proceedings of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010), Braga, Portugal, June, 2010, *CEUR Workshop Proceedings*, vol. 827, pp. 403–417. CEUR-WS.org (2010). URL `http://ceur-ws.org/Vol-827/31_KeesHee_article.pdf`
36. van Hee, K.M., Sidorova, N., van der Werf, J.M.E.M.: Business process modeling using petri nets. Trans. Petri Nets and Other Models of Concurrency **7**, 116–161 (2013). URL `https://doi.org/10.1007/978-3-642-38143-0_4`
37. Imam, I., Nounou, N., Hamouda, A., Khalek, H.A.A.: Survey of business process simulation tools: a comparative approach. In: Proc. SPIE 8285, International Conference on Graphic and Image Processing (ICGIP 2011), 82853B (1 October 2011), vol. 8285, pp. 1–7 (2011). URL `http://doi.org/10.1117/12.914265`
38. Jansen-vullers, M.H., Jansen-vullers, M.H., Netjes, M.: Business process simulation – a tool survey. Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN (2006). URL `http://doi.org/10.1.1.87.8291`
39. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. STTT **9**(3-4), 213–254 (2007). URL `https://doi.org/10.1007/s10009-007-0038-x`
40. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992). URL `https://doi.org/10.1093/logcom/2.6.719`
41. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On structured workflow modelling. In: J.A.B. Jr., J. Krogstie, O. Pastor, B. Pernici, C. Rolland, A. Sølvberg (eds.) Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE, pp. 241–255. Springer (2013). URL `https://doi.org/10.1007/978-3-642-36926-1_19`
42. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: G. Alonso, P. Dadam, M. Rosemann (eds.) Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings, *Lecture Notes in Computer Science*, vol. 4714, pp. 344–359. Springer (2007). URL `https://doi.org/10.1007/978-3-540-75183-0_25`
43. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: H. Blockeel, J. Ramon, J.W. Shavlik, P. Tadepalli (eds.) Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 4894, pp. 132–146. Springer (2007). URL `https://doi.org/10.1007/978-3-540-78469-2_16`
44. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
45. Loreti, D., Chesani, F., Ciampolini, A., Mello, P.: A distributed approach to compliance monitoring of business process event streams. Future Generation Computer Systems **82**, 104 – 118 (2018). URL `https://doi.org/10.1016/j.future.2017.12.043`

46. Marrella, A., Lespérance, Y.: A planning approach to the automated synthesis of template-based process models. Service Oriented Computing and Applications **11**(4), 367–392 (2017). URL `https://doi.org/10.1007/s11761-017-0215-z`
47. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, *Lecture Notes in Business Information Processing*, vol. 56. Springer (2010). URL `https://doi.org/10.1007/978-3-642-14538-4`
48. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. TWEB **4**(1), 3:1–3:62 (2010). URL `http://doi.acm.org/10.1145/1658373.1658376`
49. Muggleton, S., Raedt, L.D.: Inductive logic programming: theory and methods. J. Log. Program. **19/20**, 629–679 (1994). DOI 10.1016/0743-1066(94)90035-3
50. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: J. Eder, S. Dustdar (eds.) Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4103, pp. 169–180. Springer (2006). URL `https://doi.org/10.1007/11837862_18`
51. Synthetic log generation through abduction (2018). DOI 10.5281/zenodo.2625707. URL `http://ai.unibo.it/LogGeneration,https://github.com/ai-unibo/log-generator`
52. Shanahan, M.: An abductive event calculus planner. The Journal of Logic Programming **44**(1), 207 – 240 (2000). URL `ttps://doi.org/10.1016/S0743-1066(99)00077-1`
53. Stocker, T., Accorsi, R.: SecSy: security-aware synthesis of process event logs. In: Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI), pp. 71–84 (2013)
54. Stocker, T., Accorsi, R.: SecSy: A security-oriented tool for synthesizing process event logs. In: Procs. of the BPM Demo Sessions 2014, *CEUR Procs.*, vol. 1295, p. 71. CEUR-WS.org (2014). URL `http://ceur-ws.org/Vol-1295/paper13.pdf`
55. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: P. Soffer, E. Proper (eds.) Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers, *Lecture Notes in Business Information Processing*, vol. 72, pp. 60–75. Springer (2010). URL `https://doi.org/10.1007/978-3-642-17722-4_5`
56. Westergaard, M., Slaats, T.: CPN tools 4: A process modeling tool combining declarative and imperative paradigms. In: BPM Demo sessions 2013, *CEUR Procs.*, vol. 1021, pp. 1–5. CEUR-WS.org (2013). URL `http://ceur-ws.org/Vol-1021/paper_3.pdf`
57. Wynn, M.T., Dumas, M., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Business process simulation for operational decision support. In: BPM Workshops, BPM 2007, *LNCS*, vol. 4928, pp. 66–77. Springer (2007). URL `https://doi.org/10.1007/978-3-540-78238-4_8`

Author Biographies

Daniela Loreti is a post-doc researcher at CIRI Health Sciences & Technologies (HST) and assistant professor of Operating Systems at DISI University of Bologna. She received the Ph.D. degree in Computer Science in 2016. Her research focuses on big data and stream process architectures as well as distributed programming models. She is also interested in artificial intelligence techniques for Process Mining, expert systems and argumentative agents. She is currently involved in the POR-FESR project Habitat, which aims to realise an assistive domestic system to support the life of elderly people through IoT smart objects and rule-based decision support systems.

Federico Chesani, PhD, is Associate Professor of Computer Science at DISI University of Bologna. His research and teaching activities focus on the area of Logic Programming, Business Processes Modelling, Distributed Verification and Monitoring, Rule-based Decision Support Systems. Federico Chesani has published more than 100 papers in international conferences and journals and established collaborations with several research groups, universities and private companies. He has been involved in several projects, as workpackage leader for the prototypes implementation (EU FP7 e-Policy), and as researcher for probabilistic and monitoring tools (EU FP7 Farseeing, H2020 PreventIT).

Anna Ciampolini, PhD in Computer Science and Engineering, is full professor at the Department of Computer Science and Engineering University of Bologna, where she teach Operating Systems. She currently is the deputy head of the Department. Her research interests include: operating systems, virtualization techniques and cloud computing, parallel and distributed programming, automatic management of Cloud computing systems, distributed platforms for big data analysis, distributed artificial intelligence, with particular regard to distributed automated reasoning. She has been involved in several international projects, also in coordination roles. Her research covers both application and theoretical aspects as shown by her broad bibliographic production.

Paola Mello. Full Professor at the University of Bologna since 1994, she conducts her research in Artificial Intelligence. In particular, her research interest, both practical and theoretical, are about knowledge representation, computational logic and logic languages, multi-agent and decision support systems, monitoring and verification, with applications in medicine, web services configuration, business processes management. She is involved in different research projects both national and international and she is author of numerous publications in prestigious conferences and journals. In 2017 she has been nominated fellow of the European Association for Artificial Intelligence (EurAI).