Towards the Generation of the "Perfect" Log Using Abductive Logic Programming

Federico Chesani[1][0000−0003−1664−9632], Chiara Di Francescomarino[2][0000−0002−0264−9394], Chiara Ghidini[2][0000−0003−1563−4965], Daniela Loreti[1][0000−0002−6507−7565], Fabrizio Maria Maggi[3][0000−0002−9089−6896], Paola Mello[1][0000−0002−5929−8193], Marco Montali[4][0000−0002−8021−3430], Vasyl Skydanienko[3], and Sergio Tessaris[4][0000−0002−3156−2669]

[1] University of Bologna, viale Risorgimento 2, 40136—Bologna, Italy
{federico.chesani,daniela.loreti,paola.mello}@unibo.it
[2] FBK-IRST, Via Sommarive 18, 38050 Trento, Italy.
{dfmchiara,ghidini}@fbk.eu
[3] University of Tartu, Estonia
{skydanienko,f.m.maggi}@ut.ee
[4] Free University of Bozen–Bolzano, piazza Università, 1, 39100 Bozen-Bolzano, Italy.
{montali,tessaris}@inf.unibz.it

Abstract. Data is the new raw material for business. As regards Business Process Management, this natural resource is contained into a process log in the form of recorded executions of process instances. In this framework, the evaluation of process mining techniques like process discovery and conformance checking demands for solid benchmark suites composed of logs with specific characteristics, which are rarely available in real contexts. Although the creation of the "perfect log" is certainly an unattainable hope, the availability of tools for synthetic log generation could simplify the development and test of novel process mining techniques. In this work, starting from an existing approach based on Abductive Logic Programming, we further report on a tool for synthetic log generation based on this approach, and suggest a practical application of the tool in the field of process engineering.

Keywords: Synthetic Log Generation · Abductive Logic Programming · Declarative business process models · Process Mining.

1 Introduction

In the context of Business Process Management (BPM), and even more in the process mining field [1], event logs play a key role. Therefore, the availability of event logs is essential for the evaluation of process mining algorithms. Consider for example, predictive process monitoring, a branch of process mining consisting of techniques for predicting the future development of a running process instance based on historical logs. To be able to evaluate predictive process monitoring techniques, large event logs with specific characteristics are usually needed. These characteristics include not only what sequences of activities are available in the log, but also the data payloads associated with activities [9, 12]. Another typical process mining task in which the need of event logs with certain characteristics is crucial is the evaluation of process discovery algorithms. The typical procedure for evaluating process discovery algorithms indeed consists of (i) generating an event log starting from a process model; (ii) mining a process model with the given discovery algorithm from this log and (iii) comparing the initial model (representing the gold standard) with the discovered one.

Although some real-life logs are publicly available to accomplish this type of tasks, their number is still quite limited. Most companies, indeed, are not willing to share their own event logs, e.g., due to privacy issues or because they depend on third parties. Moreover, real-life logs are not always the best choice for evaluating process mining approaches, as they do not offer control over different parameters such as the log size, the length of the traces or the amount of noise. Having control over these characteristics is of utmost importance to be able to evaluate a process mining algorithm in a complete and predictable way, e.g., in order to be able to understand how robust is the mining algorithm with respect to the noise or scalable with respect to the log size.

Although the creation of the "perfect log" is an unattainable hope, the availability of tools for synthetic log generation could simplify the development and test of novel process mining techniques. Some effort has been recently done towards this direction with the realization of procedural log generators [4] for the creation of event logs and associated data. However, currently, there are very few tools available for generating artificial logs from declarative process models [8, 2, 18] and even less able to deal with the generation of data payloads.

Over the last decade, various declarative languages have been proposed to represent business processes. One of the most popular among them is probably Declare [17], whose formalism can be easily mapped into logic-based languages such as Event Calculus [16], Linear Temporal Logic (LTL) [17], Abductive Logic Programming (ALP) [7], and computational logic in general. Recently, also declarative-based tools have appeared, which are able to synthesize logs with a number of features such as the possibility to include negative traces.

In particular, in [14] we have shown how ALP can be used to generate trace templates, starting from a model specification, where the model can be described through a procedural representation language, or a declarative one, and the model can be open or closed (i.e., every activity that is not explicitly requested to be executed, might be allowed or forbidden, respectively). As sketched in [6], the key idea is that abductive explanations, i.e., sets of abudcibles (possibly containing constrained variables) obtained by the SCIFF proof procedure [3], can be interpreted from a BPM perspective as intensional trace templates. In turn, sets of ground instances of such templates could be then interpreted as process logs.

In this work, we further explore our approach to synthetic log generation by deepening its main characteristics and suggesting a practical application in the field of process engineering. In particular, we present a tool called ALGen (Abductive Log Generator) implementing our approach. Then, after clarifying how the tool deals with declarative constraints – including constraints on data – in the model specification and after providing insights on both positive and negative trace generation, we focus on the methodology adopted to generate grounded, extensional instances, as well as trace templates.

In the remainder of the paper, we summarize the description of the log generator (Section 2), introduce a running example (Section 3) and show how the constraints of the running examples can be translated in the input of the log generator. In Section 4, we show challenges and applications of the proposed log generator and, finally, in Section 5, we draw some conclusions.

2 The Abductive Log Generator

The generation process relies on a previously developed ALP framework: the Social Constrained IFF (SCIFF)[3]. This software provides a logic-based language for expressing the domain knowledge and a proof procedure supporting the abductive reasoning process. Initially, it was developed to support the run-time compliance checking of multi-agent systems' behavior w.r.t. a given model, whereas later [7], its abductive capabilities were employed to extend the concept of compliance to logs with incomplete traces (i.e., with missing events).

A SCIFF specification can be formally intended as a triple $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where:

- $\mathcal{KB}$ is a knowledge base (i.e., a Logic Program as for [13]),
- $\mathcal{A}$ is a set of abducible predicates (predicates that can be hypothesized i.e., predicates with functor **ABD**, **E**),
- $\mathcal{IC}$ is a set of Integrity Constraints (ICs).

Abducibles **ABD** correspond to commonly intended abducibles of ALP [11], whereas abducibles with functor **E** model expectations about the happening of certain events.

$\mathcal{IC}$ is a set of ICs in the form of forward rules *body* $\rightarrow$ *head*, stating that when *body* becomes true, also *head* must be true. These SCIFF's ICs model the link between the happening of events and the expectations, thus *body* contains conjunctions of special terms with functor **H** (indicating the actual happening of an event), with functors **E** and **ABD**, while the *head* is made up of disjunctions of conjunctions of terms with functor **E** or **ABD**.

This SCIFF specification is used to express, in terms of ICs, the relations between events, such as for example "If event a happens at time T_a, then b is expected to happen at time T_b":

$$\mathbf{H}(\mathsf{a}, T_a) \rightarrow \mathbf{E}(\mathsf{b}, T_b). \tag{1}$$

This formalization is important in the context of compliance checking because, if an event matching with the expectation **E** occurs in the log, then we can say that the expectation is *fulfilled*. If such an event does not happen, the expectation is *violated*.

As the goal of the generator is not to determine if an existing trace is compliant, but rather to produce sets of traces with predefined characteristics (possibly compliant with, or violating a given process model), it employs a representation of the model in terms of ICs defined in terms of abducibles and expectations only.

The generation of synthetic logs through SCIFF proceeds in two steps. First of all, the SCIFF proof procedure is given the ICs (representing the model) and an empty trace as starting points, and it is queried about the existence of a trace. This triggers the generation of a *trace template* (as defined in [6]), where all the events have been abduced through hypothetical reasoning from rules in the model specification. In this first step, timestamps and activity data (if present) are indicated as variables, each one with its own set of constraints. Hence, the output is not a proper trace containing ground values, but rather a "template" with constrained variables: each trace template can be grounded to a number of different traces. Once such a first template is found, the SCIFF is asked to look for another one so that, iteratively, all the templates are pointed out. To ensure the termination of this operation even in the presence of (indeterministic) loops, a *meta*-information about the process is required: the user must specify a *maximum length*, i.e., a maximum number of events in each trace.

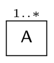
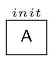
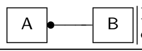
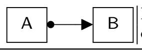
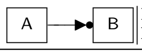
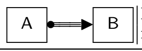
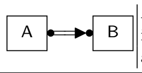
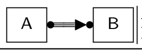
Secondly, the trace templates are *grounded* by substituting variables with values that fulfill the ICs. Two issues here arise:

- variables might have infinite domains, hence, the grounding step might not terminate. To cope with such issue, we explicitly ask the user to specify, for each variable, a finite domain. This is true also for event timestamps: the user is asked to provide a maximum time for the happening of the last event of the trace;
- synthetic process logs should provide a good "coverage" for all the data and time domains, but should not exceed by enumerating all the possible solutions.

The latter point indeed points to a more general problem about the characteristics that a log should exhibit. Roughly speaking, a log should enjoy the following features:

1. *in-trace* features such as, for example, trace length, which activities are surely in the trace, and which constraints are surely respected by the data fields in the trace templates. Such type of constraints can be easily covered by our approach, by properly representing the process model (as explained in Section 3.1);
2. *data-domain coverage* features such as, for example, ensuring that data values are distributed over a predefined standard distribution, or over a uniform distribution. To this end, we extend the grounding step by adding constraints, so as to force the solver to look for groundings with the desired features;

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

TEMPLATE	FORMALIZATION	NOTATION	DESCRIPTION
existence(A)	$\Diamond A$		A occurs at least once
init(A)	A		A is the first event to occur
resp. existence(A,B)	$\Diamond A \rightarrow \Diamond B$		If A occurs, B must occur as well
response(A,B)	$\Box(A \rightarrow \Diamond B)$		If A occurs, B must eventually follow
precedence(A,B)	$\neg B \mathcal{W} A$		B can occur only if A has occurred before
chain response(A,B)	$\Box(A \rightarrow \bigcirc B)$		If A occurs, B must occur next
alt. succession(A,B)	$(\neg B \mathcal{W} A) \wedge$ $\Box(B \rightarrow \bigcirc(\neg B \mathcal{W} A)) \wedge$ $\Box(A \rightarrow \bigcirc(\neg A \mathcal{U} B))$		A and B occur if and only if B follows A and they alternate each other
chain succession(A,B)	$\Box(A \longleftrightarrow \bigcirc B)$		A occurs if and only if B immediately follows

3. *inter-trace* features such as, for example, how many traces generated from a specific trace template are in the log; or, how many traces that do not respect the model (i.e., negative traces) are in the log.

Depending on the application purposes for which the log is generated, a number of different features might be identified. For example, in [10] several structural log metrics are identified, such as the magnitude of the log, the support, the variety and the level of detail, and the time granularity. Thanks to the expressive power of the logic programming approach, these metrics can be exploited in the log generation process, towards the creation of a log tailored to the user needs.

3 The running example

In this section, we describe the scenario used to generate the event logs, which is related to the management of a loan application. In this scenario, the process starts when an application for a loan of a certain amount is submitted by a requester with a given salary. The application undergoes several steps, which sometimes include checking the requester career and his/her medical history, and ends up with the assessment of the loan and the notification of the outcome to the requester.

The scenario can be modeled in a declarative language as, for instance, in DECLARE [17]. A DECLARE model consists of a set of rules, which are instantiations of templates on real activities. DECLARE templates have a graphical representation and their semantics can be formalized in different logics the main one being LTL on finite traces. Table 1 reports the main templates and their LTL semantics.

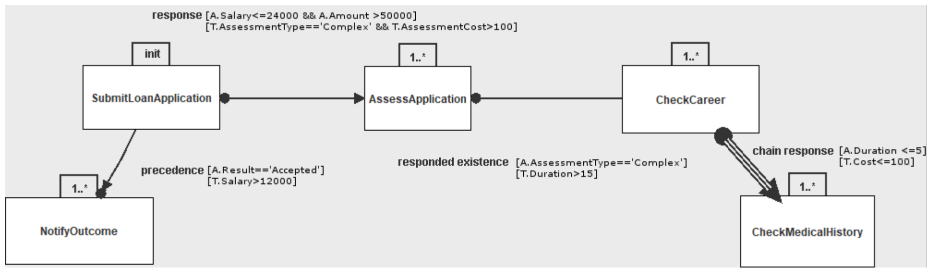

response [A.Salary<=24000 && A.Amount >50000]
[T.AssessmentType=='Complex' && T.AssessmentCost>100]
init
SubmitLoanApplication
1..*
AssessApplication
1..*
CheckCareer
precedence [A.Result=='Accepted']
[T.Salary>12000]
responded existence [A.AssessmentType=='Complex']
[T.Duration>15]
chain response [A.Duration <=5]
[T.Cost<=100]
1..*
NotifyOutcome
1..*
CheckMedicalHistory

Fig. 1. The loan process described in MP-Declare

Fig. 1 shows the declarative model described in MP-Declare [5] (Multi-Perspective Declare), an extension of Declare that allows for expressing conditions also on time and data. The MP-Declare templates allow for specifying five parameters, namely the activation, the target, the activation condition, the correlation condition and the time condition. The *activation* (A) of a constraint is an event whose occurrence imposes, because of that constraint, some obligations on the occurrence of a *target* event (T). The *activation condition* is a relation (over the payload of the activation event) that must be valid in order for the constraint to be activated. The *correlation condition* relates the payload of the activation event and the payload of the target event and has to hold in order for the constraint to be fulfilled. Finally, in MP-Declare, also a time condition can be specified through an interval ($I = [\tau_0, \tau_1)$) indicating the minimum and the maximum temporal distance allowed between the occurrence of the activation and the occurrence of the corresponding target.

According to the model, for instance, if the `Salary` of the loan requester is lower than 24 000 and the `Amount` of the requested loan higher than 50 000, the application is *assessed* with `AssessmentType` *complex* and a high `AssessmentCost` (`AssessmentCost` higher than 100). Whenever the `result` of the notification is an *acceptance*, it has to be preceded by a submission loan carried out by a requester with a `Salary` higher than 12 000. Whenever the career of the requester is checked, also his/her medical history is checked immediately after. In detail, if the `Duration` of the career check is equal or lower than 5, the `Cost` of the medical history check is low (it is lower or equal than 100).

3.1 Mapping MP-Declare models in ALP

To generate the ALP-based model of the business process, each MP-Declare constraint pattern is translated into one or more ICs. For example, the activity `SubmitLoanApplication` shown in Fig. 1 is subject to the `init` constraint, i.e., each process instance should always begin with an instance of

`SubmitLoanApplication`. This is achieved by simply stating the following IC:

$$\begin{aligned} true \rightarrow & \mathbf{ABD}(\texttt{SubmitLoanApplication(Salary,Amount)}, T) \\ & \wedge T > 0 \wedge T < 10 \\ & \wedge \texttt{Salary} > 1000 \wedge \texttt{Salary} < 100\,000 \\ & \wedge \texttt{Amount} > 10\,000 \wedge \texttt{Amount} < 300\,000 \\ & \wedge \mathbf{ABD}(\texttt{init}, T). \end{aligned} \tag{2}$$

The antecedent of IC (2) is *true*, meaning that the rule is triggered immediately at the start of the SCIFF proof procedure. As a consequence, the happening of an event `SubmitLoanApplication` is hypothesized (i.e., abduced), with data fields `Salary` and `Amount`, constrained in the specified domains. This is achieved thanks to the fact that SCIFF integrates and supports CLP (in this specific example, CLP over Finite Domains): `Salary` and `Amount` are simply represented as CLP variables. Upper bounds are placed on the time instant the event is expected to happen, as well as on the variables `Salary` and `Amount`: the latter are imposed by the original MP-Declare model, while the former is user-definable, and it is needed to ensure the grounding of the variable. Note also the abduction of the special event `init`: its role will be fundamental to ensure the semantics of the MP-Declare `init` constraint.

A characteristic feature of MP-Declare models is that they are *open*, i.e., activities that are not explicitly prohibited can happen during the process execution. To generate process logs compliant with such process models, we decided to impose a constraint: the set of activities that can freely happen during a process execution should be finite, and known a-priori. Given such assumption, for each activity `x` in the set, we add the following pair of ICs:

$$\begin{aligned} \mathbf{ABD}(start, T0) \rightarrow & true \\ & \vee \\ & \mathbf{ABD}(\texttt{x}, T) \\ & \wedge T > 0 \wedge T < 10 \wedge T > T0. \end{aligned} \tag{3}$$

$$\begin{aligned} \mathbf{ABD}(\texttt{init}, T0) \wedge \mathbf{ABD}(x, T1) \rightarrow & \\ & true \\ & \vee \\ & \mathbf{ABD}(x, T2) \\ & \wedge T2 > T1 \wedge T2 > 0 \wedge T2 < 10 \wedge T2 > T0. \end{aligned} \tag{4}$$

IC (3) ensures that as soon as the `init` of a process instance is hypothesized, then either no activity is added to the trace, or an activity `x` is abduced. IC (4) then ensures that any time the happening of an activity `x` is hypothesized, either no other activity `x` is added to the trace (first disjunct of the IC), or a further activity `x` is added (second disjunct). In other words, ICs (3) and (4) generate traces with more and more activity executions. Each new activity execution has a timestamp that is constrained to be greater than $T0$, the timestamp of the special `init` abducible, thus ensuring the semantics of the MP-Declare `init` constraint.

Finally, consider the response constraint between the activities `SubmitLoanApplication` and `AssessApplication` previously introduced. Its formalization as IC is the following:

$$\begin{aligned}
&\mathbf{ABD}(SubmitLoanApplication(Salary, Amount), T1) \\
&\quad \wedge Salary \leq 24000 \wedge Amount > 50000 \rightarrow \\
&\quad \mathbf{E}(AssessApplication(AssessmentType, AssessmentCost), T2) \\
&\quad \wedge T2 > T1 \wedge T2 > 0 \wedge T2 < 10 \\
&\quad \wedge AssessmentType == 1 \wedge AssessmentCost > 100.
\end{aligned} \tag{5}$$

Whenever the happening of `SubmitLoanApplication` is hypothesized (abduced), then the execution of `AssessApplication` is expected. Note that the expectation, in the SCIFF framework, requires to be matched by a corresponding event. In other words, the IC (5) only poses a requirement over the final trace, but no happening of event is hypothesized. Indeed the generation mechanism has been embedded through ICs (3) and (4), while IC (5) ensures that only traces with certain properties are considered as solutions. Other MP-Declare relational constraints are represented similarly.

Once the MP-Declare model is properly translated, it is possible to ask the SCIFF proof procedure to compute the abductive answer, that corresponds to the trace template. More than one abductive answer exist; for example a generated trace template is the following one:

$$\begin{aligned}
\tau = \{&abd(submitLoanApplication(Salary, Amount), T1), \\
&abd(assessApplication(1, AssessmentCost), T2), \\
&abd(checkCareer(Coverage), T3), \\
&abd(checkMedicalHistory(Cost), T4), \\
&abd(event(notifyOutcome(0), T5), \\
&Salary : 12001 \mathrel{..} 24000, \\
&Amount : 50001 \mathrel{..} 299999, \\
&T1 : 1 \mathrel{..} 8, \\
&AssessmentCost : 101..199, \\
&T2 : 2 \mathrel{..} 9, \\
&Coverage : 16 \mathrel{..} 29, \\
&T3 : 2 \mathrel{..} 9, \\
&Cost : 11 \mathrel{..} 199, \\
&T4 : 2 \mathrel{..} 9, \\
&T5 : 2 \mathrel{..} 9\}
\end{aligned} \tag{6}$$

Note that in the abdcutive answer τ, besides the abducibles that have been computed, also intervals of CLP variables are provided. Any grounded trace is simply generated starting from τ.

3.2 Generating negative traces

With the term "negative traces" we indicate those traces that are not compliant with the process model. In MP-Declare, the process model is defined in terms of a set of constraint about the execution of the activities. Hence, a trace is *negative* if it violates one or more constraints. In our approach, each MP-Declare constraint is represented through one or more ICs, i.e. (forward) implications. An implication is violated when the premises are true, but the consequences do not hold.

The main idea behind the generation of negative traces is to exploit the same mechanism envisaged for the generation of the positive ones, but feeding it with a different (modified) process model. For each (non-empty) subset of the ICs composing the model, we remove such subset from the original process model, we "negate" all the ICs in the subset, and we add them to the model again. Any trace that is compliant with the obtained model will violate the original model. Thus, the model obtained in this way is provided to the proof procedure, and traces are generated.

Several considerations need to be done. First of all, by "negating" one or more ICs and re-adding them to the model, there is no assurance that a trace can be generated, simply because the added ICs might make the new model inconsistent. This is expected: indeed, in our approach, we consider *all* the possible subsets of ICs, thus exploring all the possible combinations of "negated" ICs.

Secondly, the term "negating an IC" is slightly misleading: we are not putting a negation over the whole implication, but rather we are negating the consequent. In the simplest case, the consequent is made by a single literal, hence, its negation is straightforward. But if we consider the `response` constraint shown in Eq. (5), the consequent is made up of conjuncts and the negation of the consequent is a disjunction of negated literals. Hence, many different "negated" ICs can be obtained, each IC having as a consequent a subset of the negated literals.

As the reader can imagine, the approach for the generation of negative traces suffer a complexity issue due to the powerset of the ICs to be negated, and then a further complexity issue due to the powerset of the negated consequents of each IC. This is indeed expected, given that a constrained process model tends to limit the allowed traces: in turn, the number of not allowed traces is usually larger.

3.3 Proof of Concept

As a first proof-of-concept validation, we used ALGen (Abductive Log Generator) to generate a positive event log starting from the MP-Declare model described in Section 3 and we then applied the ProM *DeclareMiner* plug-in to

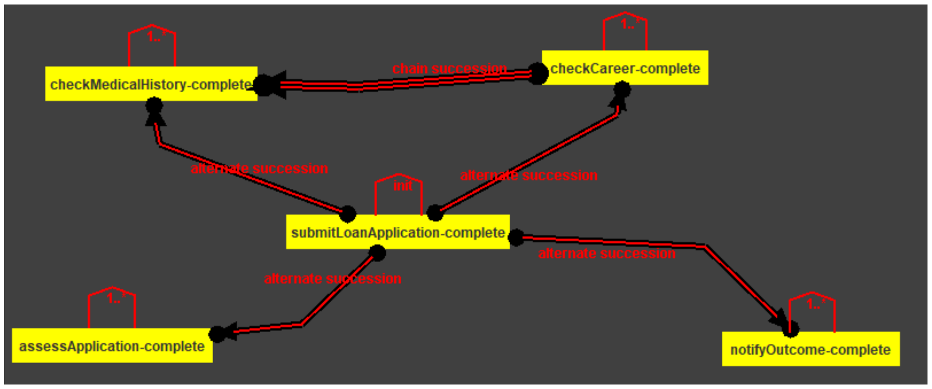


Fig. 2. DECLARE model mined by the ProM *DeclareMiner* plug-in

rediscover the declarative model. In detail, we generated 4000 traces (in ∼17 seconds) with maximum trace length equals to 6. For each trace following a certain path of the model, we generated, for each variable (e.g., `Salary`, `Amount`), three different values uniformly distributed over the domain of the variable. Concerning the parameters of the discovery plug-in, instead, we disabled the vacuity detection, i.e., if the constraints are not activated the constraints evaluate to true, and we set the minimum support threshold to 100%, i.e., only constraints supported by all the traces in the log are discovered.

Fig. 2 shows the discovered DECLARE model. The `init` and the `existence` constraints have an exact correspondence in the original model. Concerning the other constraints, instead, we observe an overall specialization with respect to the original model (e.g., `alt. successions` rather than `response` or `precedence`) and the discovery of constraints that are not present in the original model (e.g., the `alt. succession` between `SubmitLoanApplication` and `CheckMedicalHistory`). This analysis confirms that the generated log is actually compliant with the original model, though more specialized.

4 Challenges and Applications

In this section, we give a clue of one of the challenges that a log generator needs to face when generating event logs for process mining tasks (Section 4.1) and of a possible application of the log generator in an iterative user-guided methodology for discovering process models closer and closer to the desired model (Section 4.2).

4.1 The quest for the "perfect" log

Event logs represent the starting point for several approaches in the context of process mining, e.g., process discovery or predictive process monitoring techniques. In order to be used by these techniques and provide meaningful results, however, event logs should have certain quality criteria. One of the challenges of event log generators is hence questing for the *"perfect" log*. Unfortunately, a unique perfect log does not exist, as for different tasks, or even for different algorithms accomplishing the same task, the quality criteria (and hence the perfect log) could differ. One of the event log quality criteria shared by many process mining tasks is *case diversity*. In other terms, one of the main features that an event log should guarantee in order to be a good log for several process mining tasks, is that in different cases, different paths or different constraints should be activated, activities should occur in different order, data values should vary. Indeed, for instance, it would be not very effective applying discovery techniques to an event log in which all cases are actually composed of the same sequence of activities, i.e., in which all cases belong to the same process variant, while other process paths are neglected. Similarly, applying predictive process monitoring techniques that leverage data payloads to event logs in which each variable conveys the same value across all the cases of the log would produce rather inaccurate results without exercising the technique.

ALGen, as other tools do for procedural models [4], is able to address this challenge. For instance, by providing different types of heuristics (e.g., the *inter-trace* and the *data domain coverage*), it is able to generate positive event logs starting from declarative constraints with a good diversity of the generated traces, thus enabling the evaluation of declarative process discovery algorithms [15]. Furthermore, its capability of dealing with both positive and negative traces makes it suitable for generating event logs to be used for outcome-based predictive process monitoring approaches [9] demanding for positive and negative examples.

4.2 Closing the loop: a more and more refined model

As a side effect, the event log generators, and in particular ALGen, can be used for the continuous refinement of process models mined with discovery techniques. Existing (declarative) process discovery approaches can indeed be used to close the loop so as to provide an iterative methodology as in typical software engineering processes. The idea of the methodology is offering users the possibility to guide the discovery of process models by analyzing positive and negative cases generated according to the current model. At each step of the methodology:

1. positive and negative event logs are generated from an initial (declarative) model (empty, manually drawn or discovered from a log);
2. the generated event logs are analyzed, so as to evaluate whether cases in the positive log can be excluded and whether negative cases should be included;
3. based on the analysis carried out, the model can be changed (manually) or rediscovered from the positive event log;

The procedure can be iterated until no more cases to discard are contained in the positive log and no more cases to take are contained in the negative logs or until a maximum number of iterations is reached. By assuming that the user knows the actual process, at each iteration the methodology allows for discovering a process model which is closer to the behavior of the actual process.

ALGen and its heuristics can be used for applying this methodology. For instance, we can consider the Declare model discovered by *DeclareMiner* plug-in starting from the event log generated by ALGen in Section 3.3. In such a setting, if the user is interested in generalizing the model (e.g., to obtain a model closer to the initial one), she can remove traces from the log or manually constrain further the discovered model, so as to exclude the too specific behaviors in the positive log (e.g., the `alt. precedence` between `SubmitLoanApplication` and `AssessApplication`), and regenerate the log.

5 Discussion and Conclusions

The paper shows how the Abductive Logic Programming log generator ALGen can be used to produce good-quality positive and negative event logs starting from declarative constraints, so as to address the quest for the "perfect" log of process mining tasks.

A number of aspects, however, are still to be investigated. First of all, we plan to compare the logs generated through ALGen with logs generated by other approaches such as [4], from both the qualitative and quantitative perspectives. Second, in Section 3.3 we just showed the outcome obtained by applying the ProM *DeclareMiner* mining tool to the generated log: however, a number of mining algorithms are available, and a comparison of the learned models starting from the same log might be of interest. Finally, we plan to understand how the negative traces would affect the mining algorithms, and to determine which characteristics of the generated logs impact (how and how much) different algorithms. In particular, given that the "space" of negative traces might be huge, a question arise if negatives traces should exhibit specific characteristics. An interesting outcome would be to determine guidelines on the tuning of the heuristic parameters for the generation of the "perfect" log for each specific task.

Acknowledgments. This research has partially been carried out within the Euregio IPN12 KAOS, which is funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects.

References

1. van der Aalst, W.M.P., Adriansyah, A., Alves de Medeiros, A.K., Arcieri, F., Baier, T., Blickle, T., Bose, R.P.J.C., van den Brand, P., Brandtjen, R., Buijs, J.C.A.M., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B.F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D.R., Gaaloul, W., van Geffen, F., Goel, S., Günther, C.W., Guzzo, A., Harmon, P., ter Hofstede, A.H.M., Hoogland, J., Ingvaldsen, J.E., Kato, K., Kuhn, R., Kumar, A., Rosa, M.L., Maggi, F.M., Malerba, D., Mans, R.S., Manuel, A., McCreesh, M., Mello, P., Mendling,

J., Montali, M., Nezhad, H.R.M., zur Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Pérez, H.S., Pérez, R.S., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K.D., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., Wynn, M.T.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) 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), https://doi.org/10.1007/978-3-642-28108-2_19

2. 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. pp. 359–371 (2016)
3. 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), http://doi.acm.org/10.1145/1380572.1380578
4. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: 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. pp. 1–6 (2016)
5. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016). https://doi.org/10.1016/j.eswa.2016.08.040
6. Chesani, F., Ciampolini, A., Loreti, D., Mello, P.: Abduction for generating synthetic traces. In: Teniente, E., Weidlich, M. (eds.) Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers. Lecture Notes in Business Information Processing, vol. 308, pp. 151–159. Springer (2017). https://doi.org/10.1007/978-3-319-74030-0_11, https://doi.org/10.1007/978-3-319-74030-0_11
7. Chesani, F., Mello, P., De Masellis, R., 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. **161**(1-2), 75–111 (2018). https://doi.org/10.3233/FI-2018-1696, https://doi.org/10.3233/FI-2018-1696
8. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: EOMAS at CAiSE (2015)
9. Di Francescomarino, C., Dumas, M., Federici, M., Ghidini, C., Maggi, F.M., Rizzi, W., Simonetto, L.: Genetic algorithms for hyperparameter optimization in predictive business process monitoring. Inf. Syst. **74**(Part), 67–83 (2018). https://doi.org/10.1016/j.is.2018.01.003
10. Günther, C.: Process mining in flexible environments. Ph.D. thesis, Department of Industrial Engineering & Innovation Sciences (2009). https://doi.org/10.6100/IR644335, proefschrift.
11. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992), https://doi.org/10.1093/logcom/2.6.719
12. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Business Process Management - 13th International Conference, BPM

2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings. pp. 297–313 (2015). https://doi.org/10.1007/978-3-319-23063-4_21

13. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
14. Loreti, D., Chesani, F., Ciampolini, A., Mello, P.: Generating synthetic positive and negative business process traces through abduction. Knowledge and Information Systems **Accepted for publication** (2019)
15. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE (2012)
16. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, Lecture Notes in Business Information Processing, vol. 56. Springer (2010), https://doi.org/10.1007/978-3-642-14538-4
17. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. Ph.D. thesis, Eindhoven University of Technology (2008)
18. Skydanienko, V., Di Francescomarino, C., Ghidini, C., Maggi, F.M.: A tool for generating event logs from multi-perspective declare models. In: Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018. pp. 111–115 (2018)