Information Systems 64 (2017) 425–446

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Resolving inconsistencies and redundancies in declarative process models

Claudio Di Ciccio[a,*], Fabrizio Maria Maggi[b], Marco Montali[c], Jan Mendling[a]

[a] Vienna University of Economics and Business, Austria
[b] University of Tartu, Estonia
[c] Free University of Bozen-Bolzano, Italy

ARTICLE INFO

Article history:
Received 15 December 2015
Revised 20 September 2016
Accepted 21 September 2016
Available online 30 September 2016

Keywords:
Process Mining
Declarative Process
Conflict Resolution
Redundant Constraints

ABSTRACT

Declarative process models define the behaviour of business processes as a set of constraints. Declarative process discovery aims at inferring such constraints from event logs. Existing discovery techniques verify the satisfaction of candidate constraints over the log, but completely neglect their interactions. As a result, the inferred constraints can be mutually contradicting and their interplay may lead to an inconsistent process model that does not accept any trace. In such a case, the output turns out to be unusable for enactment, simulation or verification purposes. In addition, the discovered model contains, in general, redundancies that are due to complex interactions of several constraints and that cannot be cured using existing pruning approaches. We address these problems by proposing a technique that automatically resolves conflicts within the discovered models and is more powerful than existing pruning techniques to eliminate redundancies. First, we formally define the problems of constraint redundancy and conflict resolution. Second, we introduce techniques based on the notion of automata-product monoid, which guarantees the consistency of the discovered models and, at the same time, keeps the most interesting constraints in the pruned set. The level of interestingness is dictated by user-specified prioritisation criteria. We evaluate the devised techniques on a set of real-world event logs.

1. Introduction

The automated discovery of processes is the branch of the process mining discipline that aims at constructing a process model on the basis of the information reported in event data. The underlying assumption is that the recorded events indicate the sequential execution of the to-be-discovered process activities. The compact and correct representation of the behaviour observed in event data is one of the major concerns of process mining. Process discovery algorithms are classified according to the type of process model that they return, i.e., either procedural or declarative. Procedural process discovery techniques return models that explicitly describe all the possible executions allowed by the process from the beginning to the end. The output of declarative process discovery algorithms consists of a set of constraints, which exert conditions on the enactment of the process activities. The possible executions are implicitly established as all those ones that respect the given constraints. Mutual strengths and weaknesses of declarative and procedural models are discussed in [1,2].

One of the advantages of procedural models such as Petri nets is the rich set of formal analysis techniques available. These techniques can, for instance, identify redundancy in terms of implicit places or inconsistencies like deadlocks [3]. In turn, similar facilities are not provided for novel declarative modelling languages like DECLARE. This is a problem for several reasons. First, we are currently not able to check the consistency of a generated constraint set. Many algorithms that generate DECLARE models include in the output those constraints that are individually satisfied in the log in more than a given number of cases. The interaction of returned constraints is thereby neglected, with the consequence that subsets of constraints can end up contradicting one another. Second, it is currently unclear whether a given constraint set is free of redundancies. Since there are constraint types that imply one another, it is possible that the generated constraint sets are partially redundant. The lack of formal techniques for handling these two issues is unsatisfactory from both a research and a practical angle. This is also a roadblock for conducting fair comparisons in user experiments when a Petri net without deadlocks and implicit

* Corresponding author. Tel.: +43 1 31336 5222.
E-mail addresses: claudio.di.ciccio@wu.ac.at (C. Di Ciccio), f.m.maggi@ut.ee (F.M. Maggi), montali@inf.unibz.it (M. Montali), jan.mendling@wu.ac.at (J. Mendling).

http://dx.doi.org/10.1016/j.is.2016.09.005
0306-4379/

places is compared with a constraint set of unknown consistency and redundancy-freedom.

In this paper, we address the need for formal analysis of DECLARE models. We define the notion of an *automata-product monoid* as a formal notion for analysing consistency and local minimality, which is grounded in automata multiplication. Based on this structure, we devise efficient analysis techniques. Our formal concepts have been implemented as part of a process mining tool that we use for our evaluation. By analysing event log benchmarks, we are able to show that inconsistencies and redundancies occur in process models automatically discovered by state-of-the-art tools. First, our technique can take such process models as input and return constraints sets that are consistent. To this end, contradictory subsets are identified and resolved by removing the constraints generating the conflict. Second, our technique eliminates those constraints that do not restrict the behaviour of the process any further, i.e., that do not convey any meaningful information to the output. As a consequence, the returned sets are substantially smaller than the ones provided by prior algorithms, though keeping the expressed behaviour equivalent to the inconsistency-free process. This paper extends the research presented in our former publication [4] with a complete and self-consistent definition of the adopted formal concepts and algorithms. We also provide alternative strategies to be utilised during the redundancy and consistency check, so as to allow for different criteria to prioritise the constraints during the pruning phase. This is of crucial importance, since manipulating a declarative process model towards removal of inconsistencies and redundancies is intrinsically expensive from a computational point of view. Furthermore, we introduce a complementary technique to further reduce the number of redundancies in the models after the first check. Finally, we broadly extend the evaluation with an analysis of our implemented approach over real-world data sets including the event logs provided for the former editions of the BPI challenge.

The paper is structured as follows. Section 2 illustrates intuitively the problems that we tackle with the proposed research work. Section 3 describes the preliminary notions needed to formally contextualise the challenged issues. Section 4 formally specifies the problems of inconsistencies and redundancies in detail. Section 5 defines our formal notion of automata-product monoid, which offers the basis to formalise the techniques for consistency and redundancy checking. Section 6 illustrates the results of our evaluations based on real-world benchmarking data. Section 7 discusses our contributions in the light of related work. Finally, Section 8 concludes the paper.

2. Motivation

Declarative process models consist of sets of constraints exerted on tasks, which define the rules to be respected during the process execution. A well-established language for modelling declarative processes is DECLARE [5,6]. DECLARE defines a set of default *templates*, which are behavioural rules that refer to parameters in order to abstract from tasks. In DECLARE, e.g., *Init*(x) is a template imposing that a given parametric task x must be the one with which every process instance starts. *End*(x) specifies that every process instance must terminate with the given task x. *Response*(x, y) states that if task x is carried out, then task y must be eventually executed afterwards. *Precedence*(x, y) imposes that y can only be performed if x has been previously executed.

Let us consider a simple example process having three tasks, a, b, and c. By indicating the execution sequence of tasks with their name, possible enactments that fulfil a process model consisting of *Init*(a) and *End*(c) are: (*i*) abababc, and (*ii*) ababac. If we consider an event log made of the aforementioned execution sequences and use any declarative discovery algorithm to reveal a declarative process model that could have generated them, it would correctly return a set of constraints including *Init*(a) and *End*(c) because they are always satisfied. However, the set of constraints would include also (1) *Precedence*(a, b) and (2) *Precedence*(a, c), as well as (3) *Response*(a, c) and (4) *Response*(b, c): those four constraints hold true in the event log as well. Nevertheless, if a is already bound to be the first task to be carried out in every process instance (*Init*(a)), clearly no other task can be executed if a is not done before. Therefore, the first two constraints can be trivially deduced by *Init*(a). They add no information, yet they contribute to uselessly enlarge the set of constraints returned to the user as the outcome of the discovery. By the same line of reasoning, the third and fourth constraints are superfluous with respect to *End*(b). Intuitively, this example outlines the problem of *redundancy*, which is one of the two challenges that we tackle with this research work: the objective is to remove from the set of constraints in the discovered process model those ones that do not add information, i.e., that are not restricting the process behaviour any further given the remaining ones.

In the context of declarative process discovery, event logs can be affected by recording errors or report exceptional deviations from the usual enactments [7]. In such cases, constraints that were originally part of the process may be violated in some of the recorded executions. If discovery algorithms take into account only those constraints that always hold true in the event log, a minimum amount of noise might already cause several constraints to be discarded from the returned set [8–10]. To circumvent this issue, declarative discovery algorithms offer the possibility to tune a so-called *support threshold*: it specifies the minimum fraction of cases in which a constraint is fulfilled within the event log to let such constraint be included in the discovered model. However, this comes at the price of possibly having conflicts in the model though: constraints that hold true in a fraction of the event log above the set threshold can contradict other constraints. In such a case, the model becomes unsatisfiable, i.e., it exerts conditions that cannot be met by any possible execution. Such a model would clearly be to no avail to the discovery intents. This issue outlines the problem of *inconsistencies* in the discovered model, which we challenge in this research paper.

The aim of the presented approach is therefore twofold: given a discovered declarative process model, we want to (1) remove its inconsistencies, and (2) remove its redundancies. To pursue these objectives, we aim at keeping the process behaviour as similar as possible to the original one when removing inconsistencies, and retaining the minimum number of constraints that still represent the same original behaviour while getting rid of the redundancies. The number of combinations of constraints to test for the optimum of both problems is not tractable in practice, because every subset of the original constraints set should be confronted with the others. Our solution instead requires a polynomial number of checks over constraints to provide a sub-optimal yet effective solution. Furthermore, different criteria can be adopted to express (1) the desired behavioural closeness and (2) the preferability of constraints to be retained. To this extent, our solution envisages (1) the relaxation of conditions exerted by the contradicting constraints and (2) different ranking criteria for constraints, respectively.

3. Declarative process modelling and mining

This section defines the formal background for our research problem. In particular, we introduce and revisit the concepts of event logs and of declarative process modelling and mining.

Notational conventions. We adopt the following notations. Given a set X, (*i*) the multi-set of X is denoted as $\mathbb{M}(X)$; (*ii*) the