POD - A Tool For Process Discovery Using Partial Orders and Independence Information

Hernán Ponce-de-León[1], César Rodríguez[2], and Josep Carmona[3]

[1] Helsinki Institute for Information Technology HIIT and Department of Computer Science and Engineering, School of Science, Aalto University, Finland
[2] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, France
[3] Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. Current process discovery techniques focus on the derivation of a process model on the basis of the activity relations extracted from an event log. However, there are situations where more knowledge can be provided to the discovery algorithm, thus alleviating the discovery challenge. In particular, the (partial) characterization of the independence or concurrency between a pair of activities may be well-known. In this paper we present POD, a tool for discovery of Petri nets that can incorporate this type of additional information. We believe requiring independence/concurrency information is very natural in many scenarios, e.g., when there is some knowledge of the underlying process. We show with one example how this extra information can effectively deal with problems such as log incompleteness.

1 Introduction

In the last decade several algorithms have been proposed to discover a Petri net from an event log [1]. Petri nets are indeed a formalism well-suited for capturing concurrency or distribution, and are often employed as sound mathematical representations of processes tailored for formal analysis. Remarkably, they can be translated to user-friendly formalisms such as BPMN. However Petri-net-based process analysis is not widespread in industrial practice, where automata-based formalism are often instead preferred. These approaches, however, suffer from the well-known *state-explosion problem* in the presence of concurrency or distribution.

One of the reasons for this low-key industrial adoption of Petri nets, we believe, is that many existing algorithms infer the concurrency relations between transitions exclusively from the information present in the log. This poses two problems. First, the quality of this inference deteriorates with log *incompleteness*, i.e., when the log does not contain enough information to extract the component's distribution in the architecture of the process. Second, from an information-theoretical perspective, the log simply contains no information to distinguish non-deterministic actions from truly concurrent behaviour, which

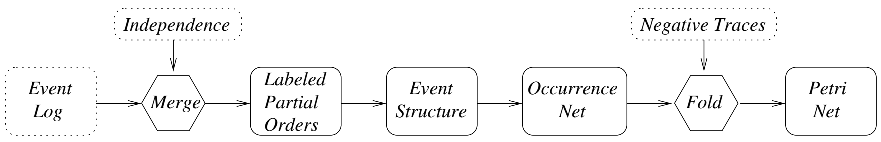

Fig. 1. Unfolding-based process discovery.

implies that most of the aforementioned algorithms can at best be based on probabilistic analysis of limited reliability.

To fill this gap, we present POD, a tool that incorporates this crucial user input: a so-called *independence relation*, i.e., an explicit, user-provided description of the concurrency relations between the system activities. One example could be the different tests that a patient should undergo in order to have a diagnosis: blood test, allergy test, and radiology test. Specifically, this input is provided under the form of an irreflexive, symmetric binary relation on the set of system actions.

One of the few works that considers the concurrency or distribution of the system as an input is [2]. They investigate the synthesis of a Petri net from a set of partial orders and rely on ad-hoc operators tailored to compose them (choice, sequentialization, parallel compositions and repetition). Since the operators may in practice introduce wrong generalizations, a domain expert is consulted for the legality of every extra run. Our tool automates the generalization step based on different properties that the final model should satisfy (replay, preservation of concurrency, avoidance of undesired behaviors). For a detailed description of the theory underlying the POD tool, the reader can refer to [3].

2 Tool Description

The approach of our tool is summarized in Figure 1. Starting from an event log and a concurrency relation on its set of activities, we construct a collection of labeled partial orders whose linearizations include both the sequences in the log as well those that could be obtained via successive permutations of concurrent activities. We then merge this collection into an event structure [4] which we next transform into an occurrence net (acyclic Petri net) representing the same behavior. Finally, we perform a controlled generalization step by selectively folding the occurrence net into a Petri net. This step yields a net that (a) can execute all traces contained in the event log, and (b) generalizes the behavior of the log in a controlled manner. The folding process is driven by a *folding equivalence relation* [3], which we synthesize using SMT (*Satisfiability Modulo Theories*, see [5]). Different folding equivalences guarantee different properties about the final net. In [3] we propose three different classes of equivalences to (a) preserve all sequential executions of the log, (b) to additionally preserve the concurrency or distribution stated in the independence relation and (c) to avoid re-introducing negative (or forbidden) traces (this feature is still unimplemented in POD).

2.1 Architecture and Maturity

POD is a command-line tool implemented in Python. It interacts with the Z3 SMT-solver [6] for computing the aforementioned folding equivalences. It is currently under development and will evolve to a more stable version in the near future. Currently, it can be seen as a prototype testbed for process discovery based on partial orders, independence information, and generalization based on folding equivalences. It has been tested with only small to medium-size examples.

The most prominent feature in POD is the ability to generate fully-fitting nets whose transitions exhibit exactly the same independence (concurrency) than the one stated by the independence relation given as input (POD can also relax this controlled generalization, i.e., generalizing more). While the underlying theory [3] allows for the discovery of more than one transition per log activity, POD merges all associated events into one single transition (to simplify the implementation). This ensures a minimum number of final transitions, but POD could sometimes be unable to find a suitable equivalence (unsatisfiable SMT encoding). Since the number of transitions in the folded net is fixed, it turns out that the quality (in terms of precision and generalization) of the mined model increases as one increases the number of places (which the user can conveniently control).

2.2 Download, Formats, Usage

POD is available from `http://github.com/cesaro/pod/releases/tag/v0.1`. It has been tested in MAC and Linux and the installation requires simply to unpack the files into a directory. Once unpacked, `cd` to it and run it as `./src/pod.py`. The tool can also perform various handy tasks related to process discovery, such as extracting a random log from a Petri net, dumping statistics of a net, a log, etc.

POD reads and writes logs in XES format and Petri nets in PNML format. As for the independence relation, it expects, for the time being, a plain-text file where every line defines one pair of *dependent* transitions (i.e., the file contains the complement of an independence relation, usually smaller), cf. POD's website.

Assume that we have a log file `log.xes` and a dependency file `dep.txt`. We instruct POD to do process discovery:

```
./src/pod.py discover log.xes dep.txt --out result.pnml
```

By default POD will output the occurrence net mentioned above, i.e., it will perform no generalization at all. In other words, it will use the identity relation as a folding equivalence, cf. [3]. Option `--eq` instructs POD to use a different folding equivalence (i.e., more interesting generalizations). The most important values are `--eq=sp-smt` and `--eq=ip-smt`. Under these two options, POD will in fact not use a predefined folding equivalence, but will rather use SMT solving to synthesize one from the log and the independence relation:

– *Option `sp-smt` (sequence, or fitness, preservation).* The synthesized folding equivalence will merge all events with equal label into one single transition. It will also merge the presets of any two merged events. Options `--smt-{min,max}-places` restrict the number of generated places.

– *Option* `ip-smt` *(independence preservation).* In addition to preserving all sequences of the log in the final net, the synthesized equivalence will preserve the concurrency expressed by the independence relation. Accepts roughly the same options as `sp-smt`.

3 Example

Consider the net in Figure 2 (a), and consider the following log containing only two of its executions (which however fire all possible eight transitions):

```
1, 3, 5, 6, 2, 3, 4, 6, 7
1, 3, 5, 6, 8
```

While one could argue that this is a very incomplete log, we show that POD can in fact *reconstruct* Figure 2 (a) using this log plus some independence relation.[4]

We supply POD with the *best* independence relation that an expert would provide. It coincides with the independence relation of the original net. For instance, transitions 1 and 8 are independent, as one can never interfere (neither make possible nor disable) the firing of the other. Similarly, 3 and 4, or even 1 and 2 are independent. Transitions, 4 and 5, or 1 and 3 are however dependent.

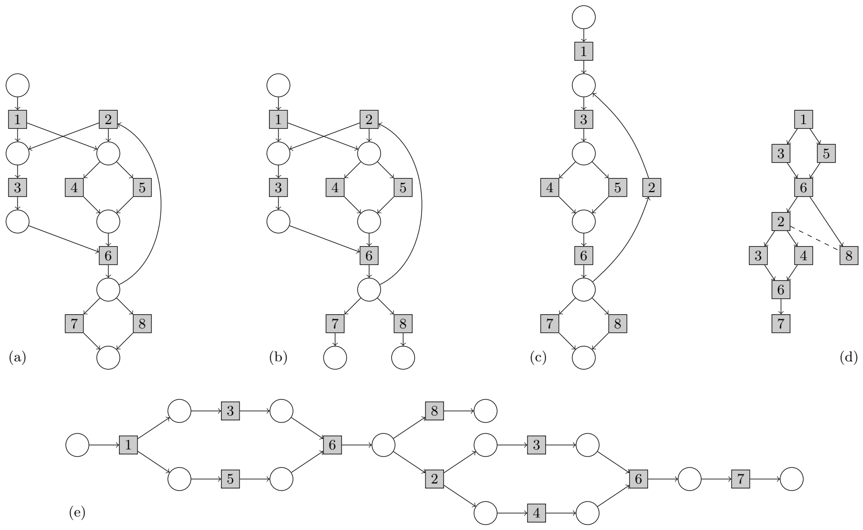

Fig. 2. (a) original Petri net; (b) net mined with POD; (c) net mined with ILP Miner; (d) and (e): event structure and occurrence net internally constructed by POD.

When we run POD with these two inputs (log and independence relation), it will internally construct the event structure depicted in Figure 2 (d). This object tracks in a compact, concurrency-aware fashion the dependencies and

[4] Go to `http://lipn.fr/~rodriguez/exp/bpm15/` to reproduce this experiment.

conflicts between the occurrences of transitions (events) of the net, as recorded in the log. POD next transforms it into Figure 2 (e), an occurrence net denoting exactly the same behaviour as the event structure. This net is a fitting and 100% precise description of the log (all executions in it but not in the log are *provably* executions of the original system). Observe that the two events labelled by 3 and 5 (immediately after 1) are concurrent, as requested by the input independence relation. Observe also that there is two occurrences of transition 6 and that this is somehow an unwinded version of the original net.

Finally POD will *fold* this occurrence net and output the result, depicted in Figure 2 (b), which is identical to the original except for the output places of 7 and 8. We asked POD to use an independence-preserving (option `--eq=ip-smt`) folding, meaning that the result net should exhibit the same concurrency information expressed in the input independence relation.

By contrast, ILP Miner [7], which only uses the log, produced Figure 2 (c). Observe that it was unable to infer the concurrency between 3 and 4 or 5 as in all log traces the latter transitions occur always after 3. Interestingly, for the same reason it was also unable to infer the direct causality between 3 and 6. This is an artifact of using the *theory of regions* on incomplete logs, that derive the most precise model which then fails at incorporating unseen behavior.

4 Significance for the BPM Field

Process discovery is recognized as one of the key enablers of BPM. However, problems like log incompleteness, as exemplified above, or noise hamper significantly the success of process discovery techniques in industrial scenarios. We believe that the development of tools like POD, which let the user incorporate knowledge that can be used to soundly cope with these challenges, can fill the gap between academic tools and industrial acceptance of process discovery.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Construction of process models from example runs. Tr. Petri Nets and Other Models of Conc. **2** (2009) 243–259
3. Ponce-de-León, H., Rodríguez, C., Carmona, J., Heljanko, K., Haar, S.: Unfolding-based process discovery. CoRR **abs/1507.02744** (2015)
4. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13** (1981) 85–108
5. de Moura, L.M., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Commun. ACM **54**(9) (2011) 69–77
6. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS. Volume 4963 of LNCS., Springer (2008) 337–340
7. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process Discovery Using Integer Linear Programming. In: Proceedings of the 29th International Conference on Applications and Theory of Petri Nets, Springer-Verlag (2008) 368–387