Business Process Conformance Checking Based on Event Structures

Luciano García-Bañuelos[1], Nick R.T.P. van Beest[2,3], Marlon Dumas[1] and Marcello La Rosa[3,2]

[1] University of Tartu, Estonia
{luciano.garcia, marlon.dumas}@ut.ee
[2] NICTA, Australia
nick.vanbeest@nicta.com.au
[3] Queensland University of Technology, Australia
m.larosa@qut.edu.au

1 Introduction

This paper addresses the problem of business process conformance checking defined as follows: Given an event log recording the actual execution of a business process, and given a process model capturing its expected or normative execution, describe the differences between the behavior captured in the event log and that captured in the process model. In this setting, a log consists of a set of traces, where each trace is a sequence of events. An event refers to the execution of an activity in the process.

This problem has been approached using *replay* [4] and *trace alignment* [7]. Replay takes as input one trace at a time and determines the maximal prefix of the trace (if any) that can be parsed by the model. When it is found that a prefix can no longer be parsed by the model, error-recovery techniques are used to correct the parsing error and continue parsing as much as possible the remaining input trace. Trace alignment identifies, for each trace in the log, the closest corresponding trace(s) produced by the model and then highlights the points where the trace and the model diverge. However, trace alignment cannot characterize the exact differences observed in a given state of the process in a concise and understandable way, particularly for processes with a large number of possible traces.

In this abstract, we outline a method that, given a process model and an event log, returns a set of statements in natural language describing all the behavior observed in the log but not allowed by the process model (and vice versa). The method relies on a well-known model of concurrency, namely prime event structures. We show that the stated problem of conformance checking can be approached by folding the input event log into an event structure, unfolding the process model into another event structure, and comparing the two event structures via an error-correcting synchronized product.

2 Approach

The overall approach is depicted in Figure 1. In this section we describe each of the steps in turn.

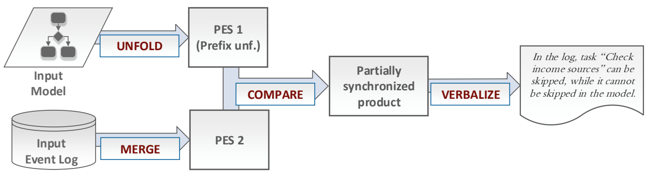


Figure 1: Overall approach

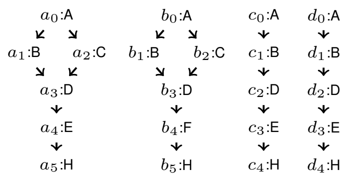
From a log to a prime event structure

In previous work [5], we presented a method to generate a prime event structure (PES) from an event log. The method consists of two steps. First the event log, seen as a set of traces, is transformed into a set of runs by invoking a *concurrency oracle*. In essence, each trace is turned into a run by relaxing the total order induced by the trace into a partial order such that two events are not causally related if the concurrency oracle has determined that they occur concurrently. Existing concurrency oracles such as those proposed in the α process mining algorithm [8] or in [1] can be used for this purpose.

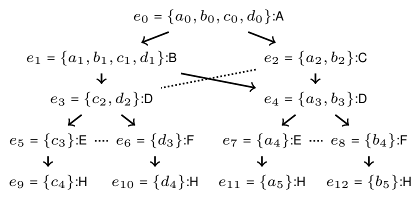
Second, the runs are merged into an event structure in a lossless manner, which means that the set of maximal configurations of the event structure is equal to the set of runs. For example, consider the log in Figure 2(a), consisting of 16 traces: 3 instances t_1 (cf. column "N"), 3 instances of t_2, so on. Using the concurrency oracle of the α algorithm, we conclude that event classes B and C are concurrent, so that the set of runs in Figure 2(b) can be constructed. The notation e:A indicates that event e represents an occurrence of event class A in the original log. By merging events with the same label and the same history (i.e. same prefix), we obtain the PES in Figure 2(c). In this figure, the notation $\{e_1, e_2 \ldots e_i\}$:A indicates that events $\{e_1, e_2 \ldots e_i\}$ represent occurrences of event class A in different runs.

Trace	Ref	N
A B C D E H	t_1	3
A B C D F H	t_2	3
A C B D E H	t_3	2
A C B D F H	t_4	2
A B D E H	t_5	3
A B D F H	t_6	3

(a) Event log



(b) Runs



(c) Induced PES

Figure 2: Example of construction of a PES from a set of traces

From a model to a prime event structure

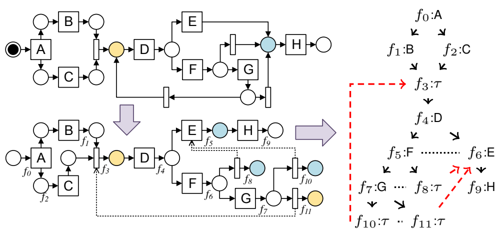
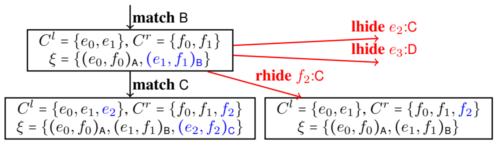


Figure 3: From a workflow net to a PES prefix

The proposed method takes process models as input, which are captured as Workflow nets (WF-nets) [6], i.e. Petri nets with a single start and a single end place such that every node is on a path from the start to the end. Mappings from common process modeling notations (e.g. BPMN) to WF-nets have been defined in the literature [2]. Event structures can be losslessly derived from workflow nets via unfoldings of Petri nets using well-known unfolding techniques. In the case of acyclic nets, a full unfolding can be computed and a PES can be trivially derived. In the case of bounded Petri nets with cycles, it is possible to calculate a finite prefix unfolding that captures all the behavior in the original net. A PES (prefix) can then be derived from such prefix unfolding. Several prefix unfoldings have been defined in the literature, e.g. the *complete prefix unfolding* [3].

Comparing prime event structures



match B
$C^l = \{e_0, e_1\}, C^r = \{f_0, f_1\}$
$\xi = \{(e_0, f_0)_A, (e_1, f_1)_B\}$
lhide e_2:C
lhide e_3:D
rhide f_2:C
match C
$C^l = \{e_0, e_1, e_2\}, C^r = \{f_0, f_1, f_2\}$
$\xi = \{(e_0, f_0)_A, (e_1, f_1)_B, (e_2, f_2)_C\}$
$C^l = \{e_0, e_1\}, C^r = \{f_0, f_1, f_2\}$
$\xi = \{(e_0, f_0)_A, (e_1, f_1)_B\}$

Figure 4: Fragment of PSP of $\mathcal{E}^1$ and $\mathcal{E}^2$

The comparison of event structures is performed by means of an error-correcting synchronized product that we call a *partially synchronized product* (PSP). A PSP is built starting from empty configurations. At each step, a pair of events from each PES is matched if and only if their labels and causal order are consistent. Every unmatched event is "hidden" to let the simulation proceed. By using a heuristic search, we construct a PSP that contains the set of optimal matchings for every runs in the event log PES. Fig. 4 presents an excerpt of the PSP of the events structures from Fig. 2 and 3. The box on top corresponds to the state where configurations $C^l = \{e_0, e_1\}$ (log PES) and $C^r = \{f_0, f_1\}$ (model PES) have been processed, resulting in the matching $\{(e_0, f_0)_A, (e_1, f_1)_B\}$. Given the above state, the events $\{e_2$:C$, e_3$:D$\}$ from log PES would be enabled, and so is f_2:C from the other PES. Thus, four possible moves are possible in the PSP: (i) the matching of events e_2 and f_2, both carrying the label C, (ii) the (left) hiding of f_2:C, and (iii) the (right) hiding of e_2:C and e_3:D. Interestingly, the fragment above alone captures the fact that *"In the event log, task* C *can be skipped, while in the model it cannot"*. Although not illustrated, the PSP supports special-purpose moves to operate with PES prefixes.

Verbalizing differences

We identify three categories of differences. The first type of difference is the one in which the labels of events confined in the mismatch context can be paired. The mismatch stems from differences in the underlying behavior relations, e.g. parallel vs. sequential relations. The second type concerns composite mismatches, in which the information of two branches in the PSP needs to be combined in order to diagnose the difference. For example, a "task skipping" is detected when an event is hidden in one branch of the PSP and there exists a sibling node where the same event is matched, and both nodes share the same parent node. The third type concerns differences comprising non-observed behavior. For instance, the model describes a cycle consisting of a set of events that cannot be found in the log, or vice versa.

Using the PSP, each change from a particular category can be verbalized to describe the exact difference between the observed log and the model. The different operations in the PSP uniquely describe the differences between the model and the log. Table 1 provides an example of the verbalization of differences with the model in Fig. 3 for each change category.

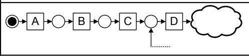
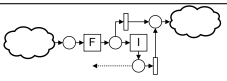
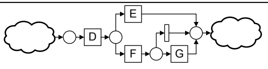
Difference	**Log compared to Fig. 3**	**Verbalization**
Type 1		In the model, B and C are in parallel, while in the log, B precedes C.
Type 2		In the log, G is substituted by I.
Type 3		In the model, the interval [D, F, G] is repeated multiple times, while in the log it is not.

Table 1: Desired verbalization of differences for each change category.

For Type 1, the PSP describes a series of mismatches concerning the same event for both PESs. When considering the context in the PESs, it becomes evident from the ordering relations that in one PES B is in parallel with C, while in the other PES B and C are causal. In the Type 2 example, on one side G is hidden, while on the other side I is hidden. From this, it can be concluded that G has been replaced with I. Finally, Type 3 concerns behavior specified in the model that is not observed in the event log. A mismatch in this category can be identified by identifying sets of events that are not present in the PSP. Of particular interest are cutoff events because they provide hints about the nature of the missing behavior. For the example shown in Table 1, the cutoff event f_{10}:τ would not be part of the PSP, the absence of which would reveal the fact that cycle [D, F, G] is never observed in the event log.

The examples of Table 1 show that we can characterize behavior in the log that is not in the model and vice versa in a way that is understandable to users. Using this method, the conformance of an event log to a model can be assessed more compactly and precisely than existing techniques.

References

[1] Jonathan E. Cook and Alexander L. Wolf. Event-based detection of concurrency. In *FSE*. ACM, 1998.

[2] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information & Software Technology*, 50(12), 2008.

[3] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design*, 20(3), 2002.

[4] Anne Rozinat and Wil M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1), 2008.

[5] Nick R.T.P. van Beest, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Log delta analysis: Interpretable differencing of business process event logs. In *BPM 2015*. Springer, 2015.

[6] Wil M.P. van der Aalst. Verification of workflow nets. In *Appl. and Theory of Petri Nets 1997*. Springer, 1997.

[7] Wil M.P. van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Min. Knowl. Discov.*, 2(2), 2012.

[8] Wil M.P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: discovering process models from event logs. *IEEE TKDE*, 16(9), 2004.