Process Mining and Verification of Properties: An Approach based on Temporal Logic

W.M.P. van der Aalst and H.T. de Beer and B.F. van Dongen

Department of Technology Management, Eindhoven University of Technology,
P.O.Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl

Abstract. Information systems are facing conflicting requirements. On the one hand, systems need to be adaptive and self-managing to deal with rapidly changing circumstances. On the other hand, legislation such as the Sarbanes-Oxley Act, is putting increasing demands on monitoring activities and processes. As processes and systems become more flexible, both the need for, and the complexity of monitoring increases. Our earlier work on *process mining* has primarily focused on process discovery, i.e., automatically constructing models describing knowledge extracted from event logs. In this paper, we present another approach supported by our ProM framework and based on both a standard XML format and temporal logic. Given an event log and a property, our LTL Checker verifies whether the observed behavior matches the (un)expected/(un)desirable behavior.

Key words: Process mining, temporal logic, business process management, workflow management, data mining, Petri nets.

1 Introduction

A constantly changing reality is forcing organizations and their information systems to adapt at an ever increasing pace. Business Process Management (BPM) and Workflow Management (WFM) systems increasingly allow for more flexibility. Instead of recoding the system it typically suffices to reconfigure the system on the basis of a process model [3]. Several researchers have addressed the problems related to workflow change [1, 14, 29, 30]. Although the work on workflow change is highly relevant, in reality many processes are not bound by a WFM or BPM system driven by an explicit process model. Some systems, e.g., the case handling system FLOWer, allow for implicit routing, other systems allow for much more behavior than desired. For example, people using the SAP R/3 system are not limited by process models described in the SAP R/3 Reference Model [24]. Deviations from the "normal process" may be desirable but may also point to inefficiencies or even fraud. New legislation such as the Sarbanes-Oxley (SOX) Act [34] and increased emphasis on corporate governance has triggered the need for improved auditing systems [22]. To audit an organization, business activities need to be monitored. As enterprises become increasingly automated, a tight coupling between auditing systems and the information systems supporting the operational processes becomes more important.

Today's information systems need to compromise between two requirements: (1) being adaptive and self-managing and (2) being able to be audited. Within the context of this struggle, we have developed a tool called *LTL Checker*. This tool has been developed in the context of the *ProM framework*[1]. The ProM framework offers a wide range of tools related to process mining, i.e., extracting information from event logs [6]. Process mining is motivated by the fact that many business processes leave their "footprints" in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), i.e., business events are recorded in so-called event logs. Until recently, the information in these logs was rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs, i.e., the basic idea of process mining is to diagnose business processes by mining event logs for knowledge.

The work presented in this paper is related to process mining, but, unlike most process-mining approaches, the emphasis is not on discovery. Instead we focus on *verification*, i.e., given an event log we want to verify certain properties. An example is the *4-eyes principle*, i.e., although authorized to execute two activities, a person is not allowed to execute both activities for the same case. For example, a manager may submit a request (e.g., to purchase equipment, to make a trip, or to work overtime) and he may also approve requests. However, it may be desirable to apply the 4-eyes principle implying that the manager is not allowed to approve his own request. If there is an event log recoding the events "submit request" and "approve request", the 4-eyes principle can be verified easily. More difficult are properties relating to the ordering or presence of activities. For example, activity A may only occur if it is preceded by activity B or activity C and immediately followed by activity D. Therefore, we propose an approach based on *temporal logic* [25, 28]. More specifically, we use an extension of *Linear Temporal Logic* (LTL) [16, 19, 20] tailored towards event logs holding information on activities, cases (i.e., process instances), timestamps, originators (the person or resource executing the activity), and related data.

This paper reports on the language developed to formulate properties in the context of event logs, the approach used to check these properties, the implementation in the ProM framework, and the relation between this work and process discovery. It is important to note that process discovery is difficult in situations where a lot of flexibility is offered. The approach based on verification is more robust because it can focus on the essential properties. Hence, the LTL Checker is a welcome addition towards a wider range of process mining tools.

This paper is organized as follows. Section 2 introduces a running example that will be used to illustrate the concept of process mining. The ProM framework and the XML format used to store event logs is presented in Section 3. Then

[1] Both documentation and software can be downloaded from www.processmining.org.

the LTL language is introduced and it is shown how properties can be specified. Section 5 shows how these properties can be verified using the LTL Checker in ProM. Finally, some related work is discussed and the paper is concluded.

2 Running example

Today, many enterprise information systems store relevant events in some structured form. For example, WFM systems typically register the start and completion of activities. ERP systems like SAP log all transactions, e.g., users filling out forms, changing documents, etc. Business-to-Business (B2B) systems log the exchange of messages with other parties. Call center packages but also general-purpose Customer Relationship Management (CRM) systems log interactions with customers. These examples show that many systems have some kind of *event log* often referred to as "history", "audit trail", "transaction log", etc. [6, 7]. The event log typically contains information about events referring to an *activity* and a *case*. The case (also named process instance) is the "thing" which is being handled, e.g., a customer order, a job application, an insurance claim, a building permit, etc. The activity (also named task, operation, action, or work-item) is some operation on the case. Typically, events have a *timestamp* indicating the time of occurrence. Moreover, when people are involved, event logs will typically contain information on the person executing or initiating the event, i.e., the *originator*. Based on this information several tools and techniques for process mining have been developed.

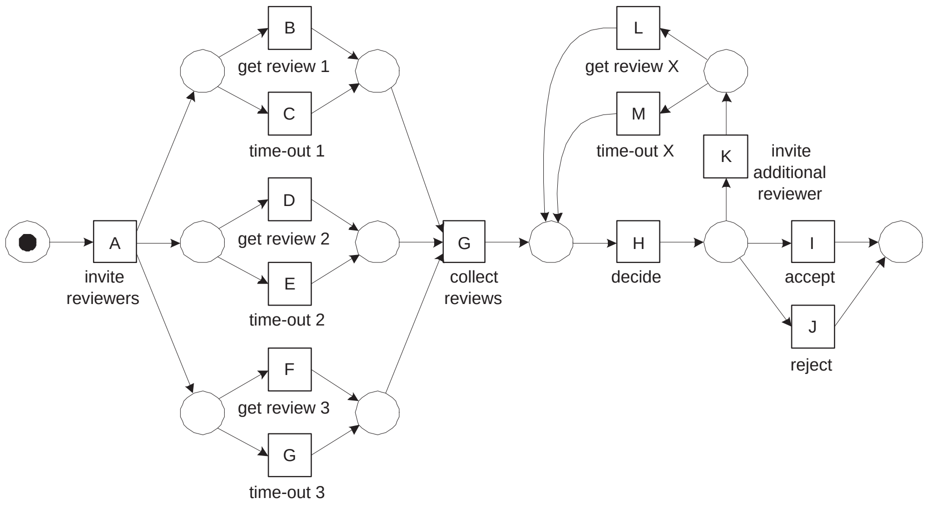


Fig. 1. Running example.

As a running example, we will use the process shown in Figure 1. The process describes the reviewing process of a paper for a journal and is represented in terms of a Petri net (more specifically a workflow net [3]). After inviting three reviewers (activity A) each of the reviewers returns a review or a time-out occurs, e.g., for the first review either B or C occurs. Then the reviews that were returned in time are collected (activity G) and a decision is made (activity H). There are three possible outcomes of this decision: (1) the paper is accepted (I), (2) the paper is rejected (J), or (3) an additional reviewer is needed (K). Similar to the original three reviewers, the additional reviewer may return the review in time (L) or not (M).

case id	activity id	originator	timestamp
...			
case_0	invite reviewers	John	2005-01-01T08:00
case_1	invite reviewers	John	2005-01-01T08:00
case_0	get review 1	Nick	2005-02-06T08:00
case_0	get review 2	Pete	2005-03-07T08:00
...			

Table 1. A fragment of the event log.

For the process shown in Figure 1, we may log events such as the ones shown in Table 1. As discussed before, each event refers to a case (e.g., case_1) and an activity (e.g., invite reviewers). Moreover, in this case the timestamp and originator are logged. The first line of the fragment shown in Table 1 states that John executed step A (invite reviewers) for case_0 on the first of January 2005. Table 1 does not show that for some events additional data is logged. For example, each case has a data element *title* and each review result (e.g., get review 1) has a *result* attribute which is either accept or reject. Table 1 only shows a fragment of the log used throughout this paper. The log holds information about 48 cases (i.e., papers) and 354 events and is used as a running example.

3 ProM Framework and XML Format

The LTL Checker presented in this paper is embedded in the ProM framework and should be seen as an addition to a collection of process mining tools. Therefore, we first describe the ProM framework and some of the process mining techniques that have been developed in this framework.

In Table 1 we showed a fragment of some event log. We assume a standard log format, named MXML, and have developed several adaptors to map logs in different information systems onto our log format (e.g., Staffware, FLOWer, MS Exchange, MQSeries, etc.). Figure 2 shows the hierarchical structure of MXML. The format is XML based and is defined by an XML schema (cf. www.processmining.org).

The ProM framework has been developed as a completely plug-able environment. It can be extended by simply adding plug-ins, i.e., there is no need to know or recompile the source code. Currently, more than 30 plug-ins have been added. The most interesting plug-ins are the mining plug-ins and the analysis plug-ins. The architecture of ProM allows for five different types of plug-ins:

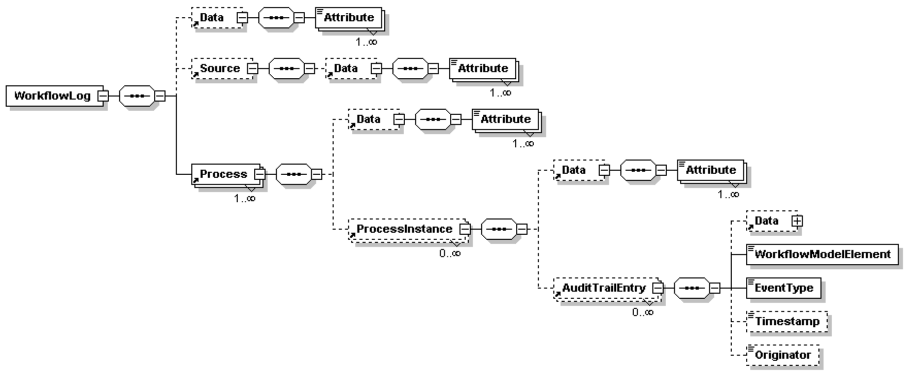


Fig. 2. XML schema for the MXML format used by ProM.

Mining plug-ins which implement some mining algorithm, e.g., mining algorithms that construct a Petri net based on some event log.

Export plug-ins which implement some "save as" functionality for some objects (such as graphs). For example, there are plug-ins to save EPCs, Petri nets, spreadsheets, etc.

Import plug-ins which implement an "open" functionality for exported objects, e.g., load instance-EPCs from ARIS PPM.

Analysis plug-ins which typically implement some property analysis on some mining result. For example, for Petri nets there is a plug-in which constructs place invariants, transition invariants, and a coverability graph.

Conversion plug-ins which implement conversions between different data formats, e.g., from EPCs to Petri nets.

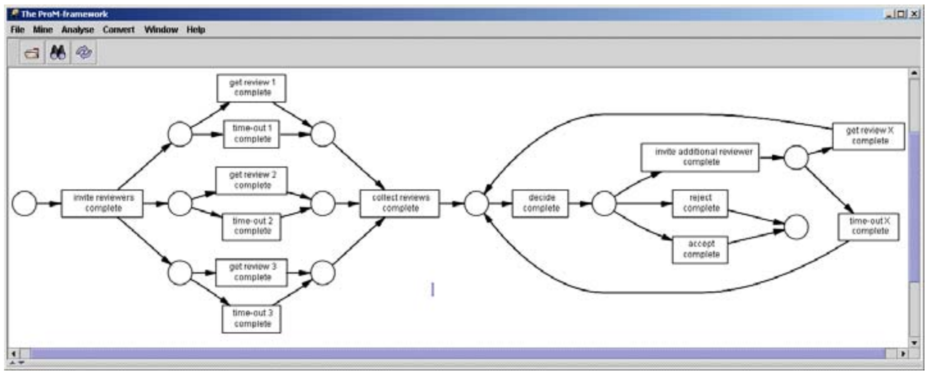
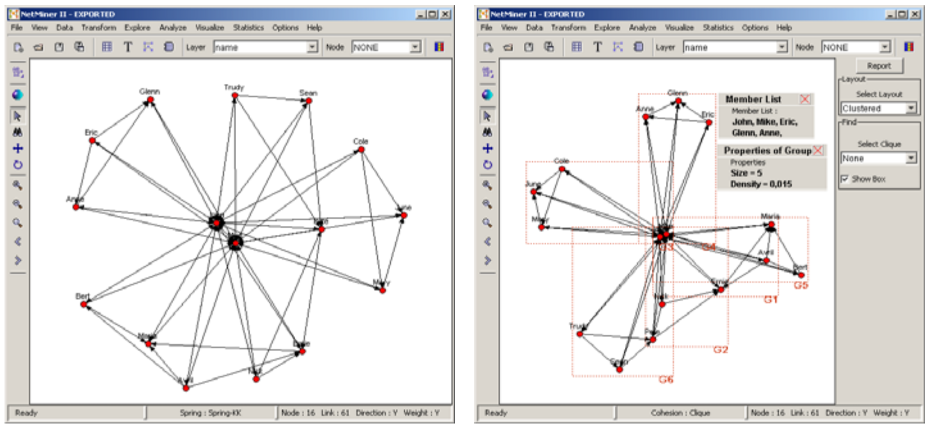


Fig. 3. Using the α mining plug-in in ProM we can discover the underlying process model.

For the process perspective, four mining plug-ins are available including the α plug-in [8], a genetic miner, and a multi-phase miner. The goal of these plug-ins is to extract a process model from a given event log without using any additional knowledge of the process. For example, based on the log mentioned in Section 2 (i.e., the log holding information on 48 papers and 354 events), the

α plug-in discovers the process model shown in Figure 3. Note that this model is identical to the one shown in Figure 1. Of course the layout is different since it is automatically generated. For the organizational/social perspective, one mining plug-in named MinSoN is available [5]. If we apply this plug-in to the same log, ProM constructs the social network shown in Figure 4. A social network shows all actors in the organization and their relationships. Based on an analysis of the log (e.g., transfer of work or similarity of work profiles), the relationships and their relative strength are derived. Figure 4 shows how these can be analyzed, e.g., using a tool like NetMiner. The screenshot on the left shows that John and Mike are the two central players in the reviewing process. This is no surprise since John is the editorial assistant (responsible for the contacts with reviewers and authors) and Mike is the editor of the journal. The screenshot on the right-hand-side of Figure 4 illustrates how NetMiner can discover "cliques" in a social network.



Member List
Member List :
John, Mike, Eric, Glenn, Anne,

Properties of Group
Properties
Size = 5
Density = 0,015

Fig. 4. The social network discovered by ProM can be exported to the SNA tool NetMiner.

Figures 3 and 4 show how process mining techniques can be used to discover models based on some event log. The results presented in this paper are related to process mining, but unlike the mining plug-ins mentioned the focus is not on discovery. Instead, the focus is on verification. Therefore, in the context of the ProM framework, the LTL Checker should be considered as an analysis plug-in rather than a mining plug-in.

4 Formulating Properties: The LTL Language

Assuming that the information system at hand left a "footprint" in some event log, it is interesting to check whether certain properties hold or not. Before being able to check such properties, a concrete language for formulating dynamic properties is needed. Given the fact that we consider behavioral properties where ordering and timing are relevant, some temporal logic seems to be the best basis to start from [25, 28]. There are two likely candidates: Computational Tree Logic (CTL) and Linear Temporal Logic (LTL) [16, 19, 20]. Given the linear nature of an event log, LTL is the obvious choice. It would only make sense to use CTL if first a model (e.g., an automaton) was built before evaluating the property. Unlike most of the existing process mining techniques supported in the ProM framework, we try to avoid this and simply use LTL as a basis directly working on the event log.

It is not sufficient to select LTL as a language. To easily specify properties in the context of MXML, a dedicated language is needed that exploits the structure shown in Figure 2. Therefore, in addition to standard logical operators, we need dedicated statements to address the properties of cases and events. For example, it should be easy to use the various attributes of a case (both standard ones such as case, activity, timestamp, originator and event type, and context specific ones such as data values).

We have developed a complete language including type definitions, renaming, formulas, subformulas, regular expressions, date expressions, propositional logic, universal and existentional quantification, and temporal operators such as nexttime ($\bigcirc F$), eventually ($\Diamond F$), always ($\Box F$), and until ($F \mathbf{U} G$). A complete description of the language is beyond the scope of this paper but is given in [10]. To illustrate the language we use the examples shown in Table 2.

The notation `ate.X` is used to refer to some attribute `X` of an audit trail entry (`ate`), i.e., an event in the event log. Similarly `pi.X` is used to refer to attribute `X` of a process instance (`pi`), i.e., a case. There are several predefined attributes, e.g., `ate.WorkflowModelElement` refers to the activity (or other process elements) being executed. `ate.Originator` is the resource executing it, i.e., the person. `ate.Timestamp` is the timestamp of the event. `ate.EventType` is the type of the event (i.e., schedule, assign, withdraw, start, complete, etc.). The three `set` declarations shown in Table 2 (lines 1-3) declare that `ate.WorkflowModelElement`, `ate.Originator`, and `ate.EventType` can be used for quantification, e.g., `ate.WorkflowModelElement` may refer to the activity related to the current event but may also be used to range over all activities appearing the a case. Line 5 declares the DateTime format used when specifying a value (note that this allows for shorter notations than the standard XML format). Line 6 declares a data attribute at the level of an event. The data attribute `result` is used to record the outcome of a single review in the running example. Line 7 shows a data attribute at the level of a case. Note that both attributes are of type `string`. To allow for shorter/customized names our language allows for renaming. As shown in lines 9-11, `ate.Originator`, `ate.Timestamp`, and `ate.WorkflowModelElement` are renamed to `person`, `timestamp`, and `activity` respectively. These names are used in the remainder of Table 2.

The goal of the LTL language is to specify properties. Properties are described using the `formula` construct. Formulas may be nested and can have parameters. To hide formulas that are only used indirectly, the `subformula` construct should be used. Table 2 describes five formulas and two subformulas. Lines 13-14 specify a formula without any parameters. The property holds for a given

```
set ate.WorkflowModelElement;
set ate.Originator;
set ate.EventType;

date ate.Timestamp := "yyyy-MM-dd";
string ate.result;
string pi.title;

rename ate.Originator as person;
rename ate.Timestamp as timestamp;
rename ate.WorkflowModelElement as activity;

formula accept_or_reject_but_not_both() :=
(<>(activity == "accept") <-> !(<>(activity == "reject")));

formula action_follows_decision() :=
[]( (activity == "decide" -> _O( ((activity == "accept" \/
activity == "reject") \/ activity == "invite additional reviewer") )));

subformula execute( p : person, a : activity ) :=
<> ( (activity == a  /\  person == p ) ) ;

formula not_the_same_reviewer() :=
forall[p:person |
(((!(execute(p,"get review 1")) \/ !(execute(p,"get review 2"))) /\
(!(execute(p,"get review 1")) \/ !(execute(p,"get review 3")))) /\
(!(execute(p,"get review 2")) \/ !(execute(p,"get review 3")))) ];

subformula accept(a : activity ) :=
<> ( (activity == a  /\  ate.result == "accept" ) ) ;

formula dont_reject_paper_unjustified() :=
(((accept("get review 1") /\ accept("get review 2")) /\
accept("get review 3"))
->  <> ( activity == "accept" ) );

formula started_before_finished_after(start_time:timestamp,
end_time:timestamp) :=
(<>( timestamp < start_time ) /\ <>( timestamp > end_time ));
```

Table 2. Some LTL formulas for the running example.

event log if for each paper there was an acceptance (activity I in Figure 1) or a rejection (activity J in Figure 1) but not both. To formulate this both temporal and logical operators are used: `<>` is the syntax for the temporal operator eventually ($\Diamond F$), `<->` denotes "if and only if", and `!` is the negation. Line 14 uses the shorthand `activity` defined in Line 11 twice. `activity == "accept"` is true if the `WorkflowModelElement` attribute of the current event points to the acceptance activity. Hence, `<>(activity == "accept")` holds if the acceptance activity was executed. Similarly, `<>(activity == "reject")` holds if the rejection activity was executed. Using `<->` and `!` we can formulate that exactly one of these two should hold. The formula `accept_or_reject_but_not_both` can be evaluated for each case in the log. If it holds for all cases, it holds for the entire log.

Lines 16-18 define the property that any decision (activity H in Figure 1) should be *directly* followed by a rejection (J), acceptance (I) or invitation (K). The following logical and temporal operators are used to achieve this: `[]` to denote the always operator ($\Box F$), `->` for implication, `_O` denote the nexttime operator ($\bigcirc F$), and `\/` for the logical or. The part `[]( (activity == "decide" ->` states that it should always be the case that if the current activity is decide, the following should hold. The second part starts with `_O` to indicate that immediately after the decision step the current activity should be of one of the three mentioned.

The formula specified in lines 23-27 uses the parameterized subformula defined in lines 20-21. The subformula states whether at some point in time activity `a` was executed by person `p`. Note that both parameters are typed through the declarations in the top part of Table 2. Formula `not_the_same_reviewer` calls the subformula six times to express the requirement that no reviewer should review the same paper twice. In terms of Figure 1: activities B, D, and F should be executed by different people. Note that universal quantification over the set people involved is used (cf. `forall[p:person | ...)` where `person` is renamed in Line 9 and declared to be a set type in Line 2.

The formula specified in lines 32-34 uses the parameterized subformula defined in lines 29-30. The subformula checks whether there is some event corresponding to activity `a` that has a data attribute `result` set to value accept, i.e., `ate.result == "accept"`. Note that `ate.result` was declared in Line 6. Formula `dont_reject_paper_unjustified` states that a paper with three positive reviews (three accepts) should be accepted for the journal.

The last formula in Table 2 (lines 36-37) shows that it is also possible to use timestamps. The formula has two parameters (start and end time) and it holds if each case was started before the given start time and ended after the given end time.

The formulas shown in Table 2 are specific for the running example introduced in Section 2. However, many generic properties can be defined, e.g., the *4-eyes principle*. Recall that this principle states that, although authorized to execute two activities, a person is not allowed to execute both activities for the same case. The following formula can be used to verify this:

```
formula four_eyes_principle(a1:activity,a2:activity) :=
forall[p:person |(!(execute(p,a1)) \/ !(execute(p,a2)))];
```

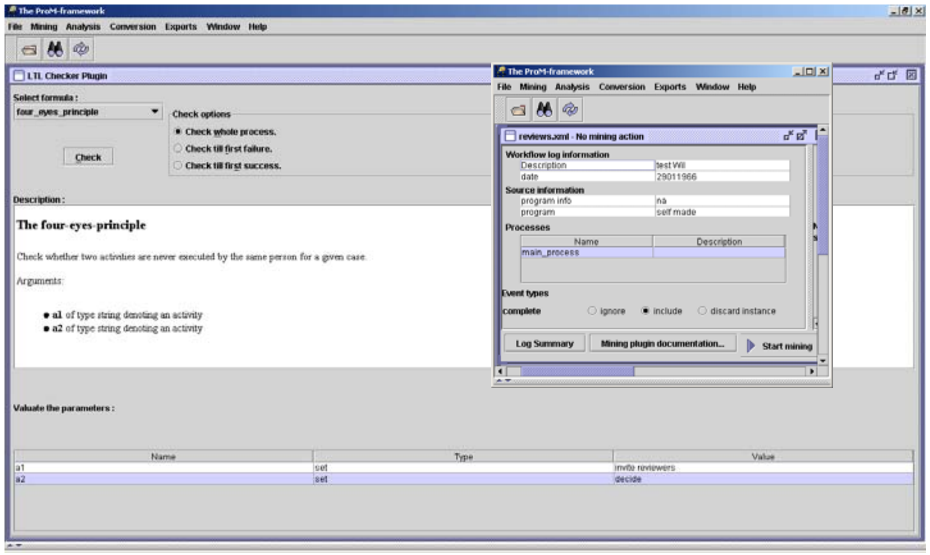
The property `four_eyes_principle("invite reviewers","decide")` checks whether activities A and H in Figure 1 are indeed executed by different people. This example and the formulas in Table 2 provide an impression of the LTL language we have developed. It can be seen as a temporal logic tailored towards events logs. For more details we refer to [10] and www.processmining.org. The next section elaborates on the tool support for this language.

5 Verifying Properties: The LTL Checker

In Section 3, the ProM framework has been introduced. To be able to verify properties using the language presented, three plug-ins are needed: (1) a mining plug-in to load and filter an MXML file, (2) an import plug-in to load LTL files like the one shown in Table 2, and (3) an analysis plug-in providing the graphical interface and doing the actual verification. For convenience a large number of generic properties have been specified (e.g., the 4-eyes principle). There are about 60 application-independent properties focusing on the `ate.WorkflowModelElement` (activity), `ate.Originator` (person), `ate.Timestamp`, and `ate.EventType` attributes. Only for specific questions (e.g., related to data) the user needs to specify new formulas. The 60 standard formulas are stored in a default file that can be applied directly without any knowledge of the LTL syntax described in the previous section. It is possible to link HTML markup to any formula. This markup is shown to the user when selecting a formula. This should support the selection and interpretation of the corresponding property. Note that formulas may be parameterized and users need to type a value for each parameter, e.g., the two activity names in `four_eyes_principle("invite reviewers","decide")`. The graphical user interface shows the HTML text and a form that need to be filled out, cf. Figure 5.

The implementation of the actual LTL Checker is rather complicated. However, the core structure of the checker is fairly straightforward as is sketched in the remainder of this section. Let L denote the event log and F a formula expressed in the language described in the previous section. (If F is parameterized, then it is evaluated for concrete parameter values.) $check_{log}(L,F) = \forall_{\pi \in L}(check(F,\pi,0))$ evaluates to true if F holds for the log L. $\pi \in L$ is a process instance (i.e., case) represented by a sequence of audit trail entries (i.e., events). $|\pi|$ is the length of π, i.e., the number of events, and π_i is the $(i-1)$-th entry, i.e., $\pi = \pi_0\pi_1 \ldots \pi_{|\pi|-1}$.
$check(F,\pi,0)$ checks whether the formula F holds for the first process-instance $\pi \in L$ (i.e., the π at position 0 in L). For temporal operators, the position in the sequence π is relevant as well. Therefore, let F denote a formula, π a case, and i a number $(0 \leq i < |\pi|)$.

$check(F,\pi,i) =$
if $F =$


The ProM-framework
File Mining Analysis Conversion Exports Window Help
LTL Checker Plugin
Check
The ProM-framework
File Mining Analysis Conversion Exports Window Help
reviews.xml - No mining action
Log Summary
Mining plugin documentation...
Start mining

Select formula : four_eyes_principle

Check options
[✓ Check whole process.]
Check till first failure.
Check till first success.

Description :

The four-eyes-principle

Check whether two activities are never executed by the same person for a given case.

Arguments:

- **a1** of type string denoting an activity
- **a2** of type string denoting an activity

Valuate the parameters :

Name	Type	Value
a1	set	invite reviewers
a2	set	decide

Workflow log information

Description	test Wil
date	29011966

Source information

program info	na
program	self made

Processes

Name	Description
main_process	

Event types

complete ignore [✓ include] discard instance

Fig. 5. The LTL Checker allows for the selection of formulas from a pull-down menu, offers help text, and supports the setting of parameters.

$expr(\pi_i) \Leftrightarrow$
  $true,$ expr is atomic and holds for i-th audit trail entry of π, i.e., π_i;
  $false,$ expr is atomic and does not hold for i-th audit trail entry of π, i.e., π_i;
$\neg\phi \Leftrightarrow \neg check(\phi, \pi, i);$
$\phi \wedge \psi \Leftrightarrow check(\phi, \pi, i) \wedge check(\psi, \pi, i);$
$\phi \vee \psi \Leftrightarrow check(\phi, \pi, i) \vee check(\psi, \pi, i);$
$\phi \rightarrow \psi \Leftrightarrow check(\phi, \pi, i) \rightarrow check(\psi, \pi, i);$
$\phi \leftrightarrow \psi \Leftrightarrow check(\phi, \pi, i) \leftrightarrow check(\psi, \pi, i);$
$\forall_{x \in X}(\phi_x) \Leftrightarrow \forall_{x \in X}(check(\phi_x, \pi, i));$
$\exists_{x \in X}(\phi_x) \Leftrightarrow \exists_{x \in X}(check(\phi_x, \pi, i));$
$\Box\phi \Leftrightarrow$
  $check(\phi, \pi, i) \wedge check(F, \pi, i+1), 0 \leq i < (|\pi| - 1);$
  $check(\phi, \pi, i), i = (|\pi| - 1);$
$\Diamond\phi \Leftrightarrow$
  $check(\phi, \pi, i) \vee check(F, \pi, i+1), 0 \leq i < (|\pi| - 1);$
  $check(\phi, \pi, i), i = (|\pi| - 1);$
$\bigcirc\phi \Leftrightarrow$
  $check(\phi, \pi, i+1), 0 \leq i < (|\pi| - 1);$
  $false, i = (|\pi| - 1);$
$\phi \mathbf{U} \psi \Leftrightarrow$
  $check(\psi, \pi, i) \vee (check(\phi, \pi, i) \wedge check(F, \pi, i+1)), 0 \leq i < (|\pi| - 1);$
  $check(\psi, \pi, i) \vee check(\phi, \pi, i), i = (|\pi| - 1);$

The *expr* function is a function which computes atomic Boolean expressions that may involve all kinds of attributes (e.g., timestamps etc. but also data values or case attributes). Given the fact that there are many comparison operators, typing issues and advanced features such as pattern matching, the coding of the LTL Checker is more complex than the sketch just given suggests.

The unfolding of the quantifications may be an expensive operation. However, no quantification is bigger than the number of events within a single case. Moreover, each case (process instance) can be checked in isolation thus making the algorithms tractable. Note that logs may contain thousands or even millions of cases. However, the number of events per case is typically less than 100. Therefore, from a practical point of view, the core algorithm is linear in the size of the log.

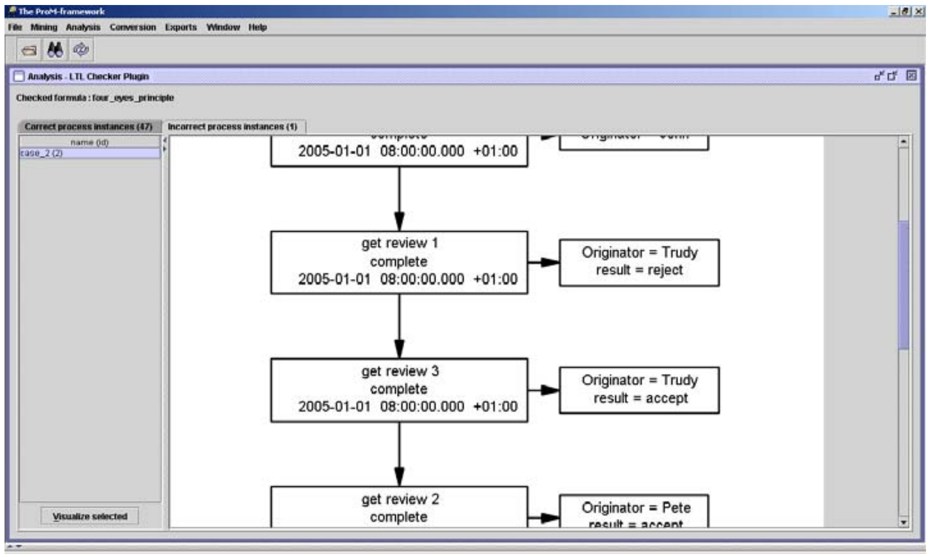


Fig. 6. The LTL Checker detected a problem when checking the 4-eyes principle.

To conclude, we show a screenshot of the result, cf. Figure 6. It shows the result of `four_eyes_principle("get review 1","get review 3")` applied to the log with 48 cases. 47 of these cases satisfy the property. Only for one case (`case_2`), the property is not satisfied as shown in Figure 6. Indeed the paper is reviewed by Trudy twice. In one review, she rejects the paper while in another one she accepts it.

For every property, the LTL Checker partitions the set of cases into two sets: L^{OK} (the cases that satisfy the property) and L^{NOK} (the ones that do not). If $L^{NOK} = \emptyset$, the property holds. Otherwise, L^{NOK} provides counterexamples. Both sets can be saved and analyzed further. For example, it is possible to construct a process model or social network for L^{OK} or L^{NOK}. This may be helpful when analyzing (root) causes for violations of a desirable property.

6 Related Work

The work reported in this paper is closely related to earlier work on process mining, i.e., discovering a process model based on some event log. The idea of applying process mining in the context of workflow management was first introduced in [9]. Cook and Wolf have investigated similar issues in the context of software engineering processes using different approaches [12]. Herbst and Karagiannis also address the issue of process mining in the context of workflow management using an inductive approach [21]. They use stochastic task graphs as an intermediate representation and generate a workflow model described in the ADONIS modeling language. Then there are several variants of the α algorithm [8, 37]. In [8] it is shown that this algorithm can be proven to be correct for a large class of processes. In [37] a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs". This is used as input for the α algorithm. As a result it is possible to tackle the problem of noise. For more information on process mining we refer to a special issue of Computers in Industry on process mining [7] and a survey paper [6]. Given the scope of this paper, we are unable to provide a complete listing of the many papers on process mining published in recent years.

Conformance testing, i.e., checking whether a given model and a given event log fit together, can be a seen a very specific form of log-based verification. Instead of some logical formula, a process model (e.g., Petri net) is used to verify whether the log satisfies some behavioral properties. Therefore, the work of Cook et al. [13, 11] is closely related to this paper. In [13] the concept of process validation is introduced. It assumes an event stream coming from the model and an event stream coming from real-life observations. Both streams are compared. Only in the last part of the paper an attempt is made to selectively try and match possible event streams of the model to a given event stream. As a result only fitness is considered and the time-complexity is problematic as the state-space of the model needs to be explored. In [11] the results are extended to include time aspects. The notion of conformance has also been discussed in the context of security [4], business alignment [2], and genetic mining [26].

Monitoring events with the goal to verify certain properties has been investigated in several domains, e.g., in the context of requirements engineering [15, 31, 32] and program monitoring [16, 19, 20]. It is also interesting to note the possible applications of such techniques in the context of monitoring web services. In such a distributed environment with multiple actors, it is highly relevant to be able to monitor the behavior of each actor.

The work Robinson [31, 32] on requirements engineering is highly related. He suggests the use of LTL for the verification of properties. Important differences between this approach and ours are the focus on real-time monitoring (with model-checking capabilities to warn for future problems) and the coding required to check the desired properties. The following quote taken from [31] illustrates the focus of this work: "Execution monitoring of requirements is a technique that tracks the run-time behavior of a system and notes when it deviates from its design-time specification. Requirements monitoring is useful when it is too difficult (e.g., intractable) to prove system properties. To aid analysis, assumptions are made as part of the requirements definition activity. The requirements and assumptions are monitored at run-time. Should any such conditions fail, a procedure can be invoked (e.g., notification to the designer)." In a technical sense, the work of Havelund et al. [19, 20] is highly related. Havelund et al. propose three ways to evaluate LTL formulas: (1) automata-based, (2) using rewriting (based on Maude), (3) and using dynamic programming. We use the latter approach (dynamic programming).

Process mining, conformance testing, and log-based verification can be seen in the broader context of Business (Process) Intelligence (BPI) and Business Activity Monitoring (BAM). In [17, 18, 35] a BPI toolset on top of HP's Process Manager is described. The BPI tools set includes a so-called "BPI Process Mining Engine". In [27] Zur Muehlen describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [23]. The latter tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [36] which is tailored towards mining Staffware logs. Note that none of the latter tools is supporting conformance testing or the checking of (temporal) properties. Instead, the focus of these tools is often on performance measurements rather than monitoring (un)desirable behavior.

For a more elaborate description of the LTL language and checker we refer to [10]. The idea to apply LTL to workflow management originates from the first author and was implemented by the second author under the supervision of the first and third author. In parallel to this implementation, a simplified prototype was developed [33].

7 Conclusion

This paper presents both a language and a tool to enable the verification of properties based on event logs. The language is based on LTL and is tailored towards events logs stored in the MXML format. The MXML format is a tool-independent format to log events and can be generated from audit trails, transaction logs and other data sets describing business events. Current software allows for the easy collection of such data, cf. BPM, WFM, CRM, BAM systems. Moreover, the need for both flexibility [1, 14, 29, 30] and auditing capabilities (cf. the Sarbanes-Oxley Act [34]) underpins the relevance of the results presented in this paper.

We have predefined 60 typical properties one may want to verify (e.g., the 4-eyes principle). These can be used without any knowledge of the LTL language. In addition the user can define new sets of properties. These properties may be application specific and may refer to data. Each property is specified in terms of a so-called formula. Formulas may be parameterized, are reusable, and carry explanations in HTML format. This way both experts and novices may use the

LTL Checker. The LTL Checker has been implemented in the ProM framework and the results can be further analyzed using a variety of process mining tools.

We do not propose to solely use LTL for the type of analysis discussed in this paper. In addition to the LTL Checker and the process mining tools, conventional tools such as SQL and XPath can also be used to query and filter event logs. For example, in the context of a case study we loaded MXML files into an Oracle database to query them using SQL.

References

1. W.M.P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, 3(3):297–317, 2001.
2. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the 5th Workshop on Business Process Modeling, Development and Support (BPMDS'04)*, volume 2 of *Caise'04 Workshops*, pages 138–145. Riga Technical University, Latvia, 2004.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst and A.K.A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *Second International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004)*, pages 69–84. STAR, Servizio Tipografico Area della Ricerca, CNR Pisa, Italy, 2004.
5. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.
6. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
7. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
8. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
9. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
10. H. de Beer. *The LTL Checker Plugins: A Reference Manual.* Eindhoven University of Technology, Eindhoven, 2004.
11. J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Proceedings of the 2001 International Conference on Software Mainenance*, pages 332–341, 2001.

12. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
13. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
14. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
15. S. Fickas, T. Beauchamp, and N.A.R. Mamy. Monitoring Requirements: A Case Study. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02)*, page 299. IEEE Computer Society, 2002.
16. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 412–416. IEEE Computer Society Press, Providence, 2001.
17. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
18. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
19. K. Havelund and G. Rosu. Monitoring Programs Using Rewriting. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01)*, pages 135–143. IEEE Computer Society Press, Providence, 2001.
20. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, Berlin, 2002.
21. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
22. T. Hoffman. Sarbanes-Oxley Sparks Forensics Apps Interest: Vendors Offer Monitoring Tools to Help Identify Incidents of Financial Fraud. *ComputerWorld*, 38:14–14, 2004.
23. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com, 2002.
24. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
25. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
26. A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results. BETA Working Paper Series, WP 124, Eindhoven University of Technology, Eindhoven, 2004.
27. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings*

of the 33rd Hawaii International Conference on System Science (HICSS-33), pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.

28. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Providence, 1977.
29. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
30. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
31. W.N. Robinson. Monitoring Software Requirements using Instrumented Code. In *Proceedings of the 35th Annual Hawaii IEEE International Conference on Systems Sciences*, pages 276–276. IEEE Computer Society , 2002.
32. W.N. Robinson. Monitoring Web Service Requirements. In *Proceedings of 11th IEEE International Conference on Requirements Engineering (RE 2003)*, pages 56–74. IEEE Computer Society , 2003.
33. E. Roubtsova. A Property Specification Language for Workflow Diagnostics. Internal note, Eindhoven University of Technology, 2005.
34. P. Sarbanes, G. Oxley, and et al. Sarbanes-Oxley Act of 2002, 2002.
35. M. Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
36. Staffware. Staffware Process Monitor (SPM). http://www.staffware.com, 2002.
37. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.