Моделирование и анализ информационных систем. Т. 24, № 2 (2017), с. 125–140
Modeling and Analysis of Information Systems. Vol. 24, No 2 (2017), pp. 125–140

DOI: 10.18255/1818-1015-2017-2-125-140

UDC 517.9

Discovering High-Level Process Models from Event Logs

Begicheva A. K., Lomazova I. A.

Received January 11, 2017

Abstract. Process mining is a relatively new field of computer science, which deals with process discovery and analysis based on event logs. In this paper we consider the problem of discovering a high-level business process model from a low-level event log, i.e. automatic synthesis of process models based on the information stored in event logs of information systems. Events in a high-level model are abstract events, which can be refined to low-level subprocesses, whose behavior is recorded in event logs. Models synthesis is intensively studied in the frame of process mining research, but only models and event logs of the same granularity are mainly considered in the literature. Here we present an algorithm for discovering high-level acyclic process models from event logs and some specified partition of low-level events into subsets associated with abstract events in a high-level model.

Keywords: Petri nets, high-level process models, event logs, process mining, process discovery

For citation: Begicheva A. K., Lomazova I. A., "Discovering High-Level Process Models from Event Logs", *Modeling and Analysis of Information Systems*, **24**:2 (2017), 125–140.

About the authors:
Antonina A. Begicheva, research assistant
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems
20 Myasnitskaya str., Moscow 101000, Russia, e-mail: akbegicheva@edu.hse.ru

Irina A. Lomazova, Doctor of science, professor
National Research University Higher School of Economics
20 Myasnitskaya str., Moscow 101000, Russia, e-mail: ilomazova@hse.ru

Acknowledgments:
This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No.16-01-00546.

1. Introduction

Process mining is a technology that provides a variety of methods to discover, monitor and improve real processes by extracting knowledge from event logs [1]. Process discovery and conformance checking are the two most prominent process mining tasks. Process discovery is needed to construct a process model, based on an event log, without any additional information. Conformance checking helps us in diagnosing and quantifying discrepancies between observed and modeled behavior. Process discovery uses only an event log for recovery view of a system as a model by behavior which is seen in the log. The general goal is to find out main events of the system and relations between them.

After this we can use other techniques of process mining for further work with the model of our system.

There are many software products which allow us to use these two techniques of Process Mining. ProM [2] is an open-source tool supporting many techniques of Process Mining, which are represented as plug-ins. Due to the flexibility of this environment, it can be used both for research and applications.

When working with business processes we typically use detailed logs, which present the full report about sequences of executed activities. Since logs are generated automatically in most information systems, keeping detailed records is not a problem. However, large and detailed models are not good to deal with. Such models are not clear and readable for experts. Experts prefer working with more abstract (high-level) models. More abstract models are easier to construct, understand and analyze. Process models developed by people are, as a rule, not very large and abstract from technical details. Although the field of process discovery is well developed, process mining projects still face the problem of different levels of abstraction when comparing events with modeled business activities. Current approaches for event log abstraction most often try to abstract from the events in an automated way which does not capture the required domain knowledge to fit business process.

In this work we consider process models represented by workflow nets – a special class of Petri nets [3] for workflow modeling. We assume that a model contains no cycles for correct handling with concurrency. In an abstract model each separate activity represents a subprocess built from a set of more refined activities. For the presentation of intermediate results we use transition systems. A history of a detailed process behavior is recorded in low-level logs. We present an algorithm for discovering process as an abstract model from a low-level event log. The algorithm will be tested on groups of input data with different characteristics.

The work is organized as follows. Section 2 introduces the situation with existing researches. In Section 3 we give a motivating example of handling a request for compensation in a particular airline company, in terms of Petri nets. Section 4 contains some basic definitions and notions, including Petri nets, event log, transition system, and theory of regions. In Section 5 we present a method for discovering an abstract process model from a low-level event log, and Section 6 gives a precise description and validation of this method. Section 7 concludes the paper.

2. Literature review

Research topics related to this article can be divided into several categories: event log and model abstraction, discovering algorithms and existing methods for abstract models synthesis.

There are many ways of abstracting models by reducing their size in order to make them more convenient to work with. Each method may be useful depending on a group of interrelated factors: the abstraction purposes, the presence of certain patterns of routine constructions, the specifics of modeling notation. Reducing the size of the model by abstraction can be done as “convolution” of some groups of elements, or implemented by losing some parts (which are non-significant in a particular case). The importance of event

log abstraction is emphasized among others in [4]. The paper contains a more detailed overview of models and abstraction techniques and formal definitions of this process in general. In particular, the paper provides definitions for two prominent abstraction operations: *elimination* and *aggregation*. A model which is generated by an elimination operation contains no information about omitted insignificant objects. In contrast the aggregation generates an abstract model, where relatively insignificant objects of abstraction combine together with several other abstraction objects into groups, each of which is significant. According to this article, business process model abstraction is an operation over the model, which transforms it into abstract model by the application of function composition, where each component of the composition is some basic abstract operation.

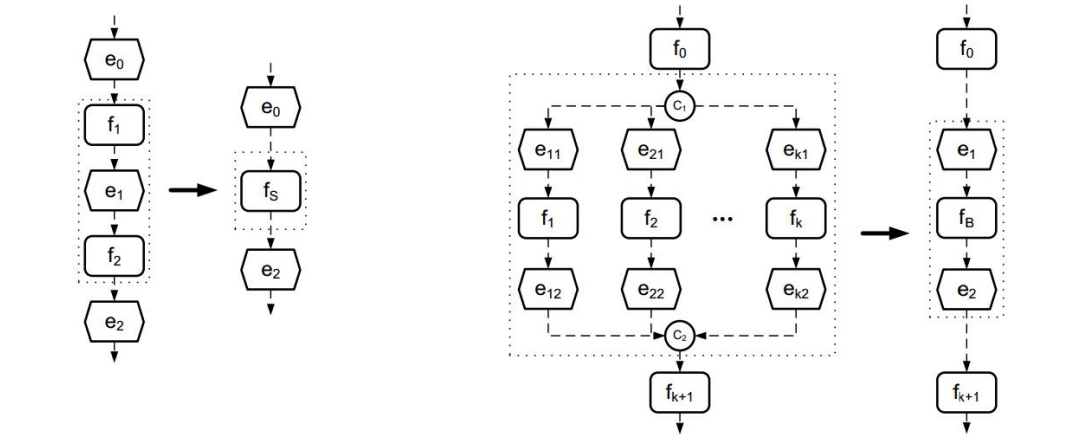
The aggregation is divided into four types: sequential abstraction (Fig. 1 (a)), block abstraction (Fig. 1 (b)), loop abstraction (Fig. 1. (c)), dead end abstraction (Fig. 1. (d)). Based on these four aggregation rules and model transformation techniques a slider approach is proposed in [5]. It provides a user control over the level of model abstraction by different criteria, where the abstraction criterion is a pair: type of criteria and relation between element property and its significance. A similar ability is used in many methods of working with abstract models due to its convenience and an analogy with scalable terrain maps.

Petri nets can be extended by adding a hierarchy e.g. in Colored Petri nets (CPN) [6]. The idea of high-level Petri nets was first described in [7] (1981), but only in 1987 this idea extended to colored Petri nets. Hierarchy also allows construction of more compact, readable and understandable models. Hierarchy can be applied as an abstraction, in the case of two-level hierarchy there are two models of one process: a high-level *abstract* model and a low-level *refined* model. In our paper the high-level model is a model with abstract transitions. An abstract transition refers to a Petri net subprocess, which refines the activity represented by this transition. The low-level model can be obtained from an abstract model by substituting subprocess models for abstract transitions.

A "flat" synthesis (when model and log are at the same-level) is a popular process mining task, and has been extensively studied in the literature. Each method [1, 8, 9] has some advantages regarding a particular kind of data. For example the α-algorithm [9] takes concurrency as a base, but this algorithm has some problems dealing with complicated routing constructs and noise. The flat model, presented as a graph, becomes more incomprehensible with each node. Hierarchical models are more suitable for humans to work with because they are more structural.

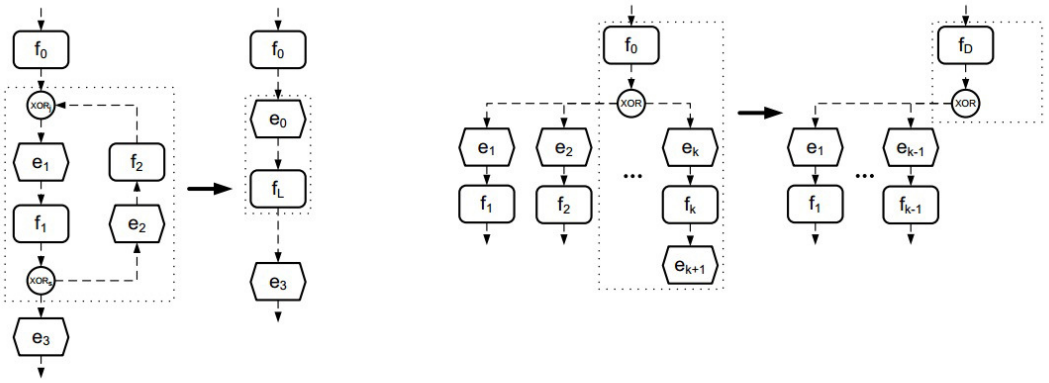
Van Dongen et al in [10] described a Petri net discovery approach using transition system. Firstly a transition system is constructed by deriving from an event log. It can be modified to avoid over-fitting. Secondly, using the theory of regions [11], the Petri net is synthesized. We note that currently only the theory of regions solves the problem of Petri net synthesis from transition systems. The proposed in [10] method can deal with complex control-flow constructs. It allows for duplicates, but does not allow for much more behavior than is actually recorded in the log and produces models, which satisfy some soundness requirements. The proposed algorithm also has some disadvantages, for example it can't deal with noise in an event log, unlike genetic [12] or heuristics [13] miners.

There are some methods for discovering abstract models but they are based on finding behavior patterns in event logs. For example in [14] the authors use recognition of



(a) Sequential abstraction

(b) Block abstraction



(c) Loop abstraction

(d) Dead end abstraction

Fig 1. Aggregations for model abstraction

behavior patterns of the process by a structural clustering algorithm and then define a specific workflow schema for each pattern. This helps to divide one complex model into several more understandable, smaller pieces, convenient to analyze and discover. Clustering of activities by their relation and role in the process was also used in [15]. The terrain maps analogy is also used here: activities in a process correspond to a certain location on a map and the relations between activities are like the roads.

In [16] a supervised event abstraction method was presented. This method takes an event log at a lower level of abstraction and transforms it to an event log at the desired level of abstraction, using behavioral activity patterns: sequence, choice, parallel, interleaving and repetition. This technique allows us to obtain a reliable abstraction mapping from low-level events to activity patterns automatically and construct a high-level event log using them. Another supervised event abstraction method is described in [17]. The essence of the proposed method is as follows: we annotated each low-level event with the correct high-level event using domain knowledge from the actual process model by special type of attribute in XES log file. Also in this paper authors make the working assumption that multiple high-level events are executed in parallel. This enables us to interpret a sequence of identical label attribute values as a single instance of a high-level event.

In [18] a two-phase approach to mining process is presented. Process models here are considered as interactive and context-dependent maps based on common execution patterns. On the first phase the event log is transformed to the desired level of detail by selecting patterns. An example of such pattern is the maximal repeat, that captures typical sequences of activities the log. Every pattern is estimated by frequency, or significance, or some other metric needed for accurate abstraction. At the second phase the fuzzy miner algorithm adapted to process maps discovery is applied to the transformed log. This two-phase approach has been implemented as a set of interrelated plug-ins in the ProM framework, the application order of which is described in [19]. In our paper we use Petrify plug-in from this set to synthesize Petri net [20].

All these papers provide no or only limited support for correct refining these mappings based on domain knowledge. They do not allow to detect subprocesses in the synthesized model. Some approaches based on subprocess detection for mining a process model with a better structure were presented in [21, 22], but these papers do not consider mining high-level models.

Also there are different approaches that apply behavior abstraction in process discovery and trace alignment [5, 23]. Besides there is an interesting method for discovering that is discussed in [24]. The contribution of this research is a mapping approach which suggests relations between events and activities in an automated manner using existing process documentation as e.g. work instructions.

Thus discovering an abstract model from a low-level event log generated by an information system is an important and challenging problem. In different applications different views on the abstraction mechanism and different levels of abstraction are needed. Here we consider one of many possible views on the problem.

$$
\begin{aligned}
L = \{ &< t0, t1, t3, t4, t5, t6, t13, t14, t7, t8, t15, t9, t16, t17, t18, t25, t26, t27 >, \\
&< t0, t1, t3, t4, t13, t14, t5, t7, t15, t6, t8, t9, t16, t17, t18, t24, t23, t22 >, \\
&< t0, t1, t3, t13, t10, t11, t14, t15, t12, t16, t17, t18, t24, t23, t22 >, \\
&< t0, t2, t3, t13, t4, t14, t15, t5, t6, t7, t8, t9, t16, t17, t18, t25, t26, t27 >, \\
&< t0, t2, t3, t4, t13, t14, t15, t5, t7, t6, t8, t9, t16, t17, t18, t24, t23, t22 >, \\
&< t0, t1, t3, t10, t11, t13, t12, t14, t15, t16, t17, t18, t19, t20, t21, t10, t11, \\
&t13, t14, t15, t12, t16, t17, t18, t25, t26, t27 >, \\
&< t0, t2, t3, t13, t10, t14, t15, t11, t12, t16, t17, t18, t24, t23, t22 >, \\
&< t0, t2, t3, t13, t10, t11, t12, t14, t15, t16, t17, t18, t19, t20, t21, t10, t13, \\
&t14, t11, t15, t12, t16, t17, t18, t24, t23, t22 >, \\
&< t0, t2, t3, t13, t14, t10, t11, t15, t12, t16, t17, t18, t25, t26, t27 >\}.
\end{aligned}
$$

Fig 2. An original event log $\mathcal{L}_1$

3. Motivation example

Let us consider some log, which describes in detail a half-hour period of handling a request for a compensation from the airline service. The log is generated by a system, a part of this los is shown in Fig. 2. Names of events are simplified to t with some index, because with real names the log was too bulky. Even if names of events have not been replaced, it is not easy to understand from the log how customers requests are handled. But there are algorithms allowing to discover (synthesize) a process model from this log.

After applying one of the known discovery algorithms we get the relatively large model $\mathcal{N}_1$ presented in Fig. 3. This model is inconvenient to work with. Experts would prefer to work with more abstract model of the same process, like the model $\mathcal{N}_2$ shown in Fig. 4. Each transition in $\mathcal{N}_2$ corresponds to a subprocess, which includes transitions from $\mathcal{N}_1$. For example, the abstract action 'register request' indicates reading a request (transition $t0$) and then recording it in one of two possible ways (transition $t1$, or $t2$). Note that sets of low-level transitons corresponding to different abstract events do not intersect. Low-level transitions in Fig. 3 are grouped into blocks (subprocesses) corresponding to

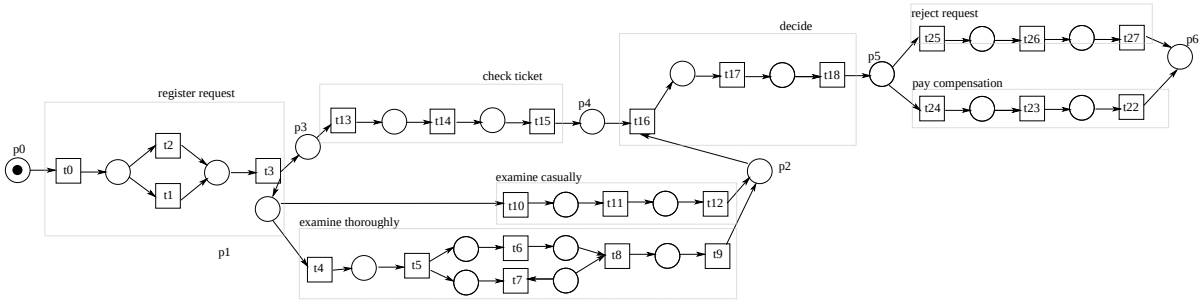


Fig 3. A low-level model $\mathcal{N}_1$ synthesized from the log $\mathcal{L}_1$ in Fig. 2

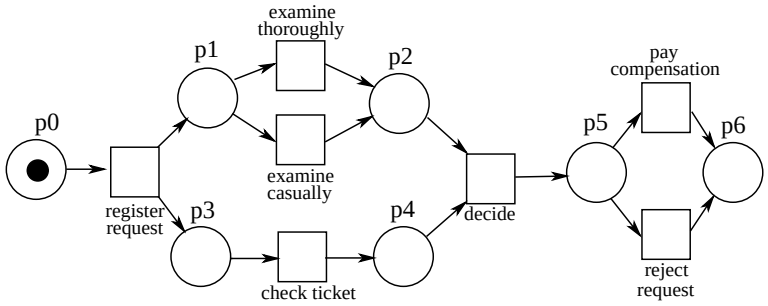


Fig 4. An abstract model $\mathcal{N}_2$ for handling compensation requests

high-level activities of the process.

We study the problem when given a low-level event log generated by a process and a partition of the set of low-level events into subsets corresponding to high-level events , we would like to construct an abstract model of the process. We suppose that the low-level log is generated by an information system, and the event partition is defined by experts or software developers

An abstract model cannot be directly obtained from low-level log since the log consists of low-level events and the model should contain transitions labeled by high-level events. So, a one-to-many correspondence between high and low-level events is needed for the model synthesis. In our example the abstract event $e0$ corresponds to the set $\{t0, t1, t2\}$ of low-level events, $e1$ — to the set $\{t13, t14, t15\}$, etc. The full correspondence is shown in Table 1. This mapping will be used in the algorithm for transformation of the original log into its high-level representation, which we use as an input data for one of the known discovery algorithm.

High-Level Activity	e0	e1	e2	e3	e4	e5	e6	e7
Low-Level Activities	t0	t4	t10	t13	t16	t19	t22	t25
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	t3	t9	t12	t15	t18	t21	t24	t27

Table 1. The correspondence between low-level and high-level activities in $\mathcal{N}_1$ and $\mathcal{N}_2$

4. Preliminaries

In this section we give some basic notions and definitions used in the paper.

Let S be a set. By S^* we denote the set of all finite sequence (words) over S.

$S = S_1 \cup S_2 \cup \cdots \cup S_n$ is a partition of S iff $\forall i, j \in [1, n] : S_i \subseteq S$ and $S_i \cap S_j = \emptyset$.

A multiset m over a set S is a mapping: $m : S \to Nat$, where Nat – is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

Definition 1 (Petri net)**.** *Let P and T be disjoint finite sets of places and transitions and $F : (P \times T) \cup (T \times P) \to Nat$. Then $N = (P, T, F)$ is a* Petri net. *Let A be a finite set*

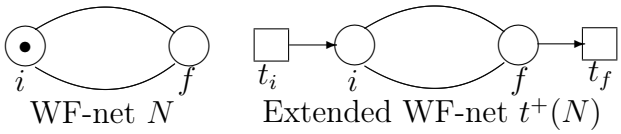


Fig 5. Extending WF-net with initial and final transitions

of activities. A labeled *Petri net is a Petri net with a labeling function* $\lambda : T \to A \cup \{\epsilon\}$, *which maps every transition to an activity (a transition label) from* A, *or a special label* ϵ *indicating an invisible action.*

A marking in a Petri net is a function $m : P \to Nat$ mapping each place to some natural number (possibly zero).

For a transition $t \in T$ a *preset* ${}^\bullet t$ and a *postset* $t^\bullet$ are defined as the multisets over P such that ${}^\bullet t = \{p | F(p,t) \neq 0\}$ and $t^\bullet = \{p | F(t,p) \neq 0\}$.

A transition $t \in T$ is *enabled* in a marking m iff $\forall p \in P\ m(p) \geq F(p,t)$. An enabled transition t may *fire* yielding a new marking m', such that $m'(p) = m(p) - F(p,t) + F(t,p)$ for each $p \in P$ (denoted $m \xrightarrow{t} m'$).

A *Workflow net* is a (labeled) Petri net with two special places: i and f. These places are used to mark the beginning and the ending of a workflow process.

Definition 2 (Workflow net)**.** *A (labeled) Petri net* $\mathrm{N} = (\mathrm{P}, \mathrm{T}, \mathrm{F}, \lambda)$ *is called a* workflow net *(*WF-net*) iff:*

1. *There is one source place* $i \in P$ *and one sink place* $f \in P$, *s. t.* ${}^\bullet i = f^\bullet = \emptyset$.
2. *Every node from* $P \cup T$ *is on a path from* i *to* f.
3. *The initial marking in* N *contains the only token in its source place.*

By abuse of notation we denote by i both the source place and the initial marking in a WF-net. Similarly, we use f to denote the final marking in a WF-net N, defined as a marking containing the only token in the sink place f.

Let $N = (P, T, F, \lambda)$ be a WF-net. Then we define the extended WF-net (EWF-net) $N' = (P', T', F', \lambda')$ as follows: $P' = P, T' = T \cup \{t_i, t_f\}$ and $F' = F \cup \{\langle t_i, i\rangle, \langle f, t_f\rangle\}$, where t_i, t_f are new (not occurring in P, T) nodes. The new transitions t_i, t_f are labeled with invisible activity ϵ, all other transitions in N' have the same labels as in N. The initial marking in an extended WF-net contains no tokens. Thus an extended WF-net may start a new case at any moment (cf. Fig. 5).

The behavior of WF-nets can be represented with a state-based models, called Transition System (TS), as they reflect the states of a process and transitions between them.

Definition 3 (Transition system)**.** *A (labeled) transition system is a triple* $(S, \Lambda, \to)$, *where* S *is a set of states,* Λ *is a set of labels, and* $\to \subseteq S \times \Lambda \times S$ *is a transition relation. If* $p, q \in S$ *and* $\alpha \in \Lambda$, $(p, \alpha, q) \in \to$ *is usually written as* $p \xrightarrow{\alpha} q$ *meaning that a transition labeled by* α*moves the system from state* p *to state* q. *Furthermore, in this paper we assume that a transition system is connected.*

Information systems can record all kinds of events with a wide range of properties, but it entirely depends on the configuration. Moreover, systems can use a specific format. For our study we abstract from additional information presented in event logs and define a *process log* as a multiset of *traces*, where a *trace* is a sequence of *events* (only their names).

In this paper a *path* is a sequence of events apart from the log, in the context of a one of possible runs from initial state of a model to final state.

Definition 4 (Event log)**.** *Let A be a finite set of activities. A trace σ is a finite sequence of activities from A, i.e. $\sigma \in A^*$. An event log L is a finite multi-set of traces, i.e. $L \in \mathcal{M}(A^*)$.*

A model represented as transition system TS corresponding to a model represented as Petri net PN iff TS can be successfully run with all possible paths of TS and vice versa.

To apply the Petri net synthesis method, we need to transform event logs into transition systems [10]. Using the translation from a single trace to a transition system, we can translate an entire log to a transition system.

Definition 5 (Event log to transition system)**.** *Let A be a set of log events and let W be an event log over A, i.e., $W \in \mathcal{M}(A^*)$. We define $TS(W) = (S; \Lambda; \rightarrow)$ to be a transition system, such that:*

- $S = \bigcup_{\sigma \in W} S_\sigma$*, i.e. the union over the states of the transition system translations of each individual trace,*
- $\Lambda = A$*, i.e. the set of labels is the set of activities,*
- $\rightarrow = \bigcup_{\sigma \in W} \rightarrow_\sigma$*, i.e. the trace is represented as a sequence of state transitions, starting from the common initial state. The transitions between each two states is made by activity at the given position in the trace.*

The algorithm [8] for constructing a transition system is straightforward: for every trace σ, iterating over $k (0 \leq k \leq |\sigma|)$, we create a new state $state(\sigma, k)$ if it does not exist yet. Then the traces are scanned for transitions $state(\sigma, k-1) \xrightarrow{\sigma(k)} state(\sigma, k)$ and these are added if it does not exist yet. If we use the complete prefix sequence representation of a state, i.e. $state(\sigma, k) = hd(\sigma, k)$, where $hd(\sigma, k)$ is a *prefix* of trace σ after executing k steps.

Once a process log is converted into transition system, we can use the some synthesis method to generate a Petri net from it, at the time of writing this article for these purposes, you can apply only Theory of Regions.

For synthesis we use Theory of Regions, and to able using it we need to make an assumption about completeness of the log. We assume also that the log shows all possible behavior, since the resulting Petri net have to be exactly mimic the behavior shown in the log. Since we work only with workflow process, we do not need to make any assumptions about the uniqueness of initial state for each trace. More details about the application of the theory of regions can be found in [10, 25].

The advantages of using theory of regions are the possibility to handle complex models with routing constructions (such as concurrency), the possibility of deduplication in the

construction of the resulting net, the flexibility, due to which the model does not allow too general behavior and are not too "tight" for some concrete routing. The disadvantage of theory of regions is high computational complexity, which makes it difficult to use for synthesis models from large size logs. The model abstraction allow us to use all of advantages of theory of regions faced with no disadvantages.

Second of the most prominent process mining tasks is conformance checking [1, 26, 27]. Given a model and an event log we would like to compare the process model behavior and the behavior recorded in the event log. Several metrics for conformance checking were defined in the literature [1]. Among the most important metrics is *fitness*. Informally speaking, fitness measures the proportion of behavior in the event log possible according to the model.

Definition 6 (Perfect fit)**.** *Let N be a WF-net with transition labels from A, an initial marking i, and a final marking f. Let σ be a trace over A. We say that a trace $\sigma = a_1, \ldots, a_k$* perfectly fits *$N$ iff there exists a sequence of firings $i = m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m_{k+1} = f$ in N, s.t. the sequence of activities $\lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k)$ after deleting all invisible activities ϵ coincides with σ. A log L* perfectly fits *N iff every trace from L perfectly fits N.*

Workflows can be modeled with varying degrees of detailing. This idea is realized in models of different abstraction levels. The correspondence between models of different level, which describe the same process, is natural to match using refinements. When a transition from the model is a reference to the sub process, this transition is an abstract (high-level) one. In this case, we can obtain detailed (low-level) model by substituting of corresponding sub-process model instead of high-level transition. This hierarchy principle is used, for example, in colored Petri nets (CPN) [6]. Refinements allow us to develop a more compact model with the composite structure of the network. Here we give precise definitions introduced in [27].

Definition 7 (Substitution)**.** *Let $N_1 = (P_1, T_1, F_1, \lambda_1)$ be a WF-net, $t \in T$ be a transition in N_1. Let also $N_2 = (P_2, T_2, F_2, \lambda_2)$ be an EWF-net with the initial and final transitions t_i, t_f correspondingly. We say that a WF-net $N_3 = (P_3, T_3, F_3, \lambda)$ is obtained by a* substitution *$[t \to N_2]$ of N_2 for t in N_1 iff $P_3 = P_1 \cup P_2$, $T_3 = T_1 \cup T_2 \setminus \{t\}$, $F_3 = F_1 \cup F_2 \setminus \{(p,t) \mid p \in {}^\bullet t\} \setminus \{(t,p) \mid p \in t^\bullet\} \cup \{(p, t_i) \mid p \in {}^\bullet t\} \cup \{(t_f, p) \mid p \in t^\bullet\}$,*

Definition 8 (Refinement)**.** *Let N, N_r be two WF-nets with sets of activities A, A_r correspondingly. Let $A = a_1, a_2, \ldots, a_n$, and $A_r = A_r^1 \cup A_r^2 \cup \ldots \cup A_r^n$ be a partition of A_r into n subsets, and $N^1, N^2, \ldots N^n$ be EWF-nets with sets of activities $A_r^1, \ldots, A_r^n$ correspondingly. We say that N_r is a refinement of N via substitutions $[a_1 \to N_r^1, a_2 \to N_r^2, \ldots a_n \to N_r^n]$ iff N_r can be obtained from N by simultaneous substitutions of N_r^i for all t s.t. $\lambda(t) = a_i$.*

5. Synthesis of Abstract Process Model from a Low-Level Event Log

In this section we describe, how to obtain an abstract model from a given low-level log L. We suppose that the set A_r of low-level events in the log L is partitioned into subsets

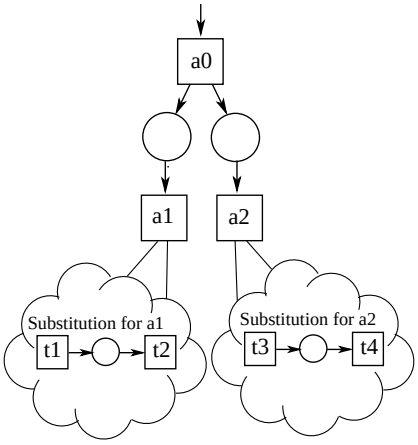


Fig 6. Concurrent execution of two subprocesses (abstract transitions in a model)

$A_r = A_1 \cup \cdots \cup A_n$, and each subset A_i is assigned an abstract event name a_i from a given set A of abstract names.

We would like to construct a model with transitions labeled by names from A, for which there exists a refinement via some substitution $[a_1 \to N_r^1, a_2 \to N_r^2, \ldots a_n \to N_r^n]$ respecting subset abstract names (i.e. if a transition t labeled by a_i is substituted by a net N_r^i, then transitions in N_r^i are labeled by names from A_i), such that the given log L conforms the refined model.

We suppose also, that the model we discover is acyclic. This assumption is due to the fact that several occurrences of low-level events belonging to the same subset A_i may be caused both by cyclic repetition of the abstract event a_i, and by executing a_i concurrently with some other abstract event via interleaving (cf. Fig. 6). So, in this paper we consider only acyclic case.

Discovery of a high-level model from a given event log will be done in two stages. First we construct a high-level transition system corresponding to the given low-level log, and then we use the well-known method based on theory of regions to discover a WF-net model from the transition system.

Now we explain our approach, the detailed algorithm is presented in the next section.

To construct a high-level transition system we first replace each activity in the given low-level log by the corresponding abstract activity. After this replacement we obtain the log, containing only high-level activities. However obtained traces may contain "stuttering" subsequences, when the same action occurs several times sequentially. This happens when several low-level activities corresponding to the same abstract activity go one by one in a trace. Such a "stuttering" subsequence should by replaced by one abstract activity.

However, removing stuttering is not enough to obtain a correct high-level log, where each abstract activity occurrence in a trace corresponds to one abstract event firing. If there are concurrent abstract actions in a system, representing subprocesses run in parallel, then low-level activities of these subprocesses may be interleaved in a trace. Then after replacing low-level activities by abstract ones and after removing stuttering we may still have a trace with several occurrences of the same abstract activity, which

actually correspond to one abstract event. An example of this is shown in Figure 6, where parallel high-level transitions $a1$ and $a2$ contain low-level transitions $t1, t2$ and $t3, t4$ respectively. Suppose the log contains the trace fragment: $< ..., t1, t3, t2, t4, ... >$. After substituting the corresponding abstract activities we get $< ..., a1, a2, a1, a2, ... >$. There is no stuttering, but two occurrences of $a1$ in the trace in fact correspond to one firing. The same is true for $a2$.

Thus, interleaving generates several occurrences of an abstract activity in a trace, but a repetition of activities may be caused also by cyclic behavior. To separate the concerns we suppose in this paper, that our system does not contain cycles. This can be easily checked: in acyclic system there are no repeated events in low-level logs.

So, we would like to construct an abstract log — a high-level version of a given low-level log with the following properties:

1. each abstract activity occurs not more than one time in each trace;
2. replacing of low-level activities by abstract ones should preserve interleaving of concurrent activities.

For that we propose the following solution. Each trace σ with k occurrences of the same abstract activity e is split into k clones of σ, where each clone is obtained by deleting all except one occurrences of e in σ. This is done for all repeated activities.

Let us illustrate this procedure by a small example. Suppose that after replacing low-level activities by abstract activities we have got two traces:
$\sigma_1 =< e0, e1, e3, e1, e3, e1, e4, e1, e7 >$,
$\sigma_2 =< e0, e1, e3, e1, e3, e1, e4, e6 >$.

Then by cloning these traces we get:
$\sigma_1^1 =< e0, e1, e3, e4, e7 >$,
$\sigma_1^2 =< e0, e3, e1, e4, e7 >$,
$\sigma_1^3 =< e0, e3, e4, e1, e7 >$,
$\sigma_2^1 =< e0, e1, e3, e4, e6 >$,
$\sigma_2^2 =< e0, e3, e1, e4, e6 >$.

Figure 7 shows the transition system obtained by convolution of the obtained traces. To synthesize a model we can just apply one of the known discovery algorithms to this transition system.

The precise description of the algorithm, based on this idea, is given in the next section.

6. Algorithm for Synthesis of Abstract Process Model from a Low-Level Event Log

Let A be a set of abstract activities and L_r be an event log (a finite multiset of traces) over a set of low-level activities from A_r, where traces do not contain repetitive activities. Let $\delta : A_r \to A$ be a function, which maps every low-level activity to some high-level activity from A. Start with the empty transition system TS.

Step 1. Convert L_r into an event log L over the set of activities A by replacing each activity $a \in L_r$ in each trace with the activity $\delta(a)$.

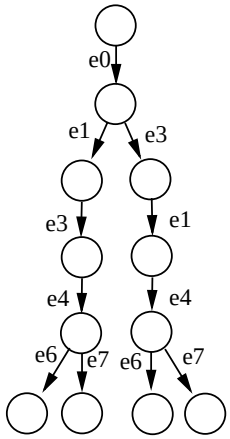


Fig 7. Transition system for traces σ_1 and σ_2

Step 2. Get rid of 'stuttering' in every $\sigma \in L$ by replacing in each trace each substring of consecutive entries of the same activity with one occurrence of this activity.

Step 3. For each trace $\sigma \in L$ do:

Step 3.1 Check whether there are more than one occurrences of the same activity in σ. If there are such occurrences, go to the Step 3.2, otherwise go to the Step 3.3.

Step 3.2 For each occurrence of repetitive activity e create a new trace by deleting all other occurrences of e in σ and go to Step 3.1.

Step 3.3 Add the new trace to transition system TS using the complete prefix sequence representation of a state.

Step 4 Apply an existing synthesis algorithm to the obtained transition system TS.

The correctness of the algorithm is justified by the following statements.

Lemma 1. *Let L_r be a low-level log without repetitive activities in its traces. If after Step 2 in our algorithm a trace $\sigma \in L$ contains two occurrences of activity a_1 and an occurrence of activity a_2 somewhere between two occurrence of a_1, then activities a_1 and a_2 are concurrent.*

Proof. Each occurrence of an abstract activity in a trace from L after Step 2 is obtained by replacing some low-level activity, corresponding to this abstract activity. So, occurrence of a_2 between two occurrences of a_1 means that at least a part of a_2 was executed, when execution of a_1 has started, but has not finished. Thus a_1 and a_2 were executed concurrently. □

Theorem 1. *Let N_{abs} be a WF-net with a set of activities A_{abs}, and let N_{ref} be a refinement of N_{abs} with a set of activities A_{ref} via substitutions $[a_1 \to N_r^1, a_2 \to N_r^2, \ldots, a_k \to M_r^k]$. Let also L_r be a log over the set of activities A_{ref}, and let L_r perfectly fit N_{ref}. Then each trace, added to the constructed transition system at Step 3.3, perfectly fits N_{abs}.*

Proof. Let $\sigma \in L$ be a trace added to transition system TS at Step 3.3. Then σ is obtained from some trace $\sigma_r \in L_r$ by replacing low-level activities with corresponding abstract activities and removing some repetitions of activities. Trace σ_r perfectly fits N_{ref}, and hence can be replayed with this model. Then σ be replayed with N_{abs} in parallel with replaying σ_r with N_{ref}. An activity a_i with the only occurrence at Step 3.1 of the algorithm will be replayed in parallel with a sequence of activities of subnet N_r^i in σ_r, and since A_{ref} is a refinement of N_{abs} via substitution $a_i \to N_r^i$, this replaying will be correct. If an activity a_j had multiple occurrences at this step, and the trace σ has retained one of this occurrences, it can be still replayed, since by lemma a_j is concurrent to its adjacent activities in σ, and concurrent activities can be replayed in any order. □

It can be also shown that the size of the constructed high-level transition system does not exceed the size of the low-level transition system, since the depth of the tree is shorter after abstraction. Note that if we want to keep real relative significance [15] of each arc in our model, we have to take into account that every trace which we generate at the Step 2.3 has an integer significance factor. This coefficient depends on the number of traces which are generated from one original trace from L_r (for k generated trace a significance factor equals to $1/k$ for each new trace).

7. Conclusion

Information system usually generate detailed event logs, which are not easy to work with. Detailed models discovered from these logs are often intricate and confusing. Abstract models are much more clear and more convenient for experts. So, the problem of discovering an abstract, high-level model from a low-level event log is important for simplification of the experts work on analysis and enhancement of information systems.

In this paper we provide a discovery technique for solving this problem, which is based on transforming a low-level event log into an abstract transition system and then applying one of already known methods for Petri net synthesis. In the further research we plan to evaluate this technique on artificial and real logs, using the fitness criteria presented earlier in [27].

References

[1] W. M. P. van der Aalst, *Process mining: discovery, conformance and enhancement of business processes*, Springer Verlag, 2011.

[2] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, W. M. P. van der Aalst, "The prom framework: A new era in process mining tool support", *International Conference on Application and Theory of Petri Nets*, Springer, 2005, 444–454.

[3] W. M. P. van der Aalst, "Verification of workflow nets", *18th International Conference on Application and Theory of Petri Nets, ICATPN'97*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, 407–426.

[4] S. Smirnov, H. A. Reijers, M. Weske, Th. Nugteren, "Business process model abstraction: a definition, catalog, and survey", *Distributed and Parallel Databases*, **30**:1 (2012), 63–99.

[5] A. Polyvyanyy, S. Smirnov, M. Weske, "Process model abstraction: A slider approach", *Enterprise Distributed Object Computing Conference, 2008* (EDOC'08. 12th International IEEE), IEEE, 2008, 325–331.

[6] K. Jensen, L. M. Kristensen, *Coloured Petri nets: modelling and validation of concurrent systems*, Springer Science & Business Media, 2009.

[7] H. J. Genrich, K. Lautenbach, "System modelling with high-level petri nets", *Theoretical computer science*, **13**:1 (1981), 109–135.

[8] W. M. P. van der Aalst, V. Rubin, B. F. van Dongen, E. Kindler, Ch. W. Günther, "Process mining: A two-step approach using transition systems and regions", *BPM Center Report BPM-06-30, BPMcenter. org*, **6**, 2006.

[9] A. J. M. M. Weijters, W. M. P. van der Aalst, A. K. A. De Medeiros, "Process mining with the heuristics miner-algorithm", *Technische Universiteit Eindhoven, Tech. Rep. WP*, **166**, 2006, 1–34.

[10] B. F van Dongen, N. Busi, G. Pinna, W. M. P. van der Aalst, "An iterative algorithm for applying the theory of regions in process mining" (Proceedings of the workshop on formal approaches to business processes and web services (FABPWS'07)), 2007.

[11] J. Carmona, J. Cortadella, M. Kishinevsky, "A Region-Based Algorithm for Discovering Petri Nets from Event Logs", *International Conference on Business Process Management*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, 358–373.

[12] W. M. P. van der Aalst, A. K. A. de Medeiros, A. J. M. M. Weijters, "Genetic process mining", *International Conference on Application and Theory of Petri Nets*, Springer, 2005, 48–69.

[13] A. J. M. M. Weijters, W. M. P. van der Aalst, "Rediscovering workflow models from event-based data using little thumb", *Integrated Computer-Aided Engineering*, **10**:2 (2003), 151–162.

[14] G. Greco, A. Guzzo, L. Pontieri, "Mining taxonomies of process models", *Data & Knowledge Engineering*, **67**:1 (2008), 74–102.

[15] Ch. W. Günther, W. M. P. van der Aalst, "Fuzzy mining–adaptive process simplification based on multi-perspective metrics", *International Conference on Business Process Management*, Springer, 2007, 328–343.

[16] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, P. J. Toussaint, "From Low-Level Events to Activities – A Pattern-Based Approach", *Business Process Management: 14th International Conference, BPM 2016*, Springer International Publishing, Cham, 2016, 125–141.

[17] N. Tax, N. Sidorova, R. Haakma, W. M. P. van der Aalst, "Event abstraction for process mining using supervised learning techniques", *Proceedings of the SAI Conference on Intelligent Systems (IntelliSys)*, 2016, 161–170.

[18] J. Li, R. P. J. Ch. Bose, W. M. P. van der Aalst, "Mining context-dependent and interactive business process maps using execution patterns", *International Conference on Business Process Management*, Springer, 2010, 109–121.

[19] R. P. J. Ch. Bose, E. H. M. W. Verbeek, W. M. P. van der Aalst, "Discovering hierarchical process models using prom", *Forum at the Conference on Advanced Information Systems Engineering (CAiSE)*, Springer, 2011, 33–48.

[20] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", *IEICE Transactions on information and Systems*, **80**, 1997, 315–325.

[21] A. Kalenkova, I. Lomazova, "Discovery of cancellation regions within process mining techniques", *Fundamenta Informaticae*, **133**:2–3 (2014), 197–209.

[22] A. Kalenkova, I. Lomazova, W. M. P. van der Aalst, "Process Model Discovery: A Method Based on Transition System Decomposition", *International Conference on Application and Theory of Petri Nets*, Springer International Publishing, Cham, 2014, 71–90.

[23] R. P. J. Chandra Bose, W. M. P. van der Aalst, "Process diagnostics using trace alignment: opportunities, issues, and challenges", *Information Systems*, **37**:2 (2012), 117–141.

[24] Th. Baier, J. Mendling, "Bridging abstraction layers in process mining by automated matching of events and activities", *Business Process Management*, Springer, 2013, 17–32.

[25] J. Desel, W. Reisig, "The synthesis problem of petri nets", *Acta informatica*, **33**:4 (1996), 297–315.

[26] A. Rozinat, *Process mining: conformance and extension*, PhD thesis, Technische Universiteit Eindhoven, 2010.

[27] A. Begicheva, I. Lomazova, Does your event log fit the high-level process model? *Modeling and Analysis of Information Systems*, **22**:3 (2015), 392–403.

Бегичева А. К., Ломазова И. А., "Построение высокоуровневой модели процесса по журналу событий", *Моделирование и анализ информационных систем*, **24**:2 (2017), 125–140.

DOI: 10.18255/1818-1015-2017-2-125-140

Аннотация. Извлечение и анализ процессов (process mining) — это достаточно новая область компьютерных наук, изучающая синтез и анализ процессов на основе журналов событий. В работе рассматривается задача извлечения высокоуровневой модели по низкоуровневому журналу событий, т.е. задача автоматического синтеза модели процесса на основе информации, хранящейся в журналах событий информационной системы. События в высокоуровневой модели — это абстрактные события, которые могут быть детализированы в виде низкоуровневых подпроцессов, поведение которых представлено в журналах событий. Синтез моделей интенсивно изучается в рамках исследований по майнингу процессов, но в основном в литературе рассматриваются только логи и модели одного и того же уровня детализации. Здесь мы представляем алгоритм для извлечения высокоуровневых ациклических моделей процессов на основании журналов событий и заранее определенного разбиения низкоуровневых событий на подмножества, ассоциированные с абстрактными событиями в высокоуровневой модели.

Ключевые слова: сети Петри, высокоуровневые модели процессов, журналы событий, Process Mining, синтез моделей

Об авторах:
Бегичева Антонина Константиновна, стажер-исследователь,
Национальный исследовательский университет «Высшая школа экономики»,
Научно-учебная лаборатория ПОИС,
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: akbegicheva@edu.hse.ru

Ломазова Ирина Александровна, доктор физ.-мат. наук, профессор,
Национальный исследовательский университет «Высшая школа экономики»,
ул. Мясницкая, 20 г. Москва, 101000 Россия, e-mail: ilomazova@hse.ru

Благодарности:
Работа выполнена при поддержке Программы фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» и Российского фонда фундаментальных исследований, проект 16-01-00546.