Transition Systems Reduction: Balancing Between Precision and Simplicity

Sergey A. Shershakov(✉), Anna A. Kalenkova, and Irina A. Lomazova

National Research University Higher School of Economics,
20 Myasnitskaya Street, 101000 Moscow, Russia
sshershakov@hse.ru
http://pais.hse.ru

Abstract. Transition systems are a powerful formalism, which is widely used for process model representation. A number of approaches were proposed in the process mining field to tackle the problem of constructing transition systems from event logs. Existing approaches discover transition systems that are either too large or too small. In this paper we propose an original approach to discover transition systems that perfectly fit event logs and whose size is adjustable depending on the user's need. The proposed approach allows the ability to achieve a required balance between simple and precise models.

Keywords: Transition systems · Process mining · Model reduction · Process model quality

1 Introduction

Process mining is a relatively new discipline, whose basic research and practical purpose is to extract process models from data given in the form of *event logs*, checking existing models for conforming to actual processes and improving them. *Transition systems* are extensively used to formalize processes extracted from event logs. A transition system can be constructed from an event log by using prefix-based techniques in a very natural way [3]. We consider several metrics that describe the model's quality [11]. Replay fitness quantifies the extent to which a process model can reproduce the behavior recorded in a log. Complexity of the model is estimated by simplicity and precision (the *metrics*, which shows how precise the model is in respect to the event log).

The major weakness of models constructed from real-life event logs is their size. Despite the fact that there are a number of approaches aimed to reduce the size of transition systems [3], application of the existing approaches results in either too large or too small models. In the former case the model size is too

This work is supported by the Basic Research Program at the National Research University Higher School of Economics in 2017 and the study was funded by RFBR and Moscow city Government according to the research project No. 15-37-70008 "mol_a_mos".

big to be readable. Beyond this point, it becomes difficult or even impossible to apply existing transition system analysis techniques that are sensitive to the size of input models. Despite the fact that there are polynomial synthesis algorithms for some classes of Petri nets, such as elementary nets [6], in general, the problem of synthesizing Petri nets is NP-complete [7]. Thus, the applicability of the state-based regions algorithm [12] is limited to fairly small models. In the latter case, due to states merging, a rather small model facilitates too much behavior that cannot be observed in the log, which makes the model less precise and thus less applicable.

A problem of a significant impact of the size of a transition system on the execution time of the regions algorithm was examined in detail in a number of papers, for example in [17,18]. Despite the fact that in this paper we do not immediately consider the task of synthesizing Petri nets from transition systems, such a task, however, is meant to be the final goal of this research. This paper is positioned as an intermediate step, which focuses precisely on the reduction of transition systems.

Several approaches for the reduction of transition systems were proposed previously. In this paper, we present an original reduction approach based on the frequency characteristics of traces. Unlike in the previous studies, efficiency of the proposed method is not estimated on the basis of the execution time of algorithms, but based on quality metrics obtained during the reduction. To this end, for calculating quality metrics of transition systems we propose new algorithms that have not previously been described.

The main goal of our study is to develop an approach for reducing the size of a transition system mined from an event log in a flexible manner. This paper describes an original 3-step algorithm achieving the goal by using a variable-size window based on a state frequency characteristic. The approach preserves the (perfect) fitness of a model and balances between its simplicity and precision by introducing a set of adjustable parameters.

Thus, the *main contributions* are as follows: (1) an original method for reducing transition systems and justification of its applicability; (2) a set of experimental results which show the advantages of the proposed approach as compared with existing methods; an openly available proof of the concept through implementation in a set of ProM plug-ins.

The remaining part of the paper is organized as follows. Section 2 gives an overview of related work in the context of inferring transition systems and their application in the process mining domain. Section 3 introduces basic concepts used further in this paper. A detailed description of the proposed algorithm is given in Sect. 4. A novel precision calculation algorithm, some significant implementation details, and experimental results are discussed in Sect. 5. Finally, Sect. 6 concludes the paper and discusses some directions for future work.

2 Related Work

A number of works concerning inferring transition systems from event traces exist. Biermann and Feldman in their work [10] proposed a k-tails algorithm which merges states of FSMs by basing on the similarity of behavior of the states. The algorithm falls into the class of prefix tree merging methods. Angluin [5] proposed a method of prefix tree states merging based on a notion of k-reversibility. In [15], Lorenzoli et al. proposed a GK-tail approach, an extension of the k-tail algorithm, dealing with parametrized finite state automata.

Cook and Wolf in their work [13] introduced *process discovery*, a new data analysis technique in the context of software engineering processes. They considered automatic generation of a formal model describing an ongoing process from captured event data. A new Markov method was developed specifically for this purpose. Moreover, two existing methods, the k-tail and RNet (based on neural networks) ones, were adopted for the process discovery technique.

Process discovery along with *conformance checking* and *process enhancement* form the basis of *process mining* [4], which deals with various types of process models including Petri nets, transition systems, fuzzy maps, C-nets, BPMN and others. In the context of process mining, transition systems are considered both a self-independent model and an intermediate model for building another type of model on this basis. In the latter case, one should mention region-based approaches discussed in [8,12,14,18].

The leading role of a transition system as an intermediate representation of a process is discussed in [3]. The paper considers a number of different strategies to construct a transition system that is more suitable to be a base for a resulting final Petri net model with respect to desirable metrics. Nevertheless, all discussed strategies are based on inferring algorithms with a fixed window.

Besides the strategies above, a number of other approaches for the transition system reduction were proposed. Most of them are based on merging of related states. In [17] authors consider an abstraction technique named *common final marking* (CFM). The technique involves merging all states without outgoing arcs (so-called *sink states*) into a single state.

In [18] authors propose an approach for compacting a transition system based on aggressive folding techniques. The proposed technique allows state space reduction through the detection of unfolded cycles in an acyclic transition system and its subsequent folding. Distinctive features of our approach are discussed in Sect. 5.3.

Another approach for the process discovery involves language-based methods which are applied directly to logs without constructing intermediate models [9,20]. We do not consider these methods here.

3 Preliminaries

This section introduces basic concepts related to event logs, transition systems and some other notations that are needed for explaining the approach.

$\mathcal{B}(X)$ is the set of all multisets over some set X. For some multiset $b \in \mathcal{B}(X)$, $b(x)$ denotes the number of times element $x \in X$ appears in b. Thus, $x \in b$ iff $b(x) > 0$. By $b = [x_1, x_2^3, x_3^5]$ we denote that elements $x_1, x_2, x_3 \in X$ appear in b one, three and five times respectively. We say that $b' \subseteq b$ iff $\forall x \in X : b'(x) \leq b(x)$.

The size of a multiset b over set X is denoted by $|b|$ and defined as $|b| = \sum_{x \in X} b(x)$. For a given set Y, Y^+ is the set of all non-empty finite sequences over Y.

Definition 1 (Trace, Event Log). *Let A be a set of activities. A* trace *is a finite sequence $\sigma = \langle a_1, a_2, ..., a_i, ..., a_n \rangle \in A^+$. By $\sigma(i) = a_i$ we denote i-th element of the trace. The $[i,k]$*-subtrace *of trace σ ended at the i-th activity a_i is defined as*

$$\sigma[i,k] = \begin{cases} \langle \sigma(1), \sigma(2), ..., \sigma(i) \rangle, & \textit{if } k > i; \\ \langle \sigma(i-k+1), ..., \sigma(i) \rangle, & \textit{if } 1 \leq k \leq i; \\ \langle \rangle, & \textit{if } k = 0. \end{cases} \tag{1}$$

The complete subtrace *of the trace σ ended at i-th event a_i is $\sigma[i] = \sigma[i,i]$. By $|\sigma|$ we denote trace length. For $k \leq i$, k denotes the length of the subtrace. $L \in \mathcal{B}(A^+)$, such that $|L| > 0$, is an* event log*. Here, $|L|$ is the number of all traces.*

We assume, that event logs do not contain process states explicitly. This way, we need to deduce the desirable states from an event log based on some approach.

In [3], four approaches to determine the state in a log were proposed. They are *past*, *future*, *past and future* and *explicit knowledge*. In this paper we consider only the *past* approach, according to which a state is constructed based on the *prefix* of a trace. Then, the order of activities is important. Hence, we apply sequence policy [3] for determining a state.

Definition 2. *A* labeled transition system *is a tuple $TS = (S, E, T, s_0, AS)$, where S is a finite* state space*, E is a finite* set of labels*, $T \subseteq S \times E \times S$ is a* set of transitions*, $s_0 \in S$ is an initial state, and $AS \subseteq S$ is a set of* accepting (final) states*. We denote the set of* output *(*input*) transitions of a state $s \in S$ as $s\bullet = \{t = (s, e, s') \in T \mid e \in E, s' \in S\}$ ($\bullet s = \{t = (s', e, s) \in T \mid e \in E, s' \in S\}$).*

By $TS(L) = (S, E, T, s_0, AS)$, where $L \in \mathcal{B}(A^+)$ is an event log, we denote a transition system, such that $E = A$.

Let $\sigma = \langle a_1, ..., a_n \rangle$ be a trace ($\sigma \in L$) and $n = |\sigma|$. We say that trace σ can be *replayed* in the transition system $TS(L)$ if there is a sequence of states $\langle s_0, ..., s_n \rangle$ such that $\exists t_1 = (s_0, a_1, s_1), t_2 = (s_1, a_2, s_2), ..., t_n = (s_{n-1}, a_n, s_n)$, where $s_0, s_1, ..., s_n \in S, t_1, t_2, ..., t_n \in T$. We denote this as $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_n} s_n$. We say that trace σ can be *partially replayed* by its prefix in a transition system $TS(L)$ if $\exists k < n : \; s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_k} s_k$ and $\nexists s_k \xrightarrow{a_{k+1}} s_{k+1} \xrightarrow{a_{k+2}} ... \xrightarrow{a_n} s_n$. We denote $\sigma^+(TS(L)) = \langle a_0, a_1, ..., a_k \rangle$ and $\sigma^-(TS(L)) = \langle a_{k+1}, ..., a_n \rangle$. Hence, $\sigma(TS(L)) = \sigma^+(TS(L)) + \sigma^-(TS(L))$ where $+$ denotes concatenation of two sequences.

Definition 3 (k-window transition system). *Let $L \in \mathcal{B}(A^+)$ be a log over set of activities A and let $k \in \mathbb{N}$ be a natural number called* window size. *$TS(L) = (S, E, T, s_0, AS)$ is a k-window (labeled) transition system built for log L and window size k, where $S = \{\sigma[i,k] \mid \sigma \in L, 0 \leq i \leq |\sigma|, k \leq i\}$, $s_0 = \sigma[0]$*[1]*, $T = \{(\sigma[i-1,k], \sigma(i), \sigma[i,k]) \mid \sigma \in L, 1 \leq i \leq |\sigma|\}$, $AS = \{s \in S \mid s = \sigma[|\sigma|, k], \sigma \in L\}$.*

Definition 4 (Full transition system). *Let $L \in \mathcal{B}(A^+)$ be a log over set A of activities. The* full transition system *$TS(L) = (S, E, T, s_0, AS)$ for log L is a labeled transition system built for log L, where $S = \{\sigma[i] \mid \sigma \in L, 0 \leq i \leq |\sigma|\}$, $s_0 = \sigma[0]$, $T = \{(\sigma[i-1], \sigma(i), \sigma[i]) \mid \sigma \in L, 1 \leq i \leq |\sigma|\}$, $AS = \{s \in S \mid s = \sigma[|\sigma|], \sigma \in L\}$.*

To measure the quality of transition systems we consider three quality metrics. We based them primarily on the work [11], where metric definitions are given for *process trees*. In this paper we adopted them for *transition systems*. *Fitness* quantifies the extent to which a transition system can reproduce traces recorded in a log. *Simplicity* quantifies the complexity of a model. Simplicity is measured by comparing the size of a given transition system $TS(L)$ with the simplest possible transition system, which is the flower model (Fig. 1a). *Precision* compares the transition system $TS(L)$ with the full transition system built for log L, considering the latter to be the most precise. Such a comparison is done through a *simulation* of the full transition system by $TS(L)$ (for further details see Sect. 5.1).

Definition 5 (Metrics). *Let L be an event log and let $TS(L) = (S, E, T, s_0, AS)$ be a transition system built for L.* Fitness *is defined to be the ratio of the number of traces from log L that can be fully replayed in transition system $TS(L)$ to the total number of all traces. Log L* perfectly fits *transition system $TS(L)$ iff all traces of L can be fully replayed in $TS(L)$.*

Simplicity *of $TS(L)$ is:*

$$Simpl(TS(L)) = \frac{|E| + 1}{|T| + |S|}.$$

Precision *of $TS(L)$ is:*

$$Prec(TS(L)) = \frac{1}{|S|} \cdot \sum_{s \in S} Prec(s), \quad Prec(s) = \frac{1}{NoV(s)} \cdot \sum_{i=1}^{NoV(s)} \frac{|s\bullet| - |\widehat{s\bullet}_i|}{|s\bullet|},$$

where $Prec(s)$ is a partial precision *for a state s, $NoV(s)$ is a number of all visits of state s during a simulation of the reference full transition system by $TS(L)$. $\widehat{s\bullet}_i$ is a set of such output transitions of state s that cannot be activated ("fired") during the i-th visit of state s. These transitions do not have active counterparts in the reference full transition system.*

[1] Note, that $s_0 \in S$, since event log L contains at least one trace by Definition 1.

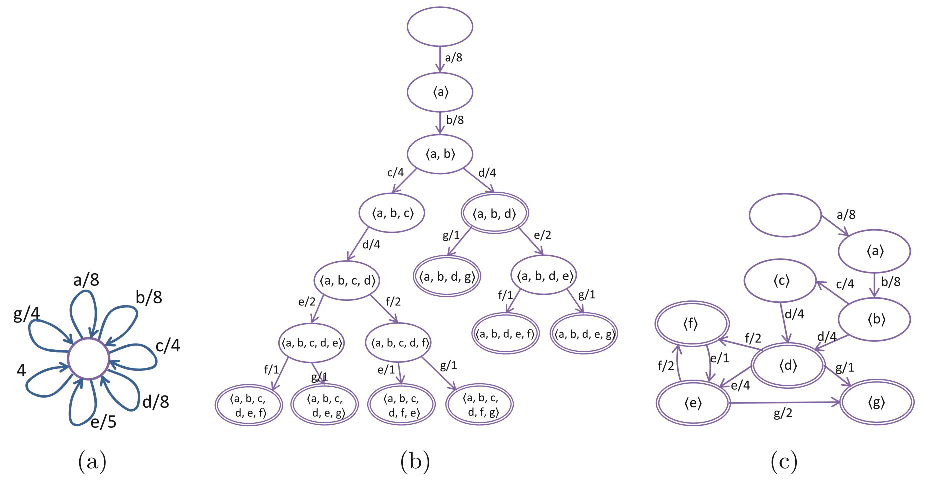

Fig. 1. (a) "Flower" model for log L_1; (b) $TS_1(L_1)$ built for log L_1; (c) transition system built for log L_1 with a fixed window of size 1. In the figures, each transition is labeled by an activity and its corresponding frequency characteristic (see. Definition 6)

4 Algorithm Description

For the clarification of the approach, the following motivating example is considered. Let L_1 be an event log that is defined as follows:

$$L_1 = [\langle a,b,c,d,e,f\rangle, \langle a,b,c,d,e,g\rangle, \langle a,b,c,d,f,e\rangle, \langle a,b,c,d,f,g\rangle, \\ \langle a,b,d\rangle, \langle a,b,d,g\rangle, \langle a,b,d,e,f\rangle, \langle a,b,d,e,g\rangle] \quad (2)$$

In addition to the previously discussed flower model, one can build a number of other models that perfectly fit log L_1. A model built with unlimited window size is depicted in Fig. 1b. This model is a *full transition system* by the definition. Another model built by an algorithm with a fixed window of size 1 is depicted in Fig. 1c. Although all these models perfectly fit log L_1, none of them is satisfactory in simplicity and precision at the same time. Thus, we are interested in a trade off between these metrics.

The proposed approach incorporates a 3-steps algorithm sequentially building 3 transition systems. The first transition system, TS_1, is built from an event log. The second (TS_2) and the third (TS_3) transition systems are built from TS_1 and TS_2, respectively. Finally, TS_3 is considered as a desirable result.

The main point of the proposed approach is dynamic variation of the window used for deducing states. For this very purpose, two adjustable parameters are involved in the approach. The first one, *Threshold*, affects the size of the intermediate transition system (TS_2). The second one, *Vwsc*, is a linear factor used for the dynamic calculation of a variable window size while building the resulting model (TS_3). Each step of the algorithm along with both parameters is thoroughly discussed in the following sections.

4.1 Constructing a Full Transition System (Step 1)

The first step of the approach is to construct a full transition system and define a special labeling function mapping every transition to a natural number that determines its *frequency characteristic.*

Definition 6 (Frequency characteristic). *Let* $L \in \mathcal{B}(A^+)$ *be a log over set of activities* A *and let* $TS(L) = (S, E, T, s_0, AS)$ *be the full transition system for* L*. The frequency characteristic of* $TS(L)$ *is a function* $f : T \to \mathbb{N}$ *defined for* $t = (\sigma[j-1], \sigma(j), \sigma[j])$ *as* $f(t) = |L'|$*, where* L' *is the maximum multiset over* A^**, such that* $L' \subseteq L$ *and* $\forall \sigma' \in L' \;\; \sigma'[j] = \sigma[j]$*.*

Frequency characteristic determines for every transition t a number of traces in log L that start with prefixes $\sigma[j]$. The entire procedure of building a full transition system is presented in Algorithm 1.

From now on, we denote the full transition system for a given log L as $TS_1(L)$.

$TS_1(L_1)$ built for log L_1 with function f is depicted in Fig. 1b; it is a tree by construction. It is easy to see that fitness of the full transition system ($TS_1(L)$) is perfect (equals 1). This is inherent in the algorithm, since it builds for each trace in, a log a full chain of states following one after another, that corresponds to a sequence of activities in the trace.

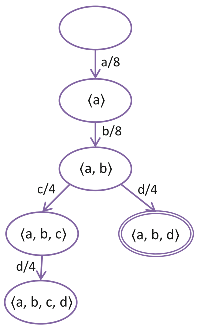

Fig. 2. $TS_2(L_1)$ (condensed) built from $TS_1(L_1)$ with $Threshold = 0.33$

4.2 Constructing a Condensed Transition System (Step 2)

The second step of our approach involves cutting some branches of the full transition system with frequency values which are less than a *cutting threshold* parameter. We refer to it as f_1. Once f_1 is set, all the states and transitions corresponding to a rarely observed behavior in the event log can be removed from the model. This results in simplifying the tree structure and reduction of the number of states and transitions.

Definition 7 (Condensed Transition System). *Let* $TS_1(L) = (S_1, E_1, T_1, s_0, AS_1)$ *be a full transition system constructed for log L and let* f *be a frequency characteristic. The Threshold is a real number from* $[0;1]$ *determining a cutting threshold* f_1 *as follows:* $f_1 = round(|L| \cdot Threshold) - 1$*. The value* $f_1 + 1 = round(|L| \cdot Threshold)$ *is a* minimum preserved frequency *for* $TS_1(L)$*.*

A condensed transition system $TS_2(L)$ *built for a* $TS_1(L)$ *with function* f *and a given cutting threshold* f_1 *is the transition system* $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$*, where* $S_2 \subseteq S_1$*,* $E_2 = E_1$*,* $T_2 \subseteq T_1$*,* $AS_2 \subseteq AS_1$ *and* $T_2 = \{t \mid t \in T_1 \;\wedge\; f(t) > f_1\}$*,* $S_2 = \{s_0\} \cup \{s \mid s \in S_1 \;\wedge\; \exists t = (s', a, s) \in T_2\}$*,* $AS_2 = AS_1 \cap S_2$*.*

Algorithm 1. Building a full transition system for a given log L

```
Input  : an event log $L$
Output : a full transition system;
$TS_1(L) = (S, E, T, s_0, AS)$; $f$ is a frequency characteristic;
begin
    $S \leftarrow \{s_0\}$;
    for $\sigma \in L$ do
        $s \leftarrow s_0$;
        for $i \leftarrow 1$ to $|\sigma|$ do
            $s' \leftarrow \sigma[i]$;
            $t \leftarrow (s, \sigma(i), s')$;
            if $t \notin T$ then
                $T \leftarrow T \cup \{t\}$;
                $f(t) \leftarrow 1$;
            else
                $f(t) \leftarrow f(t) + 1$;
            $S \leftarrow S \cup \{s'\}$;
            $E \leftarrow E \cup \{\sigma(i)\}$;
            if $i = |\sigma|$ then
                $AS \leftarrow AS \cup \{s\}$;
            $s \leftarrow s'$;
```

Algorithm 2. Building a condensed transition system $TS_2(L)$

```
Input  : an event log $L$;
a full transition system
$TS_1(L) = (S_1, E_1, T_1, s_0, AS_1)$; $f$ is a frequency characteristic;
$Threshold$ is a real number determining a cutting threshold and a minimum preserved frequency;
Output : $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$ is a condensed transition system;
begin
    $f_1 = round(|L| \cdot Threshold) - 1$;
    for $t \in T_1$ do
        if $f(t) > f_1$ then
            $T_2 \leftarrow T_2 \cup \{t\}$;
    $S_2 \leftarrow \{s_0\}$;
    for $t = (s, a, s') \in T_2$ do
        $S_2 \leftarrow S_2 \cup \{s'\}$;
    $E_2 \leftarrow E_1$;
```

Note that the frequencies of transitions diminish on the way to the leaves of $TS_1(L)$ and $TS_2(L)$. Hence, the exclusion of transition t_k from $TS_1(L)$ implies the exclusion of a total subtree that has state s_k as a root. Thus, $TS_2(L)$ obtained as a result of cutting with a given threshold, cannot be disconnected.

The entire procedure of constructing a condensed transition system is presented in Algorithm 2. For log L_1 with the size $|L| = 8$ and $Threshold = 0.33$, we have $f_1 = 2$. A $TS_2(L)$ built for the log L_1 and $f_1 = 2$ is depicted in Fig. 2. It is easy to see that not all the traces from the log can be replayed on $TS_2(L)$ as its fitness is not perfect. Therefore, we cannot consider this model as a final result.

4.3 Constructing a Reduced Transition System (Step 3)

In this section, we propose an approach to convert $TS_2(L)$ to a model with perfect fitness and a size that is less than the size of $TS_1(L)$.

Our proposal is to construct a new transition system $TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$ based on $TS_2(L)$ by adding missing states and transitions in order to fully replay all the traces. Unlike building the full transition system, in this case we use partial subtrace $\sigma[i, k]$ for representing newly added states of $TS_3(L)$. The important point here is that parameter k, is proportional to the frequency of a corresponding input transition.

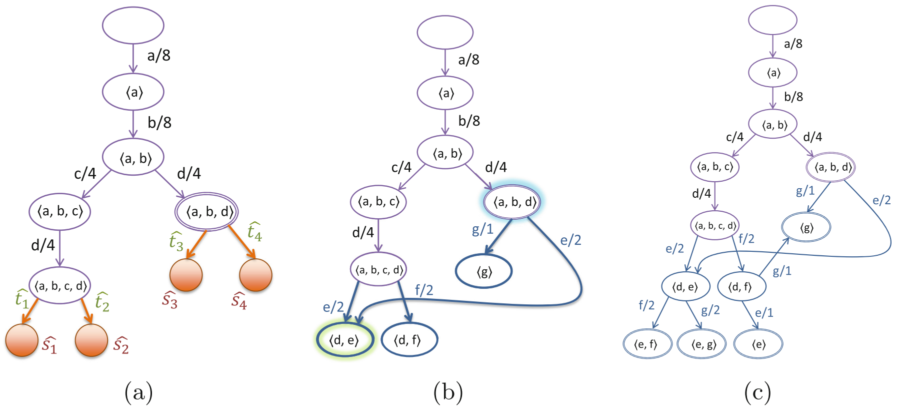

Fig. 3. (a) TS_3 under a restoring algorithm: temporary states and transitions after *the first stage*; (b) restored states and transitions after *the second stage* (c) $TS_3(L_1)$ (reduced) built from $TS_2(L_1)$ with $\mathit{Threshold} = 0.33$ for log L_1

The main part of the algorithm implementing the proposed approach is represented in Algorithm 3. In the beginning, we create a full copy of $TS_2(L)$, which is denoted as $TS_3(L)$. $TS_3(L)$ is a target transition system that is iteratively reconstructed by the following two stages of the algorithm.

At the *first stage*, implemented as function `ReplayTrace`, the algorithm tries to replay as many traces σ from event log L as possible. If all traces can be fully replayed, the algorithm successfully stops. Otherwise, if there is at least one trace that cannot be fully replayed, the *second stage* is performed. This stage is implemented as `RestateTS` procedure which tries to reconstruct the unreplayable parts of traces by adding new states and transitions to the target $TS_3(L)$.

The algorithm defines some auxiliary objects as follows. A special function ξ maps every trace σ onto number j that determines element $\sigma(j)$ splitting σ into σ^+ and σ^-, where $\sigma(j)$ is the first element of trace σ^-. A set TT of temporary transitions. A set of completely replayed traces $CompleteTraces \subseteq L$.

The stages of the algorithm are discussed below in more detail. For log L_1, we consider transition system $TS_3(L_1)$ copied from $TS_2(L_1)$ (Fig. 2). Then, the algorithm tries to replay log L_1 over $TS_3(L_1)$ and perform its transformation.

First stage (`ReplayTrace`). Let $\sigma = \langle a, b, c, d, e, f \rangle$. The longest prefix of σ that can be successfully replayed is $\sigma^+(TS_3(L_1)) = \langle a, b, c, d \rangle$. Correspondingly, the only suffix of σ that cannot be replayed is $\sigma^-(TS_3(L_1)) = \langle e, f \rangle$.

Algorithm 4 replays a single trace σ over transition system $TS_3(L)$ and also gets as its input a frequency characteristic f (discussed above). The algorithm starts with initial state s_0 as a current state s and the first element $\sigma(1)$ of a trace as a current element $\sigma(i)$. Then it tries to find an appropriate transition t starting with current state s and marked by symbol $\sigma(i)$.

Algorithm 3. Building a reduced transition system $TS_3(L)$

```
Input  : an event log $L$;
a condensed transition system $TS_2(L) = (S_2, E_2, T_2, s_0, AS_2)$; a frequency
characteristic $f$; a multiplicative factor $Vwsc \in \mathbb{R}$ for fixed window size;
Output : a reduced transition system $TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$;
Data: a set of completely replayed traces $CompleteTraces \subseteq L$;
a function mapping each trace to a number of first unreplayable symbol $\xi$;
a set of temporary transitions $TT$;
/* Main part of the algorithm                                    */
begin
    $TS_3(L) \leftarrow TS_2(L)$;
    repeat
        unreplayableTraces $\leftarrow false$;
        for $\sigma \in L$ do
            if ReplayTrace($\sigma$, $TS_3(L)$, $f$, $CompleteTraces$, $\xi$, $TT$) $= false$
             then
                unreplayableTraces $\leftarrow true$;
        if unreplayableTraces $= true$ then
            RestateTS ($L$, $TS_3(L)$, $f$, $CompleteTraces$, $\xi$, $Vwsc$, $TT$);
    until unreplayableTraces $= false$;
```

Figure 3 contains an example to illustrate the proposed approach. Next, there are three possible cases. In the *first case* (for instance, during the replay $\sigma = \langle a, b, d \rangle$), transition $t = (s, \sigma(i), s')$ exists with some state $s' \in S_3$ and this is a regular state ($t \notin TT$). In this case current state s is changed to s' and the next element $\sigma(i+1)$ is processed. In Fig. 3a, transition $t = (\langle a \rangle, b, \langle a, b \rangle)$ is an example of such a case[2].

In the *second case* (if $\sigma = \langle a, b, d, g \rangle$), t does not exist. For that case a new temporary transition $t = (s, \sigma(i), \widehat{s})$ is added to both the set of transitions T_3 and the set of temporary transitions TT. The end state $\widehat{s}$ is a special temporary state, too. It is not marked by any substring and is unique for any temporary transition. For the newly added transition t, the frequency characteristic f is defined to be equal to 1; it means transition t "fired" only once. In Fig. 3a, transition $t = \widehat{t_3} = (\langle a, b, d \rangle, g, \widehat{s_3})$ is an example of such a case.

In the *third case*, t exists and it is a temporary one ($t \in TT$). This is the case when transition t "fires" one more time; hence, its frequency should be increased by one.

In both the second and the third cases, replaying of the current trace σ is broken and $\xi(\sigma)$ is set to the position of the first unreplayable element. This is the reason why the temporary state $\widehat{s}$ cannot be marked with any subtrace by this moment.

[2] To be precise, transition t is marked in the figure not only with activity b but also with its frequency $b/8$.

Algorithm 4. Function `ReplayTrace` of the algorithm of building a reduced transition system $TS_3(L)$

```
Input  : trace $\sigma$;
reduced transition system
$TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$;
frequency characteristic $f$; set
of completely replayed traces
$CompleteTraces \subseteq L$; function
$\xi$ mapping each trace to a
number of the first
unreplayable symbol ; set of
temporary transitions $TT$;
Output : $true$, if a trace
         is replayed
completely, $false$ otherwise

/* Replays a trace */
Function ReplayTrace($\sigma$,
 $TS_3(L)$, $f$, $CompleteTraces$,
 $\xi$, $TT$): Boolean
    /* Already completed */
    if $\sigma \in CompleteTraces$
     then
        return $true$;
    $s \leftarrow s_0$;
    for $i \leftarrow 1$ to $|\sigma|$ do
        if $\exists s' : t =$
         $(s, \sigma(i), s') \in T_3$ then
            if $s' = \widehat{s}$ then
                $f(t) \leftarrow f(t) + 1$;
                $\xi(\sigma) = i$;
                return $false$;
            $s \leftarrow s'$;
        else
            $S_3 \leftarrow S_3 \bigcup \{\widehat{s}\}$;
            $t \leftarrow (s, \sigma(i), \widehat{s})$;
            $T_3 \leftarrow T_3 \bigcup \{t\}$;
            $TT \leftarrow TT \bigcup \{t\}$;
            $f(t) = 1$;
            $\xi(\sigma) = i$;
            return $false$;
    /* Trace's complete */
    $\xi(\sigma) = |\sigma| + 1$;
    return $true$;
```

Algorithm 5. Procedure `RestateTS` of the algorithm of building a reduced transition system $TS_3(L)$

```
Input  : log $L$;
reduced transition system
$TS_3(L) = (S_3, E_3, T_3, s_0, AS_3)$; frequency
characteristic $f$; set of completely
replayed traces $CompleteTraces \subseteq L$;
function $\xi$ mapping each trace to a
number of the first unreplayable symbol;
multiplicative factor for fixed window size
$Vwsc \in \mathbb{R}$; set of temporary transitions
$TT$;

/* Converts temporaries */
Procedure RestateTS($L$, $TS_3(L)$, $f$,
 $CompleteTraces$, $\xi$, $Vwsc$, $TT$)
    for $\sigma \in L$ do
        $i \leftarrow \xi(\sigma)$;
        /* If the trace is already
        complete */
        if $i = |\sigma| + 1$ then
            return ;
        $s \leftarrow \sigma[i-1]$;
        $t \leftarrow (s, \sigma(i), \widehat{s})$;
        if $t \notin TT$ then
            return ;
        $maxWndSize \leftarrow \max_{\sigma \in L}(|\sigma|)$;
        $wndSize \leftarrow round(maxWndSize \cdot$
         $f(t) \cdot Vwsc \div |L|)$;
        /* a special 'trash' state */
        if $wndSize = 0$ then
            $s' \leftarrow s_{0ws}$;
        else
            $s' \leftarrow \sigma[i, wndSize]$;
        $t' \leftarrow (s, \sigma(i), s')$;
        /* Replace the temporary
        transition and the state by
        regular ones */
        $S_3 \leftarrow S_3 \setminus \{\widehat{s}\} \cup \{s'\}$;
        $T_3 \leftarrow T_3 \setminus \{t\} \cup \{t'\}$;
        $TT \leftarrow TT \setminus \{t\}$;
        $f(t') \leftarrow f(t)$;
        if $i = |\sigma|$ then
            $AS_3 \leftarrow AS_3 \cup \{s'\}$;
    return ;
```

This way, at each iteration Algorithm 3 tries to replay as many traces from the log as possible. For each state the first unreplayable element is determined and a new temporary transition for the element, along with a temporary state, is built. $TS_3(L_1)$ with temporary transitions and states marked by symbols e, f, g and e are depicted in Fig. 3a. Note that the sum of the total frequencies for all temporary transitions is equal to the number of traces that could not be replayed in the first iteration.

Trace $\sigma_5 = \langle a, b, d \rangle$ from log L_1 is an example of a trace that can be replayed at the very first iteration. Once all the traces from the log can be replayed, the reconstruction of $TS_3(L)$ has successfully ended.

Second stage (`RestateTS`). As long as there is at least one temporary transition/state in $TS_3(L)$, it has to be converted to a regular one. This is done by Algorithm 5, which enumerates all uncompleted traces. For each such trace σ, the last regular state s, temporary transition t and temporary state $\widehat{s}$ are obtained; they correspond to the first unplayable element $\sigma(\xi(\sigma))$. Then state $\widehat{s}$ is converted to a regular state by being marked with subtrace $\sigma[\xi(\sigma), m]$ of trace σ ended by element $\sigma(\xi(\sigma))$ with length m that *is proportional to the frequency* of temporary transition t. In this way, such a state is marked with a *subtrace which is shorter* than that of the corresponding state in the full transition system. Note that temporary states and transitions are always converted to regular ones.

In Fig. 3b, transition $t = (\langle a, b, d \rangle, g, \langle g \rangle)$ is obtained from the former transition $\widehat{t_3}$. Unlike the original (full) transition system, the target state here is labeled by a shorter subtrace $\langle g \rangle$ instead of $\langle a, b, d, g \rangle$. It makes the state more "abstract" and facilitates the emergence of other states with the same label.

If a temporary state is marked with the same subtrace as one of the states existing in the model, both states are merged and the total number of states and transitions in the resulting transition system is decreased. Figure 3b shows how two states marked with subtrace $\langle d, e \rangle$ are merged to one state.

The frequency characteristic of some transitions can be relatively small for producing a window size equal to 0. In such situation the algorithm uses a dedicated state[3] s_{0ws}. This state accumulates all rare behavior patterns and prevents the appearance of unwanted states and transitions.

An important consequence of the presence of state s_{0ws} is that it is possible to obtain even *simpler* model than a model built with a fixed window of a size equal to 1 (Fig. 1c), and at the same time more *precise* one (see comparision in Sect. 5.3). Here, *simplicity* is achieved by removing a number of unimportant arcs and closing them as self-loops in state s_{0ws}. The maximum number of such self-loop arcs is limited by the number of all activities. A similar approach was applied in CFM method [17] by merging all sink states (states without outgoing arcs) into a single state s.

After the algorithm is finished, no temporary transitions and states are present in $TS_3(L)$ anymore. Moreover, all previously unreplayable elements in

[3] Formally, for a trace $\langle \rangle$, a state s_0 should be considered. Nevertheless, we explicitly distinguish the initial state s_0 and the rare-behavior state s_{0ws}.

the traces can now be replayed in $TS_3(L)$ as new states for them have been established. At the end of an iteration of the algorithm each trace from the log can be replayed at least by one more element than before the iteration. Since the length of each trace is finite, the number of iterations is also finite. Thus, the algorithm eventually stops. The resulting $TS_3(L_1)$ is depicted in Fig. 3c.

In Algorithm 3 *Vwsc* is an additional parameter used to combine a varying window size approach with a classical fixed window size approach. It is a real value from $[0, 1]$ determining a maximum size of a state window during the reconstruction phase of $TS_3(L)$.

Finally, considering the fitness of reduced transition system $TS_3(L)$, one can postulate the following proposition.

Theorem 1. *Let L be a log and let $TS_3(L)$ be a reduced transition system based on condensed transition system $TS_2(L)$. Then, $TS_3(L)$ perfectly fits L.*

Proof. Let $\sigma = \langle a_1, a_2, ..., a_n \rangle \in A^*$ be a trace of log L and $\sigma = \sigma^+(TS_2(L)) + \sigma^-(TS_2(L))$, where $\sigma^+(TS_2(L))$ is a trace prefix that can be "replayed" on $TS_2(L)$ and $\sigma^-(TS_2(L))$ is a trace suffix that cannot be "replayed" on $TS_2(L)$. $\sigma^+(TS_2(L))$ can be "replayed" on $TS_3(L)$ by the construction. Now we need to prove that the entire sequence σ can be "replayed" on $TS_3(L)$. We will prove that iteratively for sequences $\langle \sigma(1), ..., \sigma(i) \rangle$, where i varies from $|\sigma^+(TS_2(L))|$ to $|\sigma|$. Basis of induction: the proposition is valid for $i = |\sigma^+(TS_2(L))|$, since $\sigma^+(TS_2(L))$ can be "replayed" on $TS_3(L)$. Step of induction: the trace $\langle \sigma(1), ..., \sigma(i) \rangle$ can be "replayed". Now let us prove that the trace $\langle \sigma(1), ..., \sigma(i+1) \rangle$ can be "replayed" as well. According to Algorithms 4 and 5, we "replay" $\langle \sigma(1), ..., \sigma(i) \rangle$ and add a new state $\widehat{s}$ (if it has not been added previously) and a new edge $t = (s, \sigma(i+1), \widehat{s})$ correspondingly. Thus, trace $\langle \sigma(1), ..., \sigma(i+1) \rangle$ now can be replayed, and that proves the step of induction. □

5 Evaluation and Discussion

In this section we evaluate the proposed approach as applied to real-life event logs. In the beginning of the section, an algorithm for precision calculation is introduced.

5.1 Metrics Calculation

Calculation of metrics for all three transition systems is performed throughout their building.

As we have shown above, *fitness* of $TS_1(L)$ and $TS_3(L)$ is perfect. Further, *simplicity* of a model is easily calculated on the basis of the number of model's elements.

We propose an algorithm calculating *precision* metrics for a given transition system $TS(L)$ based on an idea of simulation [16]. The algorithm assumes that $TS(L)$ perfectly fits log L. Suppose that $TS(L)$ can simulate $TS_1(L)$. The behavior which is present in $TS(L)$ but not observed in the log is penalized. Finally, a normalized total penalty forms the basis of a precision value.

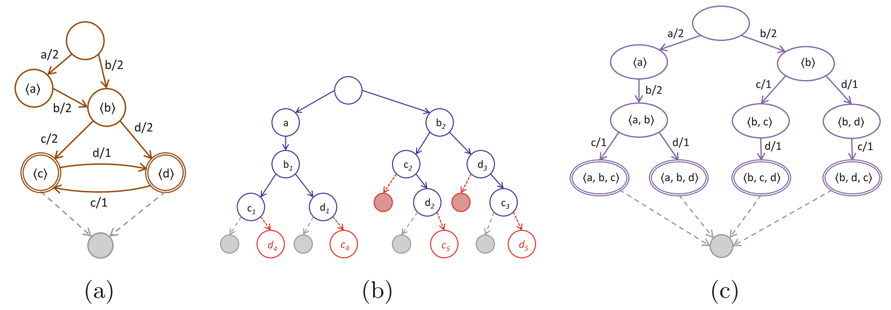

Fig. 4. Transition systems built for log L_2: (a) 1-window size (fixed) $TS_f(L_2)$; (b) unfolding graph obtained after $TS_f(L_2)$ simulated $TS_1(L_2)$; (c) full $TS_1(L_2)$

The approach for calculation of precision is described in Algorithm 6. The algorithm consists of two main steps. First, the algorithm iteratively calculates so-called *partial precisions* for every state in $TS(L)$ (Algorithm 9). Second, the algorithm sums partial precisions (Algorithm 7) and calculates the average over the number of states.

To get an illustration of this idea, consider the following log:

$$L_2 = [\langle a, b, c\rangle, \langle a, b, d\rangle, \langle b, c, d\rangle, \langle b, d, c\rangle]$$

Full transition system $TS_1(L_2)$ for the log is depicted in Fig. 4c. It is considered to be the *most precise reference model.* Transition system $TS_f(L_2)$ in Fig. 4a is built with a fixed window size (equal to 1). As a more general model than $TS_1(L_2)$ it allows more behavior than the reference model, which is *penalized* since it impacts precision.

The calculation of precision invokes a simulation routine implemented as a recursive procedure (Algorithm 8). For the example above, $TS(L) = TS_f(L_2)$ and $TS_1(L) = TS_1(L_2)$. Initially $s = s_0 \in TS(L)$ and $s_1 = s_{01} \in TS_1(l)$ are passed as parameters to the procedure. For each output transition of a current state s, the algorithm tries to find a transition labeled with the same event among output transitions of a current state s_1. Note, that all transition systems considered in the paper are deterministic by construction, i.e. it is not possible for a state to have more than one outgoing transition labeled by the same event.

For example, consider transition $(s_0, a, \langle a\rangle)$ in $TS_f(L_2)$. It has a matching transition $(s_0, a, \langle a\rangle)$ in $TS_1(L_2)$. The procedure is recursively called with parameters $s = \langle a\rangle$ and $s_1 = \langle a\rangle$. This step also produces an input edge to vertex a in an unfolding graph depicted in Fig. 4b. This step has to be repeated for transitions $(\langle a\rangle, b, \langle b\rangle)$ and $(\langle a\rangle, b, \langle a, b\rangle)$ (obtains b_1 in the unfolding graph) and transitions $(\langle b\rangle, c, \langle c\rangle)$ and $(\langle a, b\rangle, c, \langle a, b, c\rangle)$ (obtains c_1 in the unfolding graph). Here, $\langle c\rangle$ in $TS_f(L_2)$ is an accepting state; it contains a virtual output transition (depicted as a gray dashed arrow) to a virtual final state. Similarly, $\langle a, b, c\rangle$ from $TS_1(L_2)$ also has a virtual final transition, which maintains balance (depicted as a dashed output edge in the unfolding graph).

Together with that, $\langle c\rangle$ has transition $(\langle c\rangle, d, \langle d\rangle)$ to state $\langle d\rangle$, which has no counterpart in $TS_1(L_2)$ (edge (c_1, d_4) in the graph). The algorithm penalizes this extra transition and calculates partial precision for state $\langle c\rangle$ as a difference between the number of output transitions and the number of penalized output transitions divided by the total number of output transitions.

Since state $\langle a, b, c\rangle$ of $TS_1(L_2)$ does not allow any further moves, the algorithm leaves the current iteration of the procedure and, thereby, returns to a higher level, to state $\langle b\rangle$ of $TS_f(L_2)$ and $\langle a, b\rangle$ of $TS_1(L_2)$. Then, the algorithm repeats the same steps for all unvisited output transitions.

During its work, the algorithm can normally visit some states of a more general model ($TS_f(L_2)$ in the example) more than once. For each such visit a value of partial precision for the state is recalculated (Algorithm 9).

Finally, after all states of the reference model have been visited during the simulation, values of ultimate partial precision for each state of $TS(L)$ are represented by η. The last step is to calculate an average value (Algorithm 7), which is the required value of the model's precision.

Algorithm 6. Calculating precision for transition system $TS(L)$

```
Input   : general (perfectly fit) transition system
          $TS(L) = (S, E, T, s_0, AS)$ built for log $L$;
          reference full transition system
          $TS_1(L) = (S_1, E_1, T_1, s_{01}, AS_1)$;
Output  : value of precision $Prec(TS(L))$ for $TS(L)$;
Data: partial function $\eta$ mapping each state of $TS(L)$ to a real number
      determining the state's "partial precision"; partial function $\theta$ mapping
      each state $s \in TS(L)$ to a natural number determining how many times
      state $s$ has been visited;

/* Main part of the algorithm */
begin
    CalcStatePrecision ($s_0$, $s_{01}$, $TS(L)$, $TS_1(L)$, $\eta$, $\theta$);
    $Prec(TS(L))$ ← SumPartialPrecisions ($TS(L)$, $\eta$);
```

Algorithm 7. Function SumPartialPrecisions calculating total precision of $TS(L)$

```
Input   : general (perfectly fit) transition system
          $TS(L) = (S, E, T, s_0, AS)$ built for log $L$; function $\eta$ mapping each state
          of $TS(L)$ to a real number determining state's "partial precision";
Output  : value of precision $Prec(TS(L))$ for $TS(L)$;

Function SumPartialPrecisions($TS(L)$, $\eta$): Real
    $sum \leftarrow 0$;
    for $s \in S$ do
        $sum \leftarrow sum + \eta(s)$;
    $res \leftarrow sum/|S|$;
    return $res$;
```

Algorithm 8. Procedure `CalcStatePrecision` calculating "partial precision for entire states" as function η

Input : current state s of $TS(L)$;
current state s_1 of $TS_1(L)$; general (perfectly fit) transition system $TS(L) = (S, E, T, s_0, AS)$ built for log L; reference full transition system $TS_1(L) = (S_1, E_1, T_1, s_{01}, AS_1)$; partial function η mapping each state of $TS(L)$ to a real number determining the state's "partial precision"; partial function θ mapping each state $s \in TS(L)$ to a natural number determining how many times state s has been visited;

```
Procedure CalcStatePrecision($s$, $s_1$, $TS(L)$, $TS_1(L)$, $\eta$, $\theta$)
    $pen \leftarrow 0$;
    for $t = (s, a, s') \in s\bullet$ do
        /* If no matching trans.                                   */
        if $\exists t_1 = (s_1, a, s'_1) \in TS_1(L)$ then
            CalcStatePrecision ($s'$, $s'_1$, $TS(L)$, $TS_1(L)$, $\eta$);
        else
            $pen \leftarrow pen + 1$;
    /* Number of output trans-s                                    */
    $otn \leftarrow |s \bullet|$;
    if $s \in AS$ then
        $otn \leftarrow otn + 1$
        if $s_1 \notin AS_1$ then
            $pen \leftarrow pen + 1$;
    if $otn \neq 0$ then
        $partPartPrec = (otn - pen)/otn$;
        RecalcStatePrecision ($s$, $partPartPrec$);
```

Algorithm 9. Procedure `RecalcStatePrecision` refines the value of a state's "partial precision"

Input : state $s \in S$ of $TS(L) = (S, E, T, s_0, AS)$;
function η mapping each state of $TS(L)$ to a real number determining the state's "partial precision"; partial function θ mapping each state $s \in S$ to a natural number determining how many times state s has been visited; new state's partial precision $pprec$ for refining;

```
Procedure RecalcStatePrecision($s$, $\eta$, $\theta$, $pprec$)
    /* If either $\eta$ or $\theta$ is not defined for this state          */
    if $\eta(s)$ is not defined then
        $\eta(s) \leftarrow 0$;
    if $\theta(s)$ is not defined then
        $\theta(s) \leftarrow 0$;
    $stPrec \leftarrow \eta(s) \cdot \theta(s)$;
    $\theta(s) \leftarrow \theta(s) + 1$;
    $\eta(s) \leftarrow (stPrec + pprec)/\theta(s)$;
```

5.2 Implementation Details

To evaluate the proposed approach, we have developed a number of routines for the ProM toolkit [19]. The routines are implemented as plug-ins with several entry points intended for different sets of input parameters.

The “`Build and reduce transition systems (xi)`” plug-in combines routines for building $TS_1(L)$, $TS_2(L)$, $TS_3(L)$ for a given input log L, along with calculation metrics for each transition system built. In the simplest case, the plug-in obtains as its input only event log L and provides the ability to configure the settings as follows. (1) Specify a maximum window size for building $TS_1(L)$. The default size is unlimited (that is set at a value of -1). By setting the size to a natural number, one can make the algorithm act with a fixed window. We use this option for building reference models with fixed windows. (2) Specify the value of the *Threshold* parameter used for building $TS_2(L)$. (3) Specify the value of the multiplicative factor *Vwsc* used when building $TS_3(L)$.

Once successfully finished, the plug-in produces as its output *three transition systems*, and, what is the most important for analyzing the results, a *hierarchical report*. The report represents a set of characteristics organized in a tree structure. They include metrics for each built transition system and additional attributes calculated during the algorithm's operation. We created a special “view” plug-in for browsing such reports, which allows information to be exported as a JSON-structure or HTML-formatted text.

Both plug-ins are openly available to download on http://pais.hse.ru.

5.3 Experiments and Discussion

We evaluated our approach on a set of event logs, both artificial and real-life. In the following sections we consider “BPI challenge” logs [1] prepared for a Disco project [2] `L3` (11 traces, 89 activities) and `L4` (251 traces, 247 activities). Full transition systems built for the logs have the following frequency characteristics:

- $TS_1(L3)$: Maximum Window Size: 71; 514 states; 513 transitions;
- $TS_1(L4)$: Maximum Window Size: 83; 8088 states; 8087 transitions.

Our goal is to compare metrics of models built with an existing fixed window algorithm and models built with the algorithms proposed in this paper.

Table 1 shows metrics of condensed and reduced models constructed with unlimited window size. Metrics of k-window models for the logs are presented in Table 2. Comparing the results from both tables, one can consider the following outcome. By using a *fixed window* approach based on [3] the maximum reduction is reached for a parameter $k = 1$. This is the smallest model that can be built by any algorithm using *h-tail* approach [18]. Moreover, Table 2 shows that increasing window size (k) to a value greater than 7 does not significantly impact simplicity and precision.

In our algorithms, there are two parameters affecting model size: *Threshold* and *Vwsc*. Parameter *Threshold* has a limited impact on the model size. As it is shown in Table 1, dependence of the model size from *Threshold* is nonlinear in the entire domain of *Threshold*. For log $L4$, better simplicity results are in the vicinity of the points 0.2 and 0.8 and worse in the range ends. This is because in the case of negligibly small *Threshold* values, $TS_2(L)$ model is similar to $TS_1(L)$; so, the application area of a variable size window lies near the tree leaves. In this situation, a noticeable reduction of a model is achievable only for traces with similar suffixes.

Table 1. Dependence of model's metrics on parameters *Threshold* and *Vwsc* with an unlimited window size

Settings		$TS_2(L3)$: Condensed				$TS_3(L3)$: Reduced				$TS_2(L4)$: Condensed				$TS_3(L4)$: Reduced			
Thres-hold	VWSC	TThr	States #	Transs #	Act #	States #	Transs #	Simpl	Prec	TThr	States #	Transs #	Act #	States #	Transs #	Simpl	Prec
0	1	1	514	513	89	514	513	0,0876	1	0	8088	8087	247	8088	8087	0,0153	1
0,05	1	1	514	513	89	514	513	0,0876	1	13	37	36	20	320	1573	0,131	0,567
0,1	1	1	514	513	89	514	513	0,0876	1	25	12	11	10	327	1595	0,129	0,5482
0,25	1	3	11	10	9	470	478	0,0949	0,9911	63	9	8	8	320	1531	0,134	0,5585
0,33	1	4	9	8	8	464	474	0,0959	0,9905	83	9	8	8	320	1531	0,134	0,5585
0,5	1	6	7	6	6	470	478	0,0949	0,9911	126	6	5	5	308	1540	0,1342	0,5523
0,65	1	7	6	6	6	470	478	0,0949	0,9911	163	3	2	2	289	1490	0,1394	0,5595
0,75	1	8	7	6	6	470	478	0,0949	0,9911	188	3	2	2	289	1490	0,1394	0,5595
0,85	1	9	7	6	6	470	478	0,0949	0,9911	213	3	2	2	289	1490	0,1394	0,5595
0,95	1	10	7	6	6	470	478	0,0949	0,9911	238	2	1	1	308	1544	0,1339	0,5581
1	1	11	1	0	0	470	478	0,0949	0,9911	251	2	1	1	308	1544	0,1339	0,5581
0,25	0,5	3	11	10	9	397	438	0,1078	0,9505	63	9	8	8	173	992	0,2129	0,5514
0,33	0,5	4	9	8	8	395	438	0,108	0,9477	83	9	8	8	173	992	0,2129	0,5514
0,5	0,5	6	7	6	6	397	438	0,1078	0,9505	126	6	5	5	147	869	0,2441	0,556
0,75	0,5	8	7	6	6	397	438	0,1078	0,9505	188	3	2	2	141	866	0,2463	0,5809
0,25	0,25	3	11	10	9	312	403	0,1259	0,8764	63	9	8	8	73	645	0,3454	0,5116
0,33	0,25	4	9	8	8	311	405	0,1257	0,8739	83	9	8	8	73	645	0,3454	0,5116
0,5	0,25	6	7	6	6	309	400	0,1269	0,8771	126	6	5	5	78	639	0,3459	0,5387
0,75	0,25	8	7	6	6	309	400	0,1269	0,8771	188	3	2	2	86	709	0,3119	0,4843
0,33	**0,12**	4	9	8	8	139	320	**0,1961**	**0,7012**	83	9	8	8	59	565	**0,3974**	**0,5333**
0,33	**0,05**	4	9	8	8	55	201	**0,3516**	**0,676**	83	9	8	8	26	404	**0,5767**	**0,5266**

Table 2. Metrics of k-window models

k (wnd size)	L3				L4			
	States #	Transs #	Simpl	Prec	States #	Transs #	Simpl	Prec
1	90	273	0,2479	0,6117	248	1755	0,124	0,3791
2	274	371	0,1395	0,8574	1756	3602	0,046	0,7607
3	372	418	0,1139	0,9457	3603	4838	0,029	0,8892
4	419	437	0,1051	0,9828	4839	5600	0,024	0,946
5	438	451	0,1012	0,9882	5601	6129	0,021	0,9671
7	464	474	0,0959	0,9914	6515	6806	0,019	0,9836
10	492	497	0,091	0,9961	7228	7392	0,017	0,9929
15	508	508	0,0886	0,999	7840	7905	0,016	0,9968
20	513	513	0,0877	0,999	8037	8054	0,015	0,9991

When *Threshold* values come closer to 1, a smaller $TS_2(L)$ model is built. Consequently, at the stage of building $TS_3(L)$ model, most of it is to be reconstructed. Preliminary experiments show that selecting *Threshold* from a range $[0.2; 0.8]$ leads to better results; nevertheless, further elaboration of the parameter's impact is needed. We consider $\mathit{Threshold} = 0.33$ as a reference value. For such a value, only the states and transitions with frequencies that equal at least 1/3 of the total number of traces are preserved in the condensed transition system. Hence, all other states are reconstructed in a more abstract manner. The results for $\mathit{Threshold} = 0.33$ are highlighted in blue in Table 1.

By decreasing the value of *Vwsc* parameter from 1 to 0, *significant reduction of a resulting model size* is achieved (last two rows of Table 1). For example, for log $L3$ and a value of $\mathit{Vwsc} = 0.05$ we have $\mathit{Simpl}(TS_3(L3)) = 0.3516$ and $\mathit{Prec}(TS_3(L3)) = 0.676$ versus $\mathit{Simpl}(TS_f(L3)) = 0.2479$ and $\mathit{Prec}(TS_f(L3)) = 0.6117$ for the case of one window size model. Generally, by adjusting the value of *Vwsc*, one can obtain a resulting model in a wide range of sizes. Unlike parameter *Threshold*, parameter *Vwsc* gives a linear dependence of the model size. Moreover, by varying both values *Threshold* and *Vwsc* it is possible to enhance the mutual influence of the parameters on each other. That allows flexible balancing between precision and simplicity.

Comparision with existing approaches to transition systems reduction. By comparing the proposed approach with existing approaches we can make the following conclusions. Most of the existing methods are based on different approaches used to determine a *state representation function* [4].

In this paper, we proposed an essentially new *adaptive* state representation function, which was not considered before. While inferring a transition system from an event log, the function takes into account the frequency of an individual event and varies the size of the window that determines the current state. Generally, the use of a *fixed window* allows improving *simplicity* of a model while the use of an unlimited window improves its *precision*. Our approach combines both and allows balancing between these two metrics in a flexible manner.

Intuitively, the bigger the number of events in a log corresponding to a state is, a more specific (and, correspondly, less abstract) the state is. The more specific state is marked by a longer subtrace, which reduces the probability of merging the state with another state. In contrast, a big number of states corresponding to rare behavior patterns can be easily merged, which dramatically impacts the size of the transition system (and, correspondly, increases its *simplicity*). The *precision* still slightly suffers.

The proposed approach implies an important corollary: the method is essentially insensitive to noise. This is a big advantage of the method over some existing methods, e.g. [18], where a comprehensive preprocessing of input logs is needed.

The experiments on both artificial and randomly chosen real-life logs supported the proposed approach.

6 Conclusion

This paper presented a new approach for reducing transition systems, based on an inference algorithm with a varied window size. In contrast to the existing approaches, the approach presented in this paper shows the advantage of allowing flexible adjustments of the size of the resulting model. The evaluation of a model's quality is made by measuring its metrics. For calculation of the *precision* metric, an original algorithm has been developed. This way, estimation of achievability of the main goal is made on the basis of numerical characteristics of resulting models. The experiments with artificial and real-life logs justified the proposed approach.

Future work is aimed at further investigation of impacts of various algorithm coefficients on time costs of the region-based algorithm applied to resulting transition systems obtained during more comprehensive experiments. Moreover, we plan to investigate the quality metrics of Petri nets synthesized from transition systems obtained as an outcome of the algorithm.

References

1. http://www.win.tue.nl/bpi/doku.php?id=2015:challenge&redirect=1id=2015/challenge
2. http://fluxicon.com/blog/2015/05/bpi-challenge-2015/
3. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software Syst. Model. **9**(1), 87–111 (2008). http://dx.doi.org/10.1007/s10270-008-0106-z
4. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19345-3
5. Angluin, D.: Inference of reversible languages. J. ACM **29**(3), 741–765 (1982). http://doi.acm.org/10.1145/322326.322334
6. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Nielsen, M., Schwartzbach, M.I. (eds.) CAAP 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995). doi:10.1007/3-540-59293-8_207
7. Badouel, E., Bernardinello, L., Darondeau, P.: The synthesis problem for elementary net systems is NP-complete. Theoret. Comput. Sci. **186**, 107–134 (1997)
8. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998). doi:10.1007/3-540-65306-6_22
9. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75183-0_27
10. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. **21**(6), 592–597 (1972)

11. Buijs, J.C.A.M., Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) OTM 2012. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33606-5_19
12. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85758-7_26
13. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol. **7**(3), 215–249 (1998). http://doi.acm.org/10.1145/287000.287001
14. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. IEEE Trans. Comput. **47**(8), 859–882 (1998). http://dx.doi.org/10.1109/12.707587
15. Lorenzoli, D., Mariani, L., Pezzè, M.: Inferring state-based behavior models. In: 4th International Workshop on Dynamic Analysis (WODA 2006) co-located with the 28th International Conference on Software Engineering (ICSE 2006), pp. 25–32. ACM Press (2006)
16. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981). doi:10.1007/BFb0017309
17. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13675-7_14
18. Sole, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1), 192–205 (2013)
19. Verbeek, H., Buijs, J., Dongen, B., Aalst, W.: ProM 6: the process mining toolkit. In: Rosa, M.L. (ed.) Proceeding of BPM Demonstration Track 2010, CEUR Workshop Proceedings, vol. 615, pp. 34–39 (2010)
20. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68746-7_24