2020 17th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)
Mexico City, Mexico. November 11-13, 2020

Automated Modelling of Deadlock-free Petri Nets Using Duplicated Transition Labels

Román Pomares-Angelino, Ernesto López-Mellado
CINVESTAV Unidad Guadalajara
45019 Zapopan, Jal. Mexico
{jrpomares, elopez}@gdl.cinvestav.mx

***Abstract*— The paper addresses the problem of automated synthesis of deadlock free Petri nets (PN) from event sequences. A method for determining substructures that yield deadlocks is presented. Based on structural patterns that may appear during the synthesis of PN from some types of event sequences, deadlock substructures, called inconsistent, are detected. Afterwards, a repairing technique is applied to correct every inconsistent substructure by adding a new transition, which is associated to an event symbol already assigned to a transition in the inconsistent substructure. The algorithms derived from the technique, which have polynomial-time complexity, have been implemented and tested on examples of diverse structures**

Keywords—Petri net discovery, Model repair, Structural patterns, duplicated transitions labels.

I. Introduction

Automated modelling of discrete event processes from logs of observed behaviour is currently a problem addressed by several research groups in the areas of discrete manufacturing systems [1] and workflow process management [2]. In each field, this problem is called identification and discovery respectively**.**

Most of the proposed identification/discovery methods aim to obtain Petri net (PN) models, which are well suited to represent complex discrete-event processes when discovery is used as a resource for reverse engineering [3], [4], [5]. These methods often hold in the problem statement the constraint that the labelling of transitions in the PN is defined as bijective function on the events (or tasks) alphabet. Consequently, some traces of the events log cannot be represented in the model; thus, some transitions labelled with ε are used; they are called silent, invisible, or unobservable transitions.

For example, the log λ={ABC, AC} can be represented by the PN shown in Figure 1, in which the silent transition labelled with ε allows firing the sequence AC.

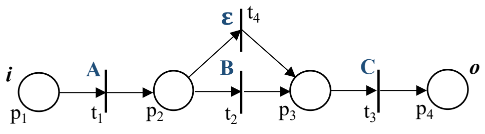


Fig. 1. PN with an ε-transition

In the last few years, several discovery methods have been proposed for relaxing this constraint allowing obtaining workflow nets (WFN) including silent transitions [6][7]; the methods called alpha# and alpha$ respectively allow determining successfully silent transitions of type *initialize*, *finalize*, *skip*, *redo*, and *switch*.

Recently, other simplest method based on a pre-processing of the set of traces in the event log λ has been proposed [8]; λ is classified into three classes: a) the normal traces, which lead to a basic WFN that may include *initialize* and *finalize* silent transitions, and b) the abnormal short traces, and the abnormal long traces, which are used to refine the basic model by adding *skip* and *redo* silent transitions respectively. Later, in [9] an event clustering method that finds switch transitions is presented.

In this paper, an alternative approach to construct PN with ε-transitions is proposed. The discovered PN may have several transitions using the same event symbol. Figure 2 shows an equivalent PN to represent the log λ={ABC, AC}. Furthermore, the use of replicated event symbols is optimised.

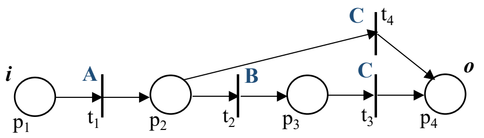


Fig. 2. PN using the same label on two transitions.

In this study, the subclass of PN called workflow nets (WFN) is dealt, but the proposed repairing method can be applied for the synthesis of ordinary PN.

The paper is organised as follows. Section 2 presents the notation on Petri nets and the problem formulation. Section 3 describes the pattern based technique for determining deadlock sub-structures. Section 4 presents the technique for rewriting such structures as deadlock-free ones with additional transitions using repeated event symbols. Section 5 outlines the developed software tool and presents tests on several case studies.

II. Background, problem and approach

A. Petri nets basics [2]

Definition 1. An ordinary Petri Net structure G is a bipartite digraph represented by the 3-tuple $G = (P, T, \mathcal{F})$; where: $P = \{p_1, p_2, ..., p_{|P|}\}$ and $T = \{t_1, t_2, ..., t_{|T|}\}$ are finite sets of nodes named places and transitions respectively; $\mathcal{F} \subseteq P\times T \cup T\times P$ is a relation representing the arcs between the nodes.

For any node $x \in P \cup T$, $^{\bullet}x = \{y|(y, x) \in \mathcal{F}\}$ and $x^{\bullet} = \{y|(x, y) \in \mathcal{F}\}$.The incidence matrix of G is $C = [c_{ij}]$; where $c_{ij}= -1$ if $(p_i, t_j) \in \mathcal{F}$ and $(t_j, p_i) \notin \mathcal{F}$; $c_{ij}= 1$ if $(t_j, p_i) \in \mathcal{F}$ *and* $(p_i, t_j) \notin \mathcal{F}$; $c_{ij}= 0$ otherwise.

The places in P can be empty or marked by one or more *tokens*. A *marking* $M: P \rightarrow \mathbb{N}$ determines the number of tokens within the places; where $\mathbb{N}$ is the set of non-negative integers. A marking M, usually denoted by a vector $(\mathbb{N})^{|P|}$, describes the current state of the modelled system.

Definition 2. A Petri Net system or Petri Net (PN) is the pair $N = (G,M_0)$, where G is a PN structure and M_0 is an initial marking. $R(G,M_0)$ denotes the set of all reachable markings from M_0.

Definition 3. A PN system is *1-bounded* or *safe* iff, for any $M_i \in R(G,M_0)$ and any $p \in P$, $M_i(p) \leq 1$. A PN system is *live* iff, for every reachable marking $M_i \in R(G,M_0)$ and $t \in T$ there is a $M_k \in R(G,M_i)$ such that t is enabled in M_k.

Definition 4. A *t-invariant Y_i of a PN* is a non-negative integer solution to the equation $CY_i=0$. The *support* of Y_i denoted as $<Y_i>$ is the set of transitions whose corresponding elements in Y_i are positive. Y is *minimal* if its support is not included in the support of other t-invariant. A *t-component* $G(Y_i)$ is a subnet of *PN* induced by a $<Y_i>$: $G(Y_i)=(P_i, T_i, F_i)$, where $P_i = {}^{\bullet}<Y_i> \cup <Y_i>^{\bullet}$, $T_i = <Y_i>$, $F_i= (P_i \times T_i \cup P_i \times T_i) \cap F$.

In a t-invariant Y_i, if we have initial marking (M_0) that enables a $t_i \in <Y_i>$, when t_i is fired, then M_0 can be reached again by firing only transitions in $<Y_i>$.

Definition 4. A *WorkFlow net* (WFN) N is a subclass of *PN* owning the following properties (van der Aalst, 2004): (i) it has two special places: i and o. Place i is a source place: $^{\bullet}i = \varnothing$, and place o is a sink place: $o^{\bullet} = \varnothing$. (ii) If a transition t_e is added to *PN* connecting place o to i, then the resulting *PN* (called *extended* WFN) is strongly connected.

Definition 5. A WFN (N, M_0) is said to be *sound* iff any marking $M_i \in R(N, M_0)$, $o \in M_i \rightarrow M_i = [o]$ and $[o] \in R(N, M_i)$ and (N, M_0) contains no dead transitions. An *extended* WFN sound is live and bounded. A WFN can represent a process behaviour by associating task labels to some transitions.

Definition 6. A labelled WFN is a four-tuple (N, M_0, Σ, L) where Σ is a finite set of event labels, and $L:T \rightarrow \Sigma \cup \{\varepsilon\}$ is the labelling function. Transitions labelled with ε are called *silent* or *unobservable*, otherwise they are called *observable*. Additionally, $\forall t_i, t_j \in T, t_i \neq t_j$, if $L(t_i),L(t_j) \in \Sigma$ then $L(t_i) \neq L(t_j)$; i.e., two transitions cannot have the same label from Σ.

B. *Problem statement*

Definition 7. Let Σ be a finite set of tasks labels; an event log λ is a set of traces $\sigma_i = A_1A_2 \ldots A_k \in \Sigma^*$, $|\sigma_i| = k$, $A_j \in \Sigma$; $1 \leq j \leq k$ refers to the task at position j.

In a workflow net, the first and the last tasks in every trace correspond to transitions in $i^{\bullet}$, $^{\bullet}o$ respectively.

Definition 8. Given a workflow log $\lambda = \{\sigma_1, \sigma_2, \ldots \sigma_j\}$ generated by a sound WFN, the discovery problem consist of synthesising a safe PN structure using observable transitions in which the event symbol ε is not used and the number of repeated symbols in Σ is minimum; that is, the constraints of Definition 7 are relaxed. The number of places is unknown.

Assumptions. Considering that C is generated by a WFN N_1 with ε-transitions, the discovered model N_2 will be equivalent to N_1 by holding the following assumptions:

A1. The process is well behaved, i.e. there are no deadlocks nor overflows during the observation period.

A2. All the traces that do not fire ε-transitions are included in the event log.

A3. Silent transitions of types *skip* and *redo* do not appear in the same t-component (of the extended WFN).

C. *Overview of the proposal*

The strategy of the method is repairing an incorrect model obtained by a discovered technique that does not deal with the behaviour of PN including ε-transitions. For example, the method CoMiner [10] obtains from λ={ABC, AC} the following set of causal dependencies {[-, A], [A, B], [A, C], [B, C], [C, -]} that lead to the PN shown in Figure 3, which cannot replay the log λ. Notice that the causal dependency [A, C] is built through the places p_2 or p_3; this provokes a deadlock since any of the traces in λ cannot fire.

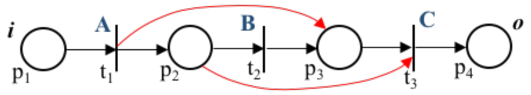


Fig. 3. PN obtained by a method not able to discover ε-transitions.

The proposed method locates these substructures in an incorrect model built from behaviours issued from silent transitions Skip, Redo, and Switch implicit in the traces of λ. Afterwards, the inconsistent substructures are transformed into an equivalent one using additional transitions labelled with an event symbol used in the wrong structure.

III. Detecting Inconsistent Substructures

A. *The method CoMiner*

Any method that does not deal with event logs involving silent events will construct wrong PN structures when the sequences follow a behaviour corresponding to silent events. We outline a method that may build WFN from non-complete (thus reduced) event logs [10].

The method processes the event log λ for determining conjoint occurrent classes, which are sets of events that always occur in a same set of sequences; this allows inferring some concurrent relations and causal relations between event symbols not observed explicitly in λ. [A, B] expresses that A and B are in a causal relation, whilst C||D states that C and D are concurrent.

Both computed and inferred dependencies determine net structures that can be composed by transition fusion. Sequential compositions build paths; if there are two dependencies [A, B] and [B, C], it can be built a path of three transitions A, B, and C. The causal dependencies [A, B] and [A, E] lead to a composed dependency [A, B||E] or [A, B+E] if B and E are concurrent or not respectively. Figure 4 summarises the composition operations. Successive compositions lead to a connected WFN in which every transition has associated a unique event symbol used in λ.

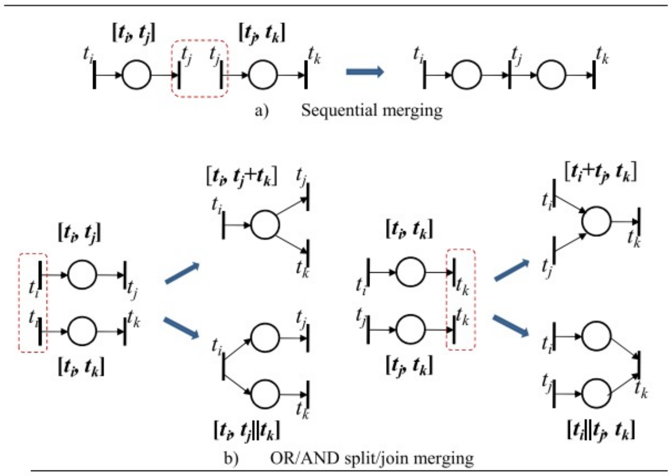


Fig. 4. Composition of dependencies [10].

B. Inconsistent structures

The set of causal dependencies {[-, A], [A, B], [A, C], [B, C], [C, -]} can be composed yielding the set of causal dependencies {[-, A], [A, B+C], [A+B, C], [C, -]}. However, the composition will construct the model shown in Figure 2 which expresses that B and C are concurrent by creating a dependency [A, B||C] through p_2 and p_3; similarly, the dependency [A||B, C] is created. These structures contradict the composed dependencies, and then the PN does not represent the behaviour of λ.

Notice that the composed dependencies that build such inconsistent relations are OR dependencies; [A, B+C] is named Or-split and [A+B, C] is named Or-join; the set of all these dependencies is called compOr.

Definition 9. Let r_i, r_o ∈ compOr be two composed Or dependencies, where r_i is an Or-split, and r_o is an Or-join. Then the pair (r_i, r_o) is an inconsistent structure (IS) iff these dependencies lead to a substructure that has an And-split or an And-join relation.

Let us analyse the relationship between IS and the behaviour due to the silent transitions.

1) Skip transition

A structure derived from traces that involve a skipping behaviour, such as {ABCDE, AE} is shown in Figure 5. The causal dependencies are {[-, A], [A, B], [B, C], [C, D], [D, E], [A, E]}, the composed Or dependencies are r_1=[A, B+E], r_2=[A+D, E], and a path B,C,D.

Fig. 5.a shows separately the components of the dependencies, and Fig. 5.b shows the WFN formed by the assembled substructures, which do not represent the traces of the log. The pair (r_1, r_2) is an IS, where r_1 is an Or-split dependency and r_2 is an Or-join dependency.

2) Redo transition

The log {ABCD, ABCBCD} represents a Redo behaviour; it includes a trace that involves an iteration. The causal dependencies are {[-, A], [A, B], [B, C], [C, D], [D, -], [C, B]}, the composed Or dependencies are r_1=[A+C, B], r_2=[C, B+D].

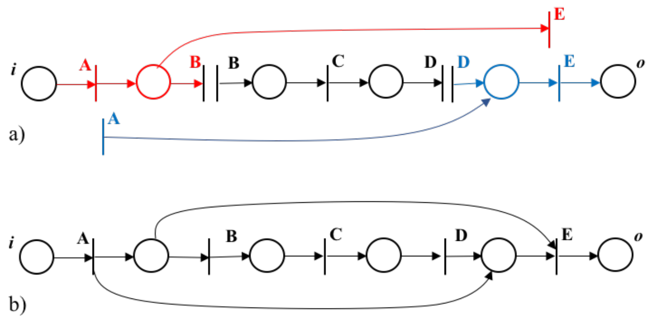


Fig. 5. PN built from a log involving a skip behaviour

Figure 6.a shows the substructures of the dependencies and Fig. 6.b the assembled WFN, which cannot reproduce the log because transition labelled with B is deadlocked. The pair (r_2, r_1) is an IS, where r_2 is an Or-split dependency and r_1 is an Or-join dependency.

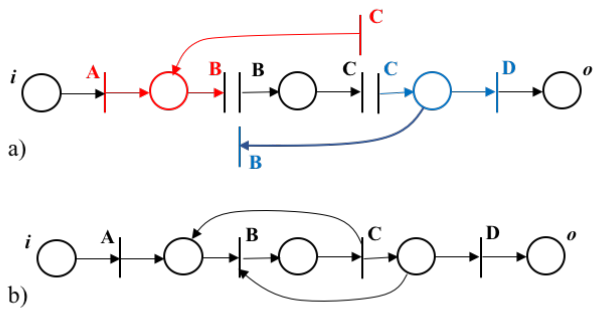


Fig. 6. PN built from a log including a Redo behaviour

3) Switch transition

The log {ABCD, AEFD, ABFD} includes a trace that involves a switch transition in a WFN. The causal dependencies are {[-, A], [A, B], [B, C], [C, D], [D, -], [A, E], [E, F], [F, D], [B, F]}, the comp-Or dependencies are r_1=[A, B+E], r_2=[C+F, D] r_3=[B, C+F], r_4=[B+E, F].

Figure 7.a shows the composed dependencies and Figure 7.b the assembled WFN, which is not sound. It cannot fire the trace AEFD, and the trace ABCD let a token behind when it is fired. The pair (r_3, r_4) is an IS, where r_3 and r_4 are Or-split and Or-join dependencies respectively.

4) Patterns of inconsistent structures

The previous analysis shows that the behaviours of a WFN built using IS are not correct. In the three cases, the form of the Or dependencies is similar and follows the same pattern stated below.

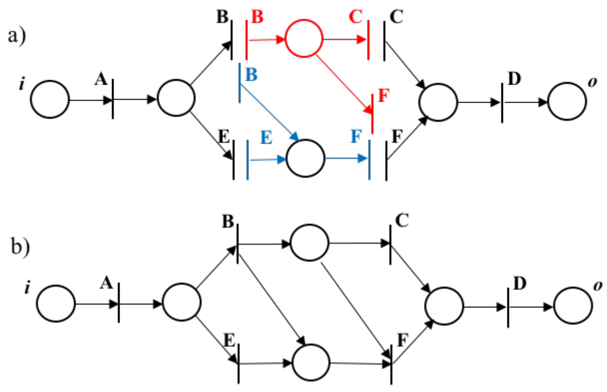


Fig. 7. PN built from a log including a Switch behaviour

Proposition 1. Let r_i, $r_o \in$ compOr be two composed Or dependencies. The pair (r_i, r_o) is an inconsistent structure iff these dependencies have the following form: r_i=[a, $b + d$+..+r], r_o= [$a + c$+…+k, d], where $a, b, c, d, r, k \in \Sigma$.

Proof. (←) r_i states a causal relation between a and d though a place p_i, whilst r_o states a causal relation between c (and some other transitions) and d through another place p_j. This constructs an And-join structure between a and c with d, which leads to a deadlock situation. (→) Conversely, by Definition 9, an IS is characterized by the structures in dependencies r_i and r_o. ■

C. Determining inconsistent structures

The previous pattern is useful to detect IS in a PN discovered by a method that does not deal with silent transitions. A matching test can determine the set of IS from the comp-Or substructures built by such a method.

In order to verify if a pair of dependencies (r_i, r_o) match the pattern stated in Proposition 1, one must fulfil the following conditions. For a dependency r =[S, T]∈R, the operator $^\bullet$ selects the parts of the dependency: $^\bullet r$=S (left) and $r^\bullet$=T (right).

Proposition 2. Let r_i =[W, X], r_o=[Y, Z] be two comp-Or dependencies. (r_i, r_o) is an IS if a) $^\bullet r_i \cap {}^\bullet r_o \neq \varnothing$ and b) $r_i^\bullet \cap r_o^\bullet \neq \varnothing$.

Proof. Condition a) implies that the single transition in W is also included in Y. Similarly, b) implies that the single transition in Z is included in X. This fulfils the matching of the pattern stated in Proposition 1. ■

Consider R the set of composed dependencies computed by a method that does not deal with silent transitions. We need to analyse the compOr dependencies to determine if some pair of dependencies fulfil the conditions in the pattern. First, the Or-dependencies are selected; the conditions of Proposition 2 are tested.

The procedure is summarised in Algorithm 1. The type of relations in S or T is obtained by the function *type-r*: *type-r*(S)∈{Or, And}.

Algorithm 1. *Ident-IS*: gets the set of inconsistent structures

```
Input: R            // composed dependencies
Output: Cei         // set of IS
1: $Cei \leftarrow \varnothing$; compOr ← ∅;
2: $\forall d \in R$
3: If (type-r($^\bullet$d) = Or) or (type-r(d$^\bullet$) = Or)
4:     then $compOr \leftarrow compOr \cup \{d\}$
5: $\forall r_i \in compOr$
6:     $\forall r_o \in compOr$
7:         If $^\bullet r_i \cap {}^\bullet r_o \neq \varnothing$ and $r_i^\bullet \cap r_o^\bullet \neq \varnothing$
8:         then $Cei \leftarrow Cei \cup \{(r_i, r_o)\}$
9: Return Cei
```

IV. Rewriting Inconsistent Structures

A. Strategy

Inconsistent structures create concurrent dependencies through transitions that become and-split or and-join as described above. The main idea is to duplicate such transitions and use the same event symbol.

This situation is clearly exhibited in Figure 5. In the PN of Fig 5.b, A is an And-split transition and E is an And-join transition; this is the outcome of merging the transitions in the IS ([A, B+E], [A+D, E]). We can see that the transitions that have to be duplicated are A and E, as suggested by Figure 5.a, then an arc to link the place $^\bullet$A to the duplicated transition A'. Similarly, transition E' is linked to the place E$^\bullet$. This is shown in Figure 8, where A'(E') is the same symbol than A (E). This PN has not deadlocks and represents the log {ABCDE, AE}; the trace AE is fired in two manners using the duplicated transitions, thus, one of them can be discarded, for example, the new transition A that involves non-determinism.

PN including Redo and Switch behaviours using duplicated transitions are shown in Figures 9 and 10 respectively.

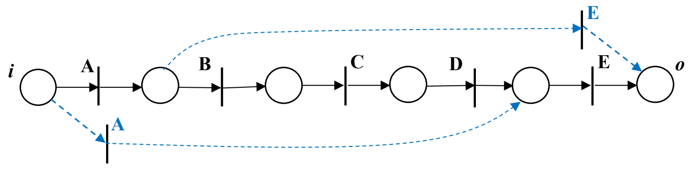


Fig. 8. PN with Skip behaviour using duplicated transitions

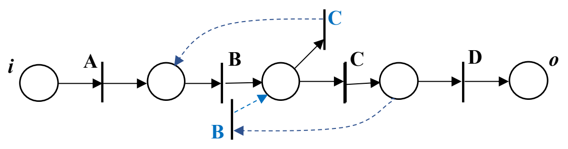


Fig. 9. PN with Redo behaviour using duplicated transitions

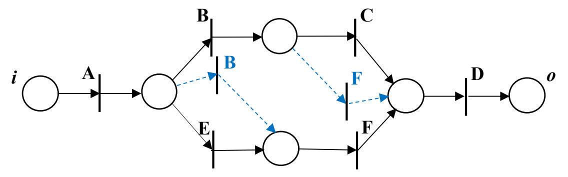


Fig. 10. PN with Switch behaviour using duplicated transitions

B. *Locating transitions to duplicate*

In the previous examples, which illustrate the case of Skip, Redo, and Switch behaviours, the transitions that need to be duplicated in an IS (r_i, r_o) are included simultaneously in two parts of each of the composed Or dependencies r_i and r_o. They can be determined using the following condition.

Proposition 3. Let (r_i, r_o) be a IS. The transitions that must be duplicated to avoid the IS are ${}^\bullet r_i \cap {}^\bullet r_o$ and $r_i^\bullet \cap r_o^\bullet$.

Proof. The transitions in ${}^\bullet r_i \cap {}^\bullet r_o$ and $r_i^\bullet \cap r_o^\bullet$ are precisely those that make fulfil the conditions of Proposition 2 to be an IS; besides, as one of the parts is a singleton, the transitions are uniquely determined. ■

C. *Rewriting IS*

Let us analyse the strategy to rewrite each substructure according to the behaviour of the silent transition.

1) *Skip*

In the example of Figure 8 the transitions in ${}^\bullet r_i \cap {}^\bullet r_o = \{t_i\}$, where L($t_i$)=A, and the transitions in $r_i^\bullet \cap r_o^\bullet = \{t_j\}$, where L($t_j$)=E have to be duplicated. The rewriting is performed as follows:

- In the case of t_i, a new transition t_k is created, with L(t_k) = A; then, the flow is updated by deleting in $\mathcal{F}$, the arc from t_i to ${}^\bullet t_j$, and adding the new arcs (${}^\bullet t_i$, t_k) and (t_k, ${}^\bullet t_j$) to $\mathcal{F}$.
- In the case of t_j, a new transition t_r is created, with L(t_r) = E; then, the flow is updated by deleting the arc from $t_i^\bullet$ to t_j, and adding the new arcs ($t_i^\bullet$, t_r) and (t_r, $t_j^\bullet$) to $\mathcal{F}$.

2) *Redo*

In the example of Figure 9 the transitions in ${}^\bullet r_i \cap {}^\bullet r_o = \{t_i\}$, where L($t_i$)=C and the transitions in $r_i^\bullet \cap r_o^\bullet = \{t_j\}$, where L($t_j$)=B have to be duplicated. The rewriting is performed similarly as defined for the skip behaviour with L(t_k) = C and L(t_r) = B.

3) *Switch*

In the example of Figure 10 the transitions in ${}^\bullet r_i \cap {}^\bullet r_o = \{t_i\}$, where L($t_i$)=B, and the transitions in $r_i^\bullet \cap r_o^\bullet = \{t_j\}$, where L($t_j$)=F must be duplicated. The rewriting is performed similarly as defined for the skip behaviour with L(t_k) = B and L(t_r) = F.

4) *Minimising duplicated transitions*

The previous strategy adds two new transitions t_k, t_r for every inconsistent structure in the discovered model before rewriting. Following this strategy, the repaired WFN is sound and represents correctly all the traces in the log.

However, it contains redundant paths regarding the traces related to silent transitions. Then, for simplifying the repaired PN, t_k can be discarded since it has the same label of t_i and both are output transitions from the same place; hence, they represent a non-deterministic choice.

5) *Rewriting procedure*

Now we can apply the test stated by Proposition 3 to correct all the IS of a model using the previous rewriting and simplification steps; then, we will obtain a sound WFN. The rewriting procedure is summarised in Algorithm 2.

Algorithm 2. *Repair-IS*

Input: *Cei*, $\mathcal{F}$

Output: $\mathcal{F}$ (repaired)

```
1: If $Cei = \varnothing$ then Return $\mathcal{F}$
2: $\forall (r_i, r_o) \in Cei$
3:     $t_i \leftarrow$ Item(${}^\bullet r_i \cap {}^\bullet r_o$); \\ obtains the object in the set $\{t_i\}$
4:     $t_j \leftarrow$ Item($r_i^\bullet \cap r_o^\bullet$);
5:     $T \leftarrow T \cup \{t_r\}$;
6:     $\mathcal{F} \leftarrow \mathcal{F} \setminus \{(t_i, {}^\bullet t_j), (t_i^\bullet, t_j)\}$;
7:     $\mathcal{F} \leftarrow \mathcal{F} \cup \{(t_i^\bullet, t_r), (t_r, t_j^\bullet)\}$;
8:     $L(t_r) \leftarrow L(t_j)$;
9: Return $\mathcal{F}$
```

V. Implementation and Tests

The algorithms derived from this method and auxiliary procedures have been implemented using Java 1.8.0 on the IDE of Netbeans 8.2. The software module is integrated with another module implementing the CoMiner and a PN editor.

The test scheme is shown in Figure 11. First, a PN is edited using PIPE [11] and the model is executed to produce an artificial event log; this allows validating in a controlled manner, that the discovered models are correct; the artificial log is the input to a module that implements CoMiner, which discovers a possible incorrect PN. The structure of such a PN, in particular, the composed dependencies R are processed by the rewriting module. Finally, the repaired PN is displayed using the Graphviz library [12].

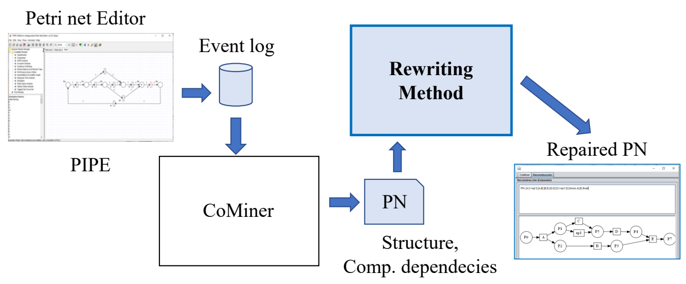


Fig 11. Implementation scheme

Using this test scheme, the developed software has applied to discover WFN from artificial event logs obtained from known models with silent transitions, which involve diverse structural complexity. We include below two simple case studies.

Test 1. The processed log is λ_2={xabcdefgy, xabcefgy, xabcdefbcdefgy, xABCDy, xabCDy}. The model discovered using CoMiner is shown in Figure 12, and the repaired model is shown in Figure 13, where the transitions using labels *b*, *C*, and *e* are duplicated, their labels are *b'*, *C'*, and *e'* respectively.

In the repaired PN, *b'* is a redo transition, *C'* is a switch transition, and *e'* is used to skip transition *d*. The processing time of the repairing algorithm in this test is 0.8162 ms.

Test 2. The processed log is λ_2={xjlmoknpqr, xjlknpqmor, xjlmknpqor, xjknpqlmor, xjklmonpqr, xjknlmopqr, xjknplmoqr, xjloknpqr, xjknpqknpqlmor, xjknpqnpqlmor, xabcef, xabdef, xaghsif, xabsif, xaf, xif}.

The model discovered using CoMiner is shown in Figure 14, and the repaired model is shown in Figure 15; in this PN there are six transitions using labels *i*, *f*, *k*, *n*, *o*, and *s*, which are duplicated.

The corresponding duplicated transitions *i'*, *f'*, and *o'* are skip transitions, *k'* and *n'* are redo transitions, and *s'* is a switch transition. The processing time of the repairing algorithm in this test is 2.1573 ms.

CONCLUSIONS

We have presented a method for discovering sound workflow nets from event logs involving behaviour due to silent transitions. The models may have two or more transitions labelled with the same event symbol. The approach held is repairing an incorrect model obtained by a method that does not deal with silent transitions. The strategy is based on the use of structural patterns that determine the inconsistent substructures in the model that provoke deadlocks. Then, such substructures are rewritten using additional transitions labelled with an event symbol already used in the inconsistent substructures.

The repaired model is sound and contains the minimum number of repeated labels. The polynomial-time algorithms derived from the proposed method have been implemented and tested on examples of diverse structures.

At our knowledge, the discovery of PN including duplicated transitions as a solution to handle event logs that involves silent transitions behaviour has not been addressed in related works.; Current research addresses the transformation of PN with duplicated transitions into PN with silent transitions.

REFERENCES

[1] Estrada-Vargas, A., Lopez-Mellado, E. and Lesage, J-J. A Black-Box Identification Method for Automated Discrete-Event Systems. *IEEE Trans. on Automation Science and Engineering*, 14(3), pp.1321-1336. (2017).

[2] Van der Aalst, W., Weijters, T. and Maruster, L. Workflow mining: discovering process models from event logs. *IEEE Trans. on Knowledge and Data Engineering*, 16(9), pp.1128-1142. (2004).

[3] Estrada-Vargas, A., Lopez-Mellado, E. and Lesage, J-J. A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems. *Mathematical Problems in Engineering*, (2010), pp.1-21.

[4] Cabasino, M., Darondeau, P., Fanti, M. and Seatzu, C. Model identification and synthesis of discrete-event systems. In: M. Zhou, H. Li and M. Weijnen, ed., *Contemporary Issues in Systems Science and Engineering*. IEEE/Wiley Press Book Series. (2013).

[5] Van der Aalst, W. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin. (2011).

[6] Wen, L., Wang, J., van der Aalst, W., Huang, B. and Sun, J. Mining process models with prime invisible tasks. *Data & Knowledge Engineering*, 69(10), pp.999-1021. (2010).

[7] Guo, Q., Wen, L., Wang, J., Yan, Z. and Yu, P. Mining invisible tasks in non-free-choice constructs. *Business Process Management: 13th Int. Conference*, 9253, pp.109-125. (2015).

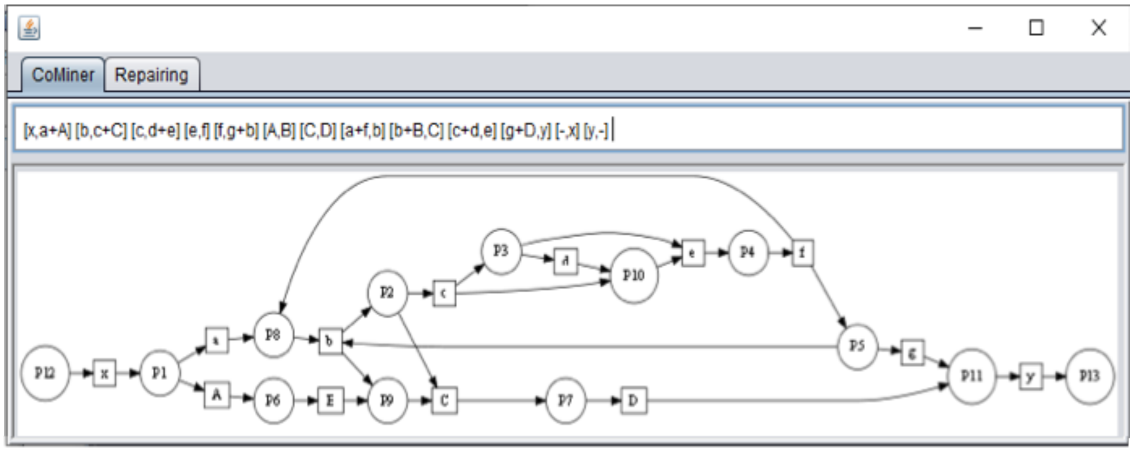
[8] Alvarez-Pérez, Y., Lopez-Mellado, E. Identifying Petri Nets with Silent Transitions by Event Traces Classification. IFAC 15th International Workshop on Discrete Event Systems, Rio de Janeiro, Brazil. May 2020. IFAC-PapersOnLine.

[9] García-Uribe, C., López-Mellado, E. An Event Clustering Method for Discovering Switch Silent Transitions in a Class of Petri Nets. *34th annual European Simulation and Modelling Conference. ESM'2020*. Toulouse France, October 2020.

[10] Tapia-Flores, T., Rodríguez-Pérez, E., and López-Mellado, E. Discovering process models from incomplete event logs using conjoint occurrence classes. *ATAED@Petri Nets/ACSD*, pp. 31-46. (2016).

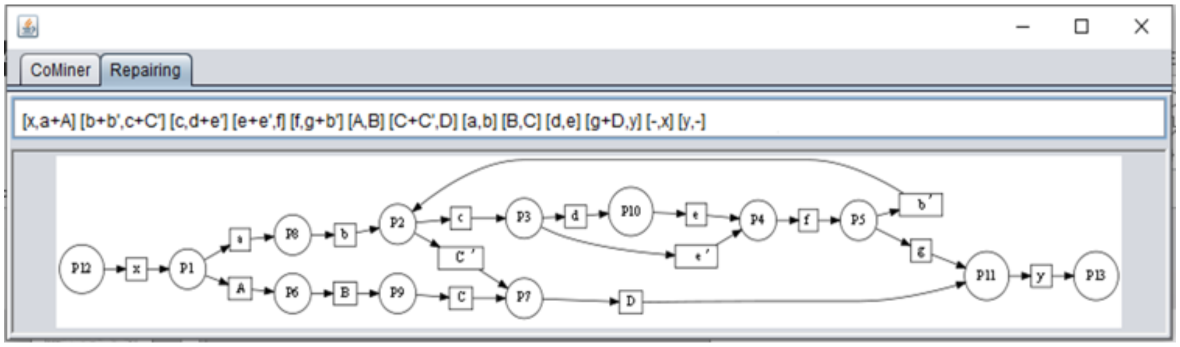
[11] Dingle, N. J., Knottenbelt, W. J., & Suto, T. PIPE2: a tool for the performance evaluation of generalised stochastic Petri Nets. *ACM SIGMETRICS Performance Evaluation Review*, 36(4), 34-39. (2009).

[12] Graphviz – Graph visualization software. https://graphviz.org



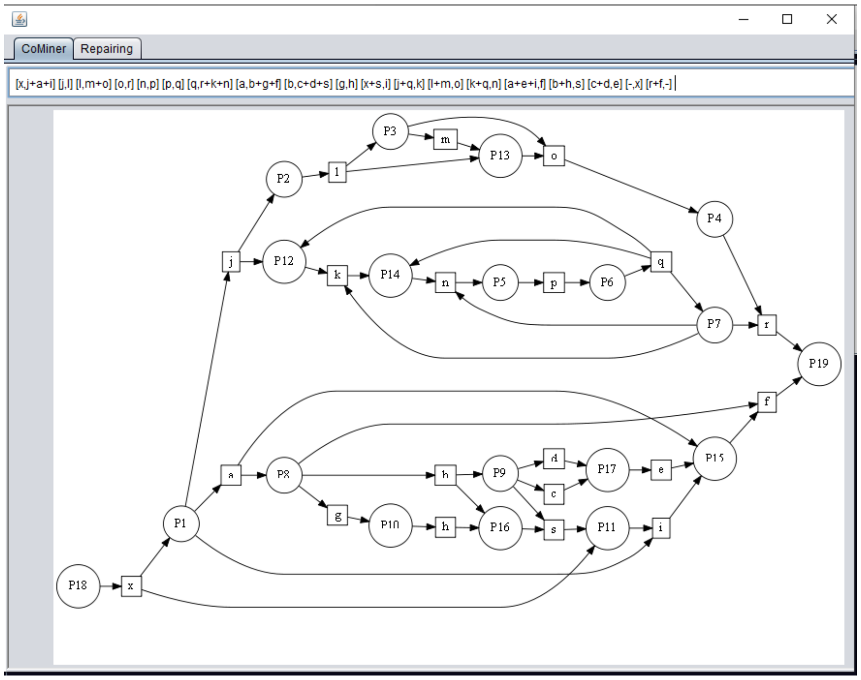
[x,a+A] [b,c+C] [c,d+e] [e,f] [f,g+b] [A,B] [C,D] [a+f,b] [b+B,C] [c+d,e] [g+D,y] [-,x] [y,-]

Figure 12. PN discovered by CoMiner from λ_1



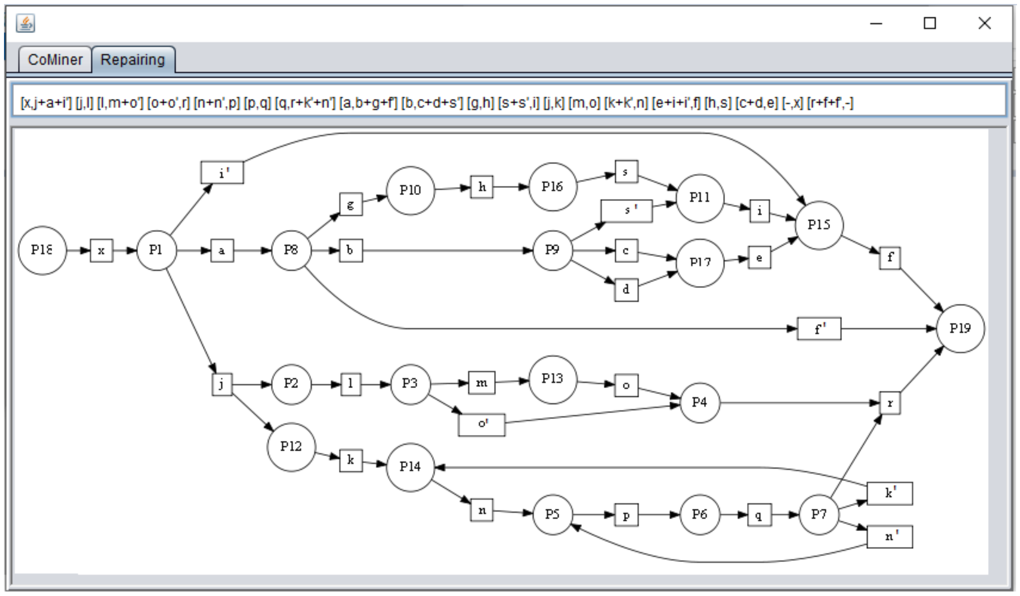
[x,a+A] [b+b',c+C'] [c,d+e'] [e+e',f] [f,g+b'] [A,B] [C+C',D] [a,b] [B,C] [d,e] [g+D,y] [-,x] [y,-]

Fig. 13. PN repaired with duplicated transitions equivalent to that in Fig.12.



[x,j+a+i] [j,l] [l,m+o] [o,r] [n,p] [p,q] [q,r+k+n] [a,b+g+f] [b,c+d+s] [g,h] [x+s,i] [j+q,k] [l+m,o] [k+q,n] [a+e+i,f] [b+h,s] [c+d,e] [-,x] [r+f,-]

Fig. 14. PN discovered by CoMiner from λ_2.



[x,j+a+i'] [j,l] [l,m+o'] [o+o',r] [n+n',p] [p,q] [q,r+k'+n'] [a,b+g+f'] [b,c+d+s'] [g,h] [s+s',i] [j,k] [m,o] [k+k',n] [e+i+i',f] [h,s] [c+d,e] [-,x] [r+f+f',-]

Fig. 15. PN repaired with duplicated transitions equivalent to the model in Fig. 14.