Detecting Data-flow Errors Based on Petri Nets With Data Operations

Dongming Xiang, Guanjun Liu, *Member, IEEE*, Chungang Yan, and Changjun Jiang

***Abstract*—In order to guarantee the correctness of business processes, not only control-flow errors but also data-flow errors should be considered. The control-flow errors mainly focus on deadlock, livelock, soundness, and so on. However, there are not too many methods for detecting data-flow errors. This paper defines Petri nets with data operations (PN-DO) that can model the operations on data such as read, write and delete. Based on PN-DO, we define some data-flow errors in this paper. We construct a reachability graph with data operations for each PN-DO, and then propose a method to reduce the reachability graph. Based on the reduced reachability graph, data-flow errors can be detected rapidly. A case study is given to illustrate the effectiveness of our methods.**

***Index Terms*—Business process modeling, data-flow errors, Petri nets, reachability graph.**

I. Introduction

BUSINESS process models pay attention to both control-flow and data-flow [1]. A good model contributes to the correctness verification of control/data-flows. Control-flow is concerned with the partial orders of tasks (e.g., workflow process [2]), while data-flow focuses on data operations. Since a large number of data operations are executed in a business process, errors easily take place if the data-flow is designed improperly. These errors include missing data, redundant data, lost data, inconsistent data, and so on [3]. Therefore, it is necessary and interesting to detect them. However, most existing studies mainly focus on the control-flow error detection, such as deadlock, livelock, infinite loop, lack of synchronization, compatibility, and soundness [4]−[8]. This paper is about the data-flow error detection.

There have been some studies on data-flows. Sadiq *et al.* [9] propose seven kinds of data-flow anomalies, but do not provide any detection methods. Sharma *et al.* [10] detect some data-flow errors based on business process modeling notation (BPMN). Guo *et al.* [11] address data exchange problems in the inter-organizational workflow, and calculate an exact data set to ensure that the integrated workflow is free of data-flow errors. Sun *et al.* [12] use UML diagram to obtain the dependence relationship of business processes according to their data association, and then detect each process instance under some given rules. Their work is generalized in [13] and [6] where a systematic graph traversal approach is proposed to detect data-flow errors. These studies lack a completely formal semantics. The following work utilizes some formal methods to detect some data-flow errors.

A dual flow net (DFN) [14] models the control- and data-flows in an embedded system. Fan *et al.* [15] simplify DFN into dual workflow net in order to detect the soundness of business processes. Based on the work in [10], Awad *et al.* [16] map a BPMN into a Petri net, and then detect and repair its errors. Although these formal models can represent the read/write operations, they are usually short of a complete semantics of concurrent read and/or coverable write. Contextual nets [17], [18] can represent concurrent read but not coverable write. In addition, some formal models like workflow nets with data (WFD-nets) [3] utilize label functions to describe data operations. They check the data-flow errors based on computation tree logic (CTL). However, this technique suffers from the state space explosion problem.

There exist some reduction techniques to alleviate the state space explosion problem. Partial-order methods, like persistent sets [19], ample sets [20] and stubborn sets [21], do a state-space search in which a subset of transitions are computed and explored from each state. For an infinite state space of an unbounded Petri net, a new reduced reachability graph is proposed in [22] and [23]. Instead of enumerating all the reachable states, symmetry reduction [24] gives equivalence classes of states w.r.t. the symmetry relation. The state hashing method (e.g., Hash compaction [25], bit-state hashing [26]) utilizes hash techniques to reduce memory usage, i.e., each state is represented by hash values instead of full state representations. In the sweep-line method [27], some certain states of a system can be deleted based on a measure of progress. The above reduction techniques are used to verify some classical properties like reachability and deadlock rather than data-flow errors. This paper utilizes the reduction technique to detect the data-flow errors.

In this paper, we define a kind of Petri nets called Petri nets with data operation (PN-DO) that can model both concurrent read and coverable write. In order to detect data-flow errors, we construct a reachability graph with data operations for each PN-DO, and then present a detection algorithm. A new method is proposed to reduce the reachability graph. Based on the reduced reachability graph, the data-flow errors can be

Manuscript received November 8, 2016; accepted January 16, 2017. This work was supported in part by the National Key R & D Program of China (20 17YFB1001804) and Shanghai Science and Technology Innovation Action Plan Project (16511100900). Recommended by Associate Editor Zhiwu Li. *(Corresponding authors: Guanjun Liu.)*

Citation: D. M. Xiang, G. J. Liu, C. G. Yan, and C. J. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA J. of Autom. Sinica*, vol. 5, no. 1, pp. 251−260, Jan. 2018.

D. M. Xiang, G. J. Liu, C. G. Yan, and C. J. Jiang are with the Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 201804, China (e-mail: flysky_xdm@163.com; li ugj1116@163.com; yanchungang@tongji.edu.cn; cjjiang@tongji.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JAS.2017.7510766

detected rapidly. Compared with our previous work in [28], we further detect redundant data and lost data. Most importantly, we propose a new method of reducing reachability graph to alleviate the state space explosion.

The rest of this paper is organized as follows. Section II introduces PN-DO. Section III defines 4 types of data-flow errors based on PN-DO. Section IV proposes an algorithm to detect data-flow errors based on the reachability graphs of PN-DOs. Section V gives a method to reduce the reachability graph in order to detect data-flow errors rapidly. Section VI shows a case study. The last section summarizes this paper.

II. PN-DO Model

This section first introduces read arcs and write arcs, and then gives the definition of PN-DO.

A. Read Arc

In the traditional Petri net, the operation of reading a data is described as a self-loop [18]. For instance, t_1 and t_2 in Fig. 1 (a) both read the data from the place v_1. However, self-loops cannot represent the concurrency semantics of reading the shared data [29]. Since the place v_1 has only one token, only one of t_1 and t_2 is fired at one time, but they cannot be fired concurrently. Therefore, read operations are represented in the form of Fig. 1 (b) [30]. In Fig. 1 (b), the dot line between v_1 and t_1 only represents that the token in v_1 is a condition of enabling t_1, but it is not consumed when t_1 is fired. Hence, t_1 and t_2 can be fired concurrently. This paper uses the read arc in [29], [30]. Notice that the two Petri nets in Fig. 1 have the same reachability graph that is an interleaving-semantics-based technique. If we indicate their differences under the concurrency semantics, we can use other techniques such as branching processes [17], [31], [32].

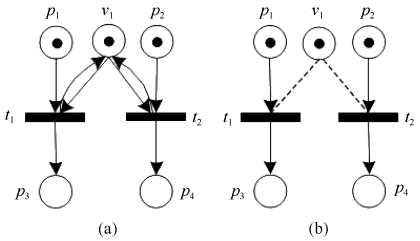


Fig. 1: (a) Self-loop representing a read operation. (b) Read arcs.

B. Write Arc

Write operations usually consider data-generation and/or data-update. Data-generation means that a variable is initialized via a write operation, i.e., this variable has no value before this write operation is executed. Data-update means that the original value of a variable is replaced by a new value.

1) If Fig. 2 (a) is used to model a data-generation (i.e., there is no token in v_1), then a token is produced into v_1 after t is fired. However, if there is a token in v_1, v_1 has two tokens after t is fired. Therefore, it cannot represent a data-update.

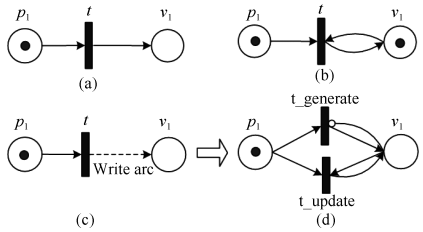


Fig. 2: (a) Data-generation. (b) Data-update. (c)−(d) A write arc and its equivalent part.

2) If Fig. 2 (b) is used to model a data-update (i.e., there is a token in v_1), then there is still a token in v_1 after t is fired. However, if there is no token in v_1, then t_1 cannot be fired. This means that Fig. 2 (b) cannot represent a data-generation.

In summary, it is not easy to use a traditional Petri net to model both data-generation and data-update.

Fortunately, Petri nets with inhibitor arcs [33] can do this as shown in Fig. 2 (d). However, this model is too complex. In this paper we present a write arc as shown in Fig. 2 (c) that is simple obviously. Especially, for concurrent write operations, the reachability graph of our model is much simpler than that of Petri net with inhibitor arcs (e.g., Fig. 3 shows this).

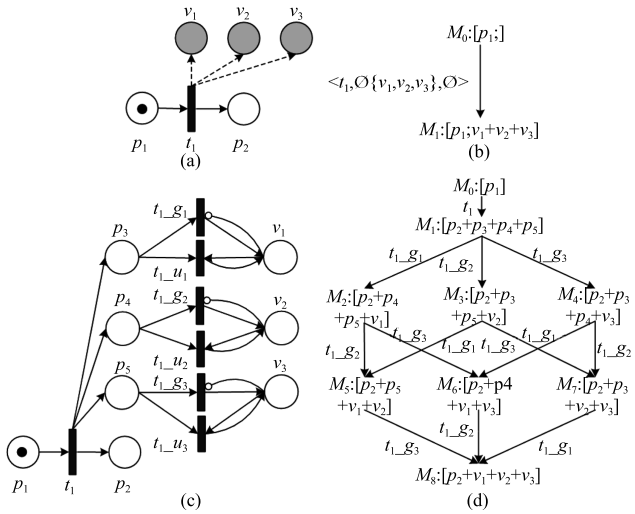


Fig. 3: Concurrent write operations. (a)−(b) A net with 3 write arcs, and its extended reachability graph. (c)−(d) The equivalent model with inhibitor arcs, and its reachability graph.

Some related models about writing data are existing. Varea *et al.* [14] consider data-generation. Data-update is considered in [34] and [35]. Although data-generation and data-update are both considered in [36] and [37] using a kind of write arc, they are about high-level Petri nets in which variables and their values are required by their write arcs. In this paper we focus on data-flows rather than the concrete variables and values, but we refer to the main ideas of [36] and [37].

C. PN-DO

$\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of non-negative integers.

Definition 1 (N-DO):

A net with data operations (N-DO) is a 3-tuple $N = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d)$, where

1) C is the set of control places, V is the set of data places, and $C \cap V = \emptyset$;

2) T is the set of transitions;

3) $F_c \subseteq C \times T \cup T \times C$ is the set of control arcs;

4) $F_r \subseteq V \times T$ is the set of read arcs;

5) $F_w \subseteq T \times V$ is the set of write arcs; and

6) $F_d \subseteq V \times T$ is the set of delete arcs such that $F_r \cap F_d = \emptyset$.

A marking of an N-DO is a mapping M: $V \cup C \to \mathbb{N}$. A Petri net with data operations (PN-DO) is an N-DO N with an initial marking M_0 and denoted as $\Sigma = (N, M_0)$.

In a PN-DO diagram, a control arc is a line with an arrow, a read arc is drawn as a dashed line without arrows, and a write/delete arc is a dashed line with an arrow. For instance, Fig. 4 shows a PN-DO, where $C = \{p_1 \cdots p_6\}$, $V = \{v_1, v_2\}$, the arc from p_1 to t_1 is a control arc, the dashed line from v_2 to t_4 is a read arc, the dashed arc from t_1 to v_1 is a write arc, the dashed arc from v_1 to t_3 is a delete arc, and $[p_1;]$ is its initial marking.

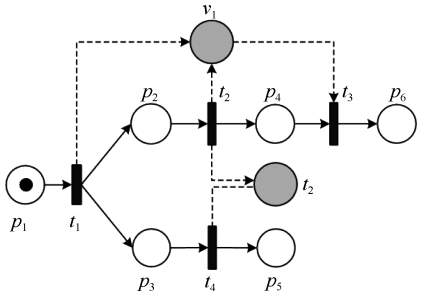


Fig. 4: PN-DO.

The following notations are used in this paper:

1) ${}^c t = \{p | p \in C \wedge (p,t) \in F_c\}$ is a control pre-set of t;

2) $t^c = \{p | p \in C \wedge (t,p) \in F_c\}$ is a control post-set of t;

3) ${}^r t = \{v | v \in V \wedge (v,t) \in F_r\}$ is a read-set of t;

4) $t^w = \{v | v \in V \wedge (t,v) \in F_w\}$ is a write-set of t;

5) ${}^d t = \{v | v \in V \wedge (v,t) \in F_d\}$ is a delete-set of t;

6) $PreD(t) = {}^d t \cup {}^r t$;

7) $PostD(t) = t^w$.

Definition 2 (Control-enabledness and Data-enabledness): Given a PN-DO $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$:

1) $t \in T$ is control-enabled at a marking M, which is denoted by $M[_c t\rangle$, if $\forall p \in {}^c t$: $M(p) > 0$. Otherwise, it is not control-enabled and denoted by $\neg M[_c t\rangle$;

2) $t \in T$ is data-enabled at a marking M, which is denoted by $M[_v t\rangle$, if $\forall v \in PreD(t)$: $M(v) > 0$. Otherwise, it is not data-enabled and denoted by $\neg M[_v t\rangle$.

If t is control-enabled and data-enabled at M, then it is enabled at M and denoted as $M[t\rangle$. Otherwise, it is disabled and denoted by $\neg M[t\rangle$. Firing an enabled transition t at M yields a marking M', and it is denoted by $M[t\rangle M'$ such that $\forall s \in C \cup V$:

$$M'(s) = \begin{cases} M(s) - 1, & \text{if } s \in {}^d t \vee s \in {}^c t \setminus t^c \\ M(s) + 1, & \text{if } (s \in t^w \wedge M(s) = 0) \vee s \in t^c \setminus {}^c t \\ M(s), & \text{otherwise.} \end{cases} \quad (1)$$

For example, at the initial marking $[p_1;]$ of the PN-DO in Fig. 4, t_1 is enabled and after firing it, we get the marking $[p_2 + p_3; v_1]$. At this marking, t_4 is control-enabled but not data-enabled since v_2 has no token.

A marking M' is reachable from another marking M, if there exists a firing sequence $\sigma = t_1 t_2 \cdots t_n$ such that $M[t_1\rangle M_1[t_2\rangle M_2 \cdots M_{n-1}[t_n\rangle M'$, i.e., $M[\sigma\rangle M'$. The set of markings reachable from M is denoted by $R(M)$. Notice that $t \in \sigma$ means that t occurs in σ.

Definition 3 (Safeness): A PN-DO is safe if $\forall M \in R(M_0)$, $\forall s \in C \cup V : M(s) \leq 1$.

This paper only considers the safe PN-DOs. There are two reasons why we only focus on the safe PN-DOs. First, it is not easy for us to define concurrency and conflict in a non-safe PN-DO. For example, if two transitions have the same input place (i.e., conflict) and the place has multiple tokens (i.e., they can concurrently occur), then their concurrency and conflict are not easily distinguished and thus our reduction technique cannot be used. Notice that the definitions of concurrency and conflict are given in the next subsection, and the reduction technique is given in Section V. The second reason is that many applications (like the example in Section VI) can be modeled by the safe PN-DOs.

D. Conflict Relation and Concurrency Relation

In a PN-DO, if two transitions are both enabled at a marking, then their relation is either conflict or concurrency. The two relations are determined by their control pre-set and marking distributions.

Definition 4 (Conflict and Concurrency): Given a marking M of a safe PN-DO, two different transitions t_1 and t_2 are in

1) a conflict relation at M, which is denoted by $t_1 +_M t_2$, if $M[_c t_1\rangle \wedge M[_c t_2\rangle \wedge ({}^c t_1 \cap {}^c t_2 \neq \emptyset)$; or

2) a concurrency relation at M, which is denoted by $t_1 \|_M t_2$, if $M[_c t_1\rangle \wedge M[_c t_2\rangle \wedge ({}^c t_1 \cap {}^c t_2 = \emptyset)$.

For example, t_1 and t_2 in Fig. 5 (a) are in a conflict relation at the initial marking $M_0 = [p_1 + p_2 + p_5;]$ since they satisfy $M_0[_c t_1\rangle$, $M_0[_c t_2\rangle$ and ${}^c t_1 \cap {}^c t_2 \neq \emptyset$. By contrast, t_1' and t_2' in Fig. 5 (b) are in a concurrency relation at the initial marking $M_0' = [p_1' + p_2' + p_5' + p_6';]$ because they satisfy $M_0'[_c t_1'\rangle$, $M_0'[_c t_2'\rangle$ and ${}^c t_1' \cap {}^c t_2' = \emptyset$.

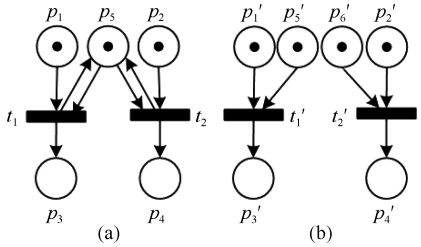


Fig. 5: (a) Conflict relation. (b) Concurrency relation.

Obviously, the two relations satisfy symmetry but not transitivity.

III. DATA-FLOW ERRORS

Data-flow errors are caused by improper data operations, which mainly include missing data, redundant data, lost data,

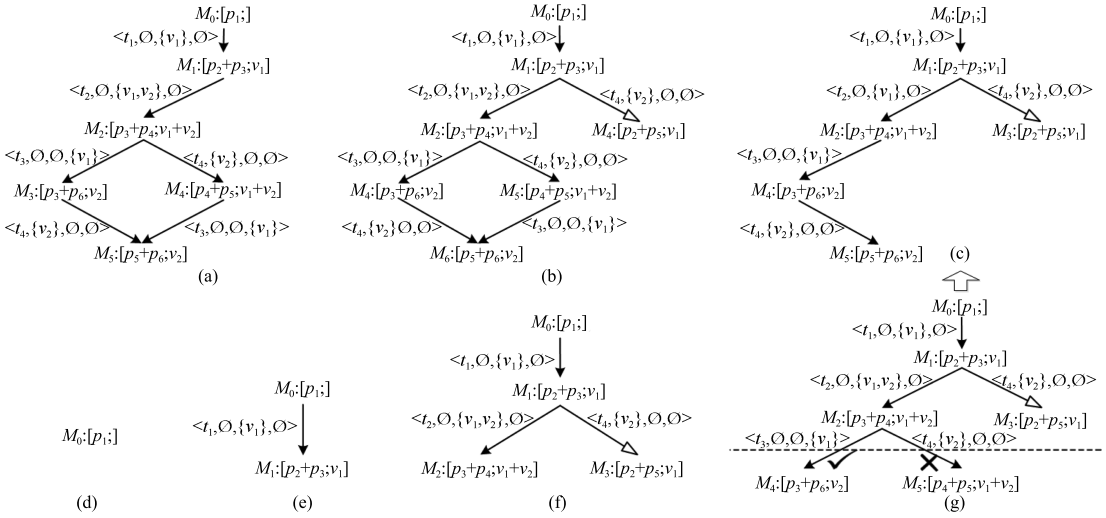


Fig. 6: For the PN-DO Σ in Fig. 4, its reachability graph, extended reachability graph, reduced ERG and prefixes are respectively: (a) $RG(\Sigma)$; (b) $ERG(\Sigma)$; (c) the reduced ERG; (d)−(g) the prefixes of $ERG(\Sigma)$.

and inconsistent data. In the following definitions, $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$ is a safe PN-DO.

Definition 5 (Missing Data): Σ has an error of missing data if $\exists t \in T$, $\exists M \in R(M_0)$: $M[_c t\rangle \wedge \neg M[_v t\rangle$.

Missing data occurs when a process is reading or deleting some data, but this data is not existing at this time. In Fig. 4, t_4 is control-enabled but not data-enabled at the reachable marking $[p_2 + p_3; v_1]$. It cannot read the data from v_2 since there is no token in v_2. At this time, missing data occurs.

Definition 6 (Inconsistent Data): Σ has an error of inconsistent data if $\exists M \in R(M_0)$, $\exists t_1, t_2 \in T$: $(t_1 \|_M t_2) \wedge ({}^r t_1 \cup {}^d t_1 \cup t_1^w) \cap ({}^d t_2 \cup t_2^w) \neq \emptyset$.

Inconsistent data occurs when one process is reading or writing or deleting some data, but another process is concurrently writing or deleting this data. In Fig. 4, at the reachable marking $[p_2 + p_3; v_1]$, t_2 is writing a data into v_2, but t_4 is reading v_2 concurrently. At this time, inconsistent data occurs.

Definition 7 (Redundant Data): Σ has an error of redundant data if one of the following two conditions holds:

1) $\exists M_1, M_2 \in R(M_0)$, $\exists t_1 \in T$, $\exists v \in V$: $M_1[t_1\rangle M_2 \wedge v \in t_1^w \wedge (\forall M_3 \in R(M_2), \forall t_2 \in T$: $M_3[t_2\rangle \rightarrow v \notin {}^r t_2)$;

2) $\exists M_1, M_2 \in R(M_0)$, $\exists t_1, t_2 \in T$, $\exists \sigma \in T^*$, $\exists v \in V$: $M_1[t_1\sigma\rangle M_2[t_2\rangle \wedge v \in t_1^w \wedge v \in {}^d t_2 \wedge (\forall t_3 \in \sigma : v \notin {}^r t_3)$.

Redundant data occurs if a data is never read before it is deleted or the business process terminates. In Fig. 4, at the reachable marking $[p_2 + p_3; v_1]$, t_2 is to overwrite the data of v_1. But the data has never been read before it is deleted by t_3. Therefore, there is an error of redundant data.

Definition 8 (Lost Data): Σ has an error of lost data if $\exists M_1$, $M_2 \in R(M_0)$, $\exists t_1, t_2 \in T$, $\exists \sigma \in T^*$, $\exists v \in V$: $M_1[t_1\sigma\rangle M_2[t_2\rangle \wedge v \in t_1^w \cap t_2^w \wedge (\forall t_3 \in \sigma : v \notin {}^r t_3)$.

Lost data means that a data has never been read before it is overwritten. In Fig. 4, t_1 writes a data into v_1. But this data has never been read before it is overwritten by t_2. Therefore, this is an error of lost data.

IV. Data-flow Errors Detection Based on PN-DO

This section first constructs the reachability graph of a PN-DO based on its firing rules, then we extend the reachability graph via considering those transitions that are control-enabled but not data-enabled at some reachable markings. Finally, a detection algorithm for data-flow errors is proposed based on the extended reachability graph.

A. Reachability Graph

Definition 9 (Reachability Graph, RG): Let $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$ be a PN-DO. $RG(\Sigma) = (R(M_0), E, \ell)$ is the reachability graph of Σ, where

1) $E = \{(M_i, M_j) | M_i, M_j \in R(M_0) \wedge \exists t \in T : M_i[t\rangle M_j\}$;

2) ℓ: $E \rightarrow T \times 2^V \times 2^V \times 2^V$ such that $\ell(M_i, M_j) = \langle t, {}^r t, t^w, {}^d t\rangle$ if $(M_i, M_j) \in E$ and $M_i[t\rangle M_j$.

For example, Fig. 6 (a) is the reachability graph of the PN-DO in Fig. 4. We know that the errors of missing data and inconsistent data occur at the reachable marking $M_1 = [p_2 + p_3; v_1]$, but this reachability graph cannot directly indicate these errors. Therefore, in order to easily detect these errors, we extend the reachability graph, i.e., a control-enabled but not data-enabled transition is added into the reachability graph. The related edges are drawn as lines with empty arrows.

Definition 10 (Pseudo Reachable Marking): Let $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$ be a PN-DO, $M \in R(M_0)$, and $t \in T$. If t is control-enabled but not data-enabled, then M' is called a pseudo reachable marking from M via t, where

$$M'(s) = \begin{cases} M(s) - 1, & \text{if } s \in {}^c t \setminus t^c \\ M(s) + 1, & \text{if } s \in t^c \setminus {}^c t \\ M(s), & \text{otherwise.} \end{cases} \quad (2)$$

A pseudo reachable marking is yielded by operating control places only. We denote $M[_c t\rangle M'$ if M' is a pseudo reachable marking from M via t. We add all pseudo reachable markings into the reachability graph and then get the extended reachability graph (ERG). The following definition is its formal representation.

Definition 11 (Extended Reachability Graph, ERG): Let $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$ be a PN-DO, $ERG(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ is the extended reachability graph of Σ, where

1) $(R(M_0), E, \ell)$ is the reachability graph of Σ;

2) $R'(M_0)$ is the set of all pseudo reachable markings;

3) $E' = \{(M_i, M_j) | M_i \in R(M_0) \wedge M_j \in R'(M_0) \wedge \exists t \in T : M_i[_c t\rangle M_j\}$; and

4) $\ell' : E' \to T \times 2^V \times 2^V \times 2^V$ such that $\ell'(M_i, M_j) = \langle t, {}^r t, t^w, {}^d t\rangle$ if $(M_i, M_j) \in E'$ and $M_i[_c t\rangle M_j$.

For example, Fig. 6 (b) is the ERG of the PN-DO in Fig. 4, where M_3 is a pseudo reachable marking such that $M_1[_c t_4\rangle M_3, (M_1, M_3) \in E'$ and $\ell(M_1, M_3) = \langle t_4, \{v_2\}, \emptyset, \emptyset\rangle$.

B. Data-flow Error Detection Based on ERG

A detection method for data-flow errors is proposed based on ERG, as shown in Algorithm 1.

Algorithm 1. Data-flow error detection algorithm

Input: A PN-DO $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$.

Output: All data-flow errors.

1) Initialize

Marking set $MD = \emptyset$; /* The detected markings. */
Construct $ERG(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$, and generate its concurrency hash-table *ConHash*;

2) Detect data-flow errors

for each $M \in R(M_0)$ such that $M \notin MD$ **do**
  $MD.add(M)$;
  $Detect\ IS\ Data(M, ConHash, ERG(\Sigma))$;
  /* Detect inconsistent data, shown in Procedure_1. */
  $E_1 = \{(M, M') | (M, M') \in E \cup E'\}$;
  for each $(M, M') \in E_1$
    if $(M, M') \in E'$ and $M[_c t\rangle M'$
      print Missing Data;
    else
      if $M[t\rangle M'$ and $t^w \neq \emptyset$
        for each $v \in t^w$
          $MT = \emptyset$; /* The traversed markings */
          Set $M_r = M_w = M_d = \emptyset$;
          $FindRWD(v, M', ERG(\Sigma), MT)$;
          /* As shown in Procedure_2, it is used to compute M_r, M_w and M_d */
          if $M_w \neq \emptyset$
            print Lost Data;
          end if
          if $M_d \neq \emptyset$
            print Redundant Data;
          end if
          if $M_r = M_w = M_d = \emptyset$
            print Redundant Data;
          end if
        end for
      end if
    end if
  end for
end for

Procedure_1 *DetectISData* (M, $ConHash$, $ERG(\Sigma)$)

/* Detect inconsistent data */
$T^{co} = ConHash.get(M)$;
/* Get all pairs of concurrent transitions at M; */
if $T^{co} \neq \emptyset$
  for each $(t_1, t_2) \in T_c$
    if $({}^r t_1 \cup t_1^w \cup {}^d t_1) \cap (t_2^w \cup {}^d t_2) \neq \emptyset \vee ({}^r t_2 \cup t_2^w \cup {}^d t_2) \cap (t_1^w \cup {}^d t_1) \neq \emptyset$
      print Inconsistent Data between t_1 and t_2;
    end if
  end for
end if

Procedure_2 *FindRWD*(v, M', $ERG(\Sigma)$, MT)

/* Traverse all markings reachable from M', and obtain three reachable marking sets M_r, M_w and M_d. M_r (resp. M_w, M_d) is the set of reachable markings at which there is a read (resp. write, delete) operation. */
if $M' \notin MT$ **then**
  $MT.add(M')$;
  $E_2 = \{(M', M'') | (M', M'') \in E \cup E'\}$;
  if $E_2 \neq \emptyset$ **then**
    for each $(M', M'') \in E_2$ **do**
      if $M'[t'\rangle M''$ or $M'[_c t'\rangle M''$ **then**
        if $v \in {}^r t'$ **then**
          $M_r.add(M')$;
        end if
        if $v \in t'^w$ **then**
          $M_w.add(M')$;
        end if
        if $v \in {}^d t'$ **then**
          $M_d.add(M')$;
        end if
        if $v \notin^r t' \cup t'^w \cup^d t'$ **then**
          FindRWD(v, M'', $ERG(\Sigma)$, MT);
        end if
      end if
    end for
  end if
end if

1) In Algorithm 1, we first construct the ERG of a PN-DO and a hash-table $ConHash$. For each node M in the ERG, this hash-table stores all pairs of transitions that are in the concurrency relation at M, i.e.,

$$ConHash.get(M) = \{(t_i, t_j) | t_i \|_M t_j\}.$$

2) An edge in E' indicates an error of missing data. Based on this hash-table, we detect inconsistent data for concurrent transitions through the following function

$$DetectISData(M, ConHash, ERG(\Sigma)).$$

3) If a transition enabled at a marking is to write a data, we traverse all successors of this marking. Then, the markings related to operations on this data are obtained by the function

$$FindRWD(v, M', ERG(\Sigma), MT).$$

Finally, according to these markings, we determine whether there is an error of redundant data or lost data.

The time complexity of Algorithm 1 is $O(W \times K^2)$, where K represents the number of markings, and W denotes the number of write arcs. Obviously, the detection complexity depends on the number of markings. However, the number of markings grows exponentially with the size of a net. Therefore, we propose a method to reduce an ERG.

V. A Reduced ERG

Our reduction method depends on the following two facts.

1) When two concurrent transitions conduct operations on the different data, no data-flow error occurs. Therefore, for the multiple paths yielded by these concurrent transitions, only one can be kept in the reduced ERG.

2) When two concurrent transitions conduct the same operations on the same data (i.e., concurrently write, or concurrently delete), an error occurs. At this time, each path in the ERG yielded by the two transitions can reflect this error. Therefore, we only need to keep one path in the reduced ERG for this case.

Notice that when two concurrent transitions conduct the different operations on the same data (e.g., one is write, and another one is read), an error occurs too. However, the multiple paths yielded by the two transitions are all kept in the reduced ERG. If one path is deleted, this error possibly cannot be represented in the reduced ERG.

Our method is not to cut these paths after the ERG is produced. In fact, they are not yielded in the process of constructing the reduced ERG. In order to describe our algorithm, we need to introduce the following concepts.

Definition 12 (Prefix): Let $ERG(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the ERG of a PN-DO Σ. A prefix of $ERG(\Sigma)$ is a subgraph of the ERG such that M_0 is in the prefix, and for each node M of this subgraph, there is a directed path from M_0 to M in the subgraph.

For example, Figs. 6 (d), (e) and (f) are three prefixes of ERG in Fig. 6 (b), where the prefix in Fig. 6 (d) has only the initial marking and is called the basic prefix. For convenience, a prefix of the ERG is denoted by $G_1 = (R_1, E_1, \ell_1)$.

Definition 13 (Possible Extension): Let $ERG(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the ERG of a PN-DO Σ, $G_1 = (R_1, E_1, \ell_1)$ be a prefix of $ERG(\Sigma)$, and $M \in R_1$ be a node of the prefix. (e, M') is a possible extension of the prefix at M if $e = (M, M') \wedge e \notin E_1 \wedge e \in E \cup E'$. A new prefix $G_2 = (R_2, E_2, \ell_2)$ of $ERG(\Sigma)$ is generated after (e, M') is added into G_1, where

1) $R_2 = R_1 \cup \{M'\}$;

2) $E_2 = E_1 \cup \{e\}$;

3) $\forall e' \in E_1 : \ell_2(e') = \ell_1(e')$, and $\ell_2(e) = (\ell \cup \ell')(e)$.

For example, at M_1 of the prefix in Fig. 6 (e), there are two possible extensions $((M_1, M_2), M_2)$ and $((M_1, M_3), M_3)$. Adding the two possible extensions into this prefix, a new prefix is generated as shown in Fig. 6 (f). The notions of prefix and possible extensions refer to the work of unfolding of Petri nets [38], [39].

The idea of generating a reduced ERG is that starting from the basic prefix, we repeatedly add some possible extensions into it until no possible extensions can be added. Obviously, the key is that which possible extensions of a given prefix can be added and which ones cannot.

Given two possible extensions at a marking of a prefix, if the corresponding transitions are in the concurrency relation, then we should consider whether one of them is not added. If no inconsistent data exists in them, or there is an inconsistent data but they conduct the same operation on the same data, then we select one from the two possible extensions to add it into the prefix, while the other one is not added. Their formal descriptions are in selection criterion. Except for the above case, other possible extensions are added.

Selection Criterion: Let $ERG(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the *ERG* of a PN-DO, and $G_1 = (R_1, E_1, \ell_1)$ be a prefix of $ERG(\Sigma)$, $M \in R_1$, and (e_1, M_1) and (e_2, M_2) be two possible extensions of the prefix at M such that $\ell_1(e_1) = \langle t_1, {}^r t_1, t_1^w, {}^d t_1 \rangle \wedge \ell_1(e_2) = \langle t_2, {}^r t_2, t_2^w, {}^d t_2 \rangle \wedge t_1 \| _M t_2$. If one of the two following conditions holds, one of the two possible extensions is added into the prefix and the other one is not.

1) There is no inconsistent data between t_1 and t_2;

2) there is an inconsistent data between them but they satisfy $t_1^w \cap y^w \neq \emptyset$ or ${}^d x \cap {}^d y \neq \emptyset$.

For example, $((M_2, M_4), M_4)$ and $((M_2, M_5), M_5)$ are two possible extensions of the prefix in Fig. 6 (f) at M_2. Because there is no inconsistent data between t_3 and t_4 at M_2, they satisfy selection criterion. Hence, $((M_2, M_4), M_4)$ is added into this prefix but $((M_2, M_5), M_5)$ is not, as shown in Fig. 6 (g). Certainly, we can also select the latter but not the former. By this method we can get a reduced ERG of the PN-DO in Fig. 4, as shown in Fig. 6 (c).

Algorithm 2 illustrates the computation process of possible extensions, and its time complexity is $O(Z^2)$, where Z represents the number of enabled transitions at the marking M. Based on the reduced ERG, we can detect data-flow errors. The related algorithm is similar to Algorithm 1 and omitted here.

Algorithm 2. The possible extensions algorithm

Input: A PN-DO $\Sigma = (C \cup V, T, F_c \cup F_r \cup F_w \cup F_d, M_0)$ and a marking M.

Output: An extension set S.

1: Initialize: A transition pairs $P = \emptyset$, and a transition set $T_S = \emptyset$.

2: $T' = \{t | M[_c t\rangle, t \in T\}$.

3: Get concurrent transition pair T^{co} and conflict transition pair $T^{\#}$, where

$T^{co} = \{(t_1, t_2) | t_1 \|_M t_2, t_1, t_2 \in T'\}$

$T^{\#} = \{(t_1, t_2) | t_1 +_M t_2, t_1, t_2 \in T'\}$.

4: **for all** $(t_1, t_2) \in T^{co}$ and $t_1, t_2 \notin S$ **do**

5: According to $T^{\#}$, get two transition sets S_1 and S_2, which are respectively in conflict with t_1 and t_2, i.e.,

$S_1 = \{t | t +_M t_1, (t, t_1) \in T^+, t \in T'\}$

$S_2 = \{t | t +_M t_2, (t, t_2) \in T^+, t \in T'\}$.

6: Boolean $se = TRUE$; /* Selective selection. */

7: **for each** $x \in S_1 \cup \{t_1\}$, $y \in S_2 \cup \{t_2\}$ such that $(x, y) \notin P$ **do**

```
      P.add(x, y);
      if there exists (x, y) that does not satisfy selection criterion then
        se = FALSE; break;
      end if
    end for
    if se == FALSE then
      T_S.add(S_1 ∪ {t_1} ∪ S_2 ∪ {t_1});
    else
      if |S_1| ≤ |S_2| then
        T_S.add(S_1 ∪ {t_1});
      else
        T_S.add(S_2 ∪ {t_2});
      end if
    end if
  end for
  if |T'| = 1 or T^co = ∅ then
    T_S.add(T');
  end if
  for each t ∈ T_S do
    if M[t⟩M_1 then
      S.add ((M, M_1), M_1));
    end if
    if ¬M[_v t⟩ and M[_c t⟩M_2 then
      S.add ((M, M_2), M_2));
    end if
  end for
  return S
```

Our reduction method is suitable for these PN-DOs that have many concurrent transitions and satisfy the selection criterion. For example, Fig. 7 shows the best reduction case. There are 9 pairs of concurrent transitions in Fig. 7 (a), and Fig. 7 (b) shows its ERG. Because all concurrent transitions satisfy the selection criterion, we can reduce most of markings and arcs, and then obtain reduced ERGs as shown in Figs. 7 (c) and (d). Obviously, due to different selections of possible extensions, the reduced ERGs are possibly not unique.

It is possible that some errors of inconsistent data cannot be reflected in a reduced ERG. For example, the error of inconsistent data caused by the concurrent transitions t_3 and t_4 is not be detected based on the reduced ERG in Fig. 7 (d), but it can be detected in view of the reduced ERG in Fig. 7 (c). This is because the concurrency structure of t_3 and t_4 is deleted in Fig. 7 (d) after $((M_2, M_4), M_4)$ is selected and added into a prefix at the marking M_2. Hence, we cannot decide their concurrency relation and thus the error of inconsistent data cannot be checked. Therefore, the selection policy of possible extension is important, and we will improve it in the future work.

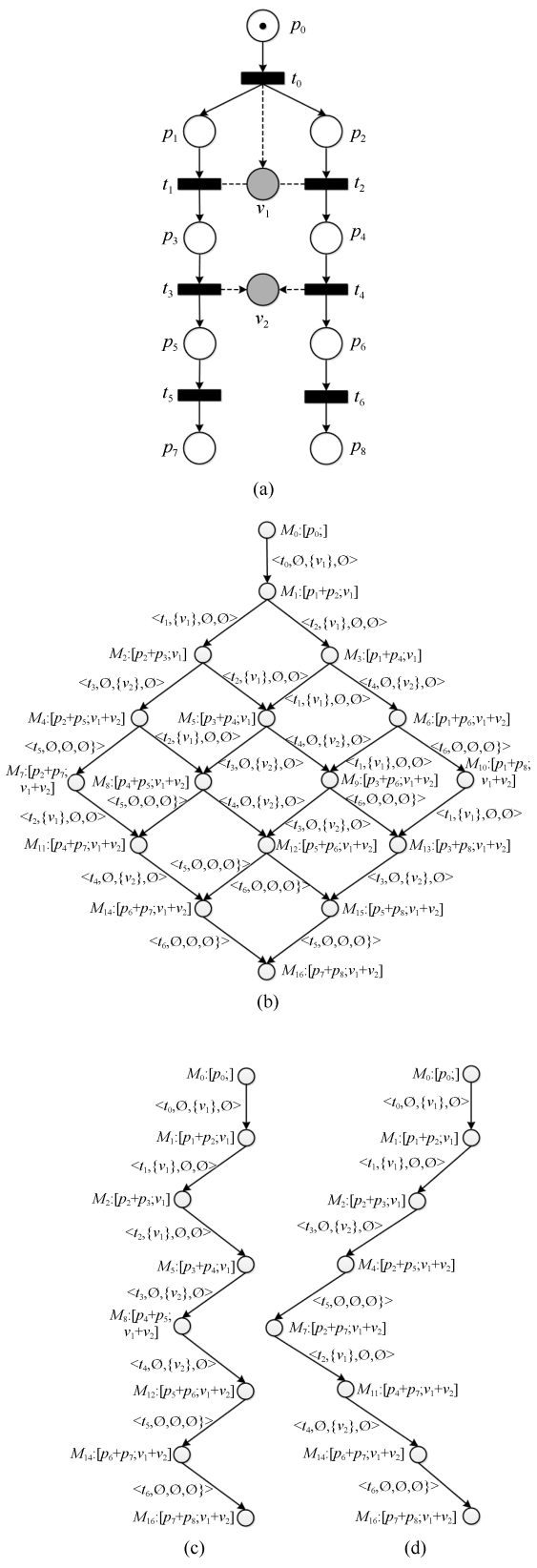


Fig. 7: Different reduced ERGs.

VI. Case Study

We give a case study by referring to the business process in [6]. This example is described as follows.

A company seeks and publishes its write-ups every month. Hence, some selected employees are assigned to collect and write these reports. After write-ups are submitted to the group manager (GM), the GM decides whether these reports are accepted or sent back for revision. Once they are accepted, they can undergo the second review from the department head (DH); otherwise, these write-ups will be asked for revision and

re-submitting. If these accepted reports fail to pass through the review from the DH, they will be asked to revise and resubmit to the DH. Otherwise, these write-ups are archived in the company's software repository and go through editorial review. For the reports which have passed through all reviews, where to publish them (on the Web sites or in the newsletter) will be decided by the editorial review. Then, the write-ups selected for the newsletter will be catalogued to print publication.

In order to describe the business process of write-up publication, it is modeled by a PN-DO, and then its data-flow errors are respectively detected based on ERG and the reduced ERG. Now, the concrete procedures are listed as follows.

1) This business process is modeled by the PN-DO in Fig. 8. The related data items and business activities are respectively shown in Tables I and II.

2) Fig. 9 is its ERG and Fig. 10 is its reduced *ERG*.

3) According to Algorithm 1, its data-flow errors are detected and listed in Table III.

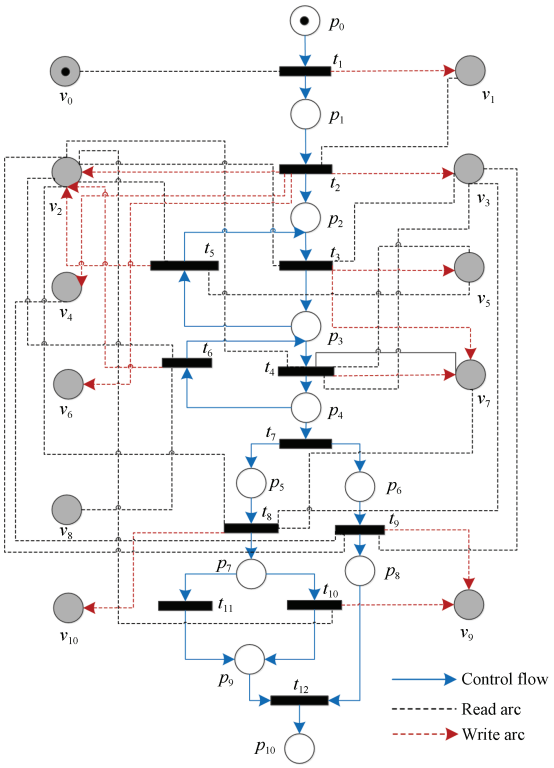


Fig. 8: Write-ups publication process (PN-DO Σ).

VII. Conclusion

Successful business process management depends on effective modeling and analysis. The model should consider both control- and data-flows, and the model checking technique should be effective. This paper proposes such a model and a technique. PN-DOs are defined to model both concurrent read and coverable write. A method of reducing their reachability graphs is proposed. Based on them, data-flow errors are detected rapidly.

TABLE I
Data Information

Data item	Name
v_0	Email ID
v_1	Email message
v_2	Report
v_3	Employee ID
v_4	Topic/Subject name
v_5	GM feedback
v_6	Contact number
v_7	Publish in website/newsletter
v_8	DH guideline/feedback
v_9	Article No.
v_{10}	Editorial review

TABLE II
Business Activities and Data Operations

Transition ID	Business activities	Input data (Read)	Output data (Write)
t_1	Send email	v_0	v_1
t_2	Submit report	v_1	v_2, v_3, v_4, v_6
t_3	Review by GM	v_2, v_3	v_5, v_7
t_4	Review by DH	v_2, v_3, v_5, v_7	v_7
t_5	Re-submit report	v_2, v_5	v_2
t_6	Revise report	v_2, v_8	v_2
t_7	-	-	-
t_8	Editorial review	v_2, v_3, v_7	v_{10}
t_9	Archive	v_2, v_3, v_4	v_9
t_{10}	Catalogue	v_2	v_9
t_{11}, t_{12}	-	-	-

In future work, we plan to improve the selection criterion that guarantees the reduced reachability graph can reflect all errors. Another work is to develop a software tool that uses the proposed algorithms to check the data-flow errors. We also plan to study the unfolding technique of PN-DOs to avoid state space explosion more effectively.

References

[1] C. C. Dolean and R. Petrusel, "Data-flow modeling: a survey of issues and approaches," *Inf. Econ.*, vol. 16, no. 4, pp. 117−130, Oct. 2012.

[2] W. M. Van Der Aalst, "Workflow verification: finding control-flow errors using petri-net-based techniques," in *Business Process Management: Models, Techniques, and Empirical Studies*, W. van der Aalst, J. Desel, and A. Oberweis, Eds. Berlin Heidelberg, Germany: Springer, vol. 1806, pp. 161−183, 2000.

[3] N. Trčka, W. M. P. Van der Aalst, and N. Sidorova, "Data-flow anti-patterns: Discovering data-flow errors in workflows," in *Proc. 21st Int. Conf. Advanced Information Systems Engineering*, Heidelberg, Germany, 2009, pp. 425−439.

[4] W. M. P. Aalst, K. M. Hee, A. H. M. Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Aspec. Comput.*, vol. 23, no. 3, pp. 333−363, 2011.

[5] G. J. Liu and C. J. Jiang, "Net-structure-based conditions to decide compatibility and weak compatibility for a class of inter-organizational workflow nets," *Sci. China Inf. Sci.*, vol. 58, no. 7, pp. 1−16, Jul. 2015.

[6] H. S. Meda, A. K. Sen, and A. Bagchi, "On detecting data flow errors in workflows," *J. Data Inf. Qual.*, vol. 2, no. 1, Article No. 4, Jul. 2010.

TABLE III
DATA-FLOW ERRORS LIST

Data-flow errors	Marking	Illustration
Missing data	M_6	v_8 is missing for t_6
Redundant data	M_1	v_6 is written by t_2 but never read again
	M_7, M_9	v_{10} is written by t_8 but never read again
Lost data	M_{10}, M_{11}, M_{12}	t_9 and t_{10} overwrite some data into v_9
Inconsistent data	M_8	t_9 and t_{10} overwrite some data into v_9 concurrently

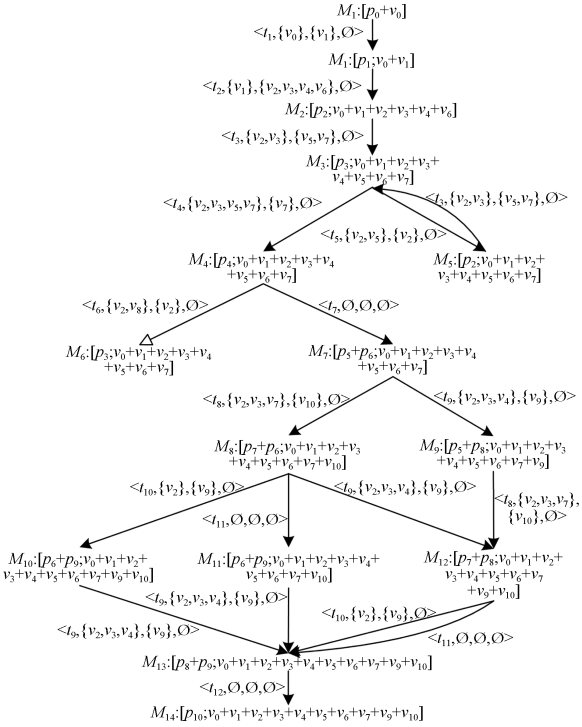


Fig. 9: The *ERG* of Σ.

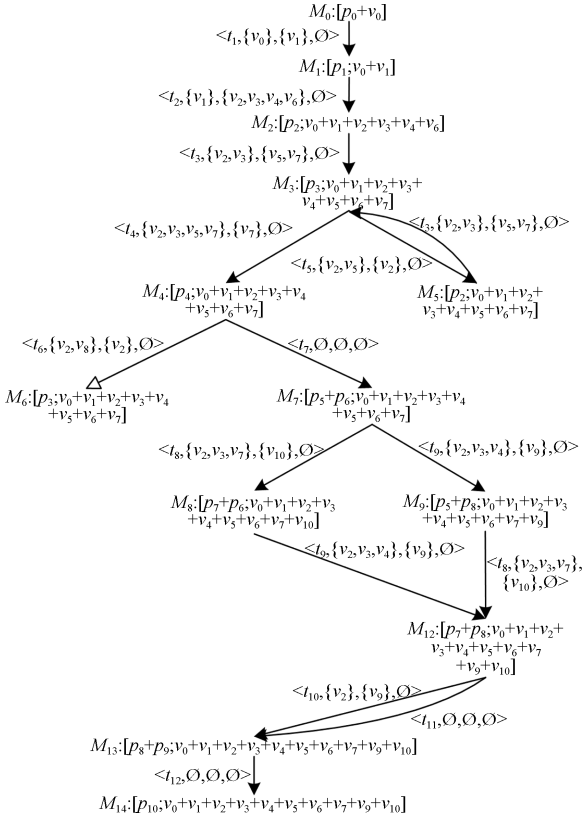


Fig. 10: The reduced *ERG* of Σ.

[7] S. Roy, A. S. M. Sajeev, S. Bihary, and A. Ranjan, "An empirical study of error patterns in industrial business process models," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 140−153, Apr.−Jun. 2014.

[8] S. G. Wang, M. D. Gan, and M. C. Zhou, "Macro liveness graph and liveness of omega-independent unbounded nets," *Sci. China Inf. Sci.*, vol. 58, no. 3, pp. 1−10, Mar. 2015.

[9] S. Sadiq, M. Orlowska, and W. Sadiq, "Data flow and validation in workflow modelling," in *Proc. 15th Australasian Database Conf.*, Dunedin, New Zealand, 2004, pp. 207−214.

[10] D. Sharma, S. Pinjala, and A. K. Sen, "Correction of data-flow errors in workflows," in *Proc. 25th Australasian Conf. Information Systems*, Auckland, New Zealand, 2014.

[11] X. T. Guo, S. X. Sun, and D. Vogel, "A dataflow perspective for business process integration," *ACM Trans Manage Inf. Syst.*, vol. 5, no. 4, Article No. 22, Mar. 2015.

[12] S. X. Sun, J. L. Zhao, J. F. Nunamaker, and O. R. L. Sheng, "Formulating the data-flow perspective for business process management," *Inf. Syst. Res.*, vol. 17, no. 4, pp. 374−391, Dec. 2006.

[13] H. S. Meda, A. K. Sen, and A. Bagchi, "Detecting data flow errors in workflows: A systematic graph traversal approach," in *Proc. 17th Annual Workshop on Information Technologies and Systems*, Montreal, Canada, 2007.

[14] M. Varea, B. M. Al-Hashimi, L. A. Cortës, P. Eles, and Z. B. Peng, "Dual flow nets: modeling the control/data-flow relation in embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 1, pp. 54−81, Feb. 2006.

[15] S. K. Fan, W. C. Dou, and J. J. Chen, "Dual workflow nets: mixed control/data-flow representation for workflow modeling and verification," in *Advances in Web and Network Technologies, and Information Management*, K. C. C. Chang, W. Wang, L. Chen, C. A. Ellis, C. H. Hsu, A. C. Tsoi, and H. X. Wang, Eds. Heidelberg, Germany: Springer, vol. 4537, pp. 433−444, 2007.

[16] A. Awad, G. Decker, and N. Lohmann, "Diagnosing and repairing data anomalies in process models," in *Business Process Management Workshops*, S. Rinderle-Ma, S. Sadiq, and F. Leymann, Eds. Berlin, Heidelberg, Germany: Springer, vol. 43, pp. 5−16, 2009.

[17] P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodrïguez, and S. Schwoon, "Efficient unfolding of contextual Petri nets," *Theor. Comput. Sci.* vol. 449, pp. 2−22, Aug. 2012.

[18] U. Montanari and F. Rossi, "Contextual nets," *Acta Inf.*, vol. 32, no. 6, pp. 545−596, Jun. 1995.

[19] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani,

"Partial-order reduction in symbolic state-space exploration," in *Proc. 9th Int. Conf. Computer Aided Verification*, Berlin, Heidelberg, Germany, 1997, pp. 340−351.

[20] A. Lluch-Lafuente, S. Edelkamp, and S. Leue, "Partial order reduction in directed model checking," in *Proc. 9th Int. SPIN Workshop on Model Checking of Software*, Grenoble, France, 2002, pp. 112−127.

[21] A. Valmari and H. Hansen, "Can stubborn sets be optimal?," in *Proc. 31st Int. Conf. Applications and Theory of Petri Nets*, Berlin, Heidelberg, Germany, 2010, pp. 43−62.

[22] S. G. Wang, M. C. Zhou, Z. W. Li, and C. Y. Wang, "A new modified reachability tree approach and its applications to unbounded Petri nets". *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 43, no. 4, pp. 932−940, Jul. 2013.

[23] S. G. Wang, M. D. Gan, M. C. Zhou, and D. You, "A reduced reachability tree for a class of unbounded Petri nets," *IEEE/CAA J. Autom. Sin.*, vol. 2, no. 4, pp. 345−352, Oct. 2015.

[24] P. A. Bourdil, B. Berthomieu, S. D. Zilio, and F. Vernadat, "Symmetry reduced state classes for time Petri nets," in *Proc. 30th Annu. ACM Symp. Applied Computing*, Salamanca, Spain, 2015, pp. 1751−1758.

[25] M. Westergaard, L. M. Kristensen, G. S. Brodal, and L. Arge, "The ComBack method: extending hash compaction with backtracking," in *Proc. 28th Int. Conf. Application and Theory of Petri Nets*, Berlin, Heidelberg, Germany, 2007, pp. 445−464.

[26] G. J. Holzmann, "An analysis of bitstate hashing," *Formal Methods Syst. Des.*, vol. 13, no. 3, pp. 289−307, Nov. 1998.

[27] S. Christensen, L. M. Kristensen, and T. Mailund, "A sweep-line method for state space exploration," in *Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg, Germany, 2001, pp. 450−464.

[28] D. M. Xiang, G. J. Liu, C. G. Yan, and C. J. Jiang, "Checking the inconsistent data in concurrent systems by petri nets with data operations," in *Proc. IEEE 22nd Int. Conf. Parallel and Distributed Systems (ICPADS)*, Wuhan, China, 2016, pp. 501−508.

[29] W. Vogler, "Partial order semantics and read arcs," in *Proc. 22nd Int. Symp. Mathematical Foundations of Computer Science*, Berlin, Heidelberg, Germany, 1997, pp. 508−517.

[30] W. Vogler, "Efficiency of asynchronous systems and read arcs in Petri nets," in *Proc. 24th Int. Colloquium on Automata, Languages, and Programming*, London, UK, 1998, pp. 538−548.

[31] G. J. Liu, W. Reisig, C. J. Jiang, and M. C. Zhou, "A branching-process-based method to check soundness of workflow systems," *IEEE Access*, vol. 4, pp. 4104−4118, Jan. 2016.

[32] W. Vogler, A. L. Semenov, and A. Yakovlev, "Unfolding and finite prefix for nets with read arcs," in *Proc. 9th Int. Conf. Concurrency Theory*, London, UK, 1998, pp. 501−516.

[33] S. Christensen and N. D. Hansen, "Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs," in *Proc. 14th Int. Conf. Application and Theory of Petri Nets*, London, UK, vol. 691, pp. 186−205, 1993.

[34] S. Bandinelli and A. Fuggetta, "Computational reflection in software process modeling: the SLANG approach," in *Proc. 15th Int. Conf. Software Engineering*, Los Alamitos, CA, USA, 1993, pp. 144−154.

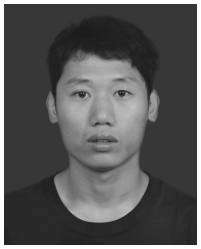
[35] L. Ma and J. P. Tsai, "Formal modeling and analysis of a secure mobile-agent system," *IEEE Trans. Syst. Man Cybern. A Syst. Hum.*, vol. 38, no. 1, pp. 180−196, Jan. 2008.

[36] J. Desel, V. Milijic, and C. Neumair, "Model validation in controller design," in *Lectures on Concurrency and Petri Nets*, J. Desel, W. Reisig, and G. Rozenberg, Eds. Berlin Heidelberg, Germany: Springer, 2004, pp. 467−495.

[37] C. X. Xu, W. L. Qu, H. P. Wang, Z. Z. Wang, and X. J. Ban, "A petri net-based method for data validation of web services composition," in *Proc. IEEE 34th Annu. Computer Software and Applications Conf.*, Washington, DC, USA, 2010, pp. 468−476.

[38] B. Bonet, P. Haslum, V. Khomenko, S. Thiëbaux, and W. Vogler, "Recent advances in unfolding technique," *Theor. Comput. Sci.*, vol. 551, pp. 84−101, Sep. 2014.

[39] J. Esparza, S. Römer, and W. Vogler, "An improvement of McMillanś unfolding algorithm," *Formal Methods Syst. Des.*, vol. 20, no. 3, pp. 285−310, May 2002.



Dongming Xiang graduated from University of Jinan, China, in 2010. He received the M.S. degree from this university in 2013. He is currently working toward the Ph.D. degree in the Department of Computer Science and Technology, Tongji University. His research interests include model checking, Petri net, business process management, and service computing.

Guanjun Liu (M'2016) received the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2011. He was a Postdoctoral Research Fellow at Singapore University of Technology and Design, Singapore, from 2011 to 2013. He worked at Humboldt-University zu Berlin, Germany, from 2013 to 2014 as a Postdoctoral Research Fellow supported by the Alexander von Humboldt Foundation. He is currently an Associate Professor with the Department of Computer Science and Technology, Tongji University. He has published 50+ papers. His research interests include Petri net theory, model checking, web service, workflow, discrete event systems, and information security.

Chungang Yan received the Ph.D. degree from Tongji University, Shanghai, China, in 2006. She is currently a Professor with the Department of Computer Science and Technology, Tongji University, Shanghai, China. Her current research interests include concurrent model and algorithm, Petri net theory, formal verification of software, and trusty theory on software process. She has published more than 30 papers in domestic and international academic journals and conference proceedings.

Changjun Jiang received the Ph.D. degree from the Institute of Automation, Chinese Academy of Science, Beijing, China, in 1995. He is currently the leader of the Key Laboratory of the Ministry of Education for Embedded System and Service Computing, Tongji University, Shanghai, China. He is an IET Fellow and an Honorary Professor with Brunel University London. He has published more than 300 papers in journals and conference proceedings, including *Chinese Science*, *IEEE Transactions on Robotics and Automation*, and *IEEE Transactions on Fuzzy Systems*. He has led over 30 projects supported by the National Natural Science Foundation of China, the National High Technology Research and Development Program of China, and the National Basic Research Developing Program of China. His research interests include concurrency theory, Petri nets, formal verification of software, cluster, grid technology, intelligent transportation systems, and service-oriented computing. Prof. Jiang has been the recipient of one international prize and seven prizes in the field of science and technology.