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], ame 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 detected.
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].

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.

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).

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 \( F_c \cap F_d = \emptyset \).

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 \( C = \{p_1, \ldots, p_6\} \), \( V = \{v_1, v_2\} \), the arc from \( p_1 \) to \( t_1 \) is a control arc, the dashed line from \( v_2 \) to \( t_3 \) 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.](image)

The following notations are used in this paper:

1. \( c^t = \{p \mid p \in C \wedge (p, t) \in F_c\} \) is a control pre-set of \( t \);
2. \( v^t = \{v \mid v \in V \wedge (v, t) \in F_r\} \) is a control post-set of \( t \);
3. \( c^t = \{v \mid v \in V \wedge (v, t) \in F_c\} \) is a read-set of \( t \);
4. \( t^w = \{v \mid v \in V \wedge (t, v) \in F_r\} \) is a write-set of \( t \);
5. \( d^t = \{v \mid v \in V \wedge (v, t) \in F_d\} \) is a delete-set of \( t \);
6. \( \text{PreD}(t) = d^t + c^t \);
7. \( \text{PostD}(t) = t^w \).

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 \( M \models c[t] \), if \( \forall p \in c^t : M(p) > 0 \). Otherwise, it is not control-enabled and denoted by \( \neg M \models c[t] \);
2. \( t \in T \) is data-enabled at a marking \( M \), which is denoted by \( M \models v[t] \), if \( \forall v \in \text{PreD}(t) : M(v) > 0 \). Otherwise, it is not data-enabled and denoted by \( \neg M \models v[t] \).

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

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

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 \( \sigma = t_1t_2 \cdots t_n \) such that \( M(t_1)M_1(t_2)M_2 \cdots M_{n-1}(t_n)M' \), i.e., \( M[\sigma]M' \). The set of markings reachable from \( M \) is denoted by \( R(M) \). Notice that \( t \in \sigma \) means that \( t \) occurs in \( \sigma \).

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 \( M \models c[t_1] \wedge M \models c[t_2] \wedge (c^t_1 \cap c^t_2 \neq \emptyset) \); or
2. a concurrency relation at \( M \), which is denoted by \( t_1 \parallel M t_2 \), if \( M \models c[t_1] \wedge M \models c[t_2] \wedge (c^t_1 \cap c^t_2 = \emptyset) \).

For example, \( t_1 \) and \( t_2 \) in Fig. 5(a) are in a conflict relation at the initial marking \( M_0 = [p_1 + p_2 + p_3;] \) since they satisfy \( M_0[t_1], M_0[t_2] \) 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 \( M_0' = [p_1' + p_2' + p_3';] \) because they satisfy \( M_0'[t_1'], M_0'[t_2'] \) and \( c^t_1' \cap c^t_2' = \emptyset \).

![Fig. 5: (a) Conflict relation. (b) Concurrency relation.](image)

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

**Definition 5 (Missing Data):** $\Sigma$ has an error of missing data if $\exists t \in T, \exists M \in R(M_0): M[t] \wedge -M_{0}t$.

Missing data occurs when a process is reading or deleting some data, but this data is not existing at this time. In Fig. 4, at reachable marking $t_4$ is writing or deleting this data. In Fig. 4, 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):** $\Sigma$ has an error of inconsistent data if $\exists M \in R(M_0), \exists t_1, t_2 \in T: (t_1 \parallel M_{t2}) \wedge (t_1 \cup d_{t1} \cup t_{w1}) \cap (d_{t2} \cup t_{w2}) \neq \emptyset$.

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):** $\Sigma$ has an error of redundant data if one of the following conditions holds:

1) $\exists M_1, M_2 \in R(M_0), \exists t \in T, \exists v \in V: M_1[t_1]M_2 \wedge v \in t^{\nu}_{w1} \wedge (\forall M_3 \in R(M_2), \forall t_2 \in T: M_3[t_2] \rightarrow v \notin t_{w2})$;

2) $\exists M_1, M_2 \in R(M_0), \exists t_1, t_2 \in T, \exists \sigma \in T^*, \exists v \in V: M_1[t_1\sigma]M_2[t_2] \wedge v \in t^{\nu}_{w1} \wedge v \in d_{t2} \wedge (\forall t_3 \in \sigma: v \notin t_{w3})$.

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 $t_4$. Therefore, there is an error of redundant data.

**Definition 8 (Lost Data):** $\Sigma$ has an error of lost data if $\exists M_1, M_2 \in R(M_0), \exists t_1, t_2 \in T, \exists \sigma \in T^*, \exists v \in V: M_1[t_1\sigma]M_2[t_2] \wedge v \in t^{\nu}_{w1} \wedge \forall t_3 \in \sigma: v \notin t_{w3}.$

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 $\Sigma = (C \cup V, T, F_c \cup F_u \cup F_d, M_0)$ be a PN-DO. $RG(\Sigma) = (R(M_0), E, t)$ is the reachability graph of $\Sigma$, where

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

2) $t: E \rightarrow T \times 2^V \times 2^V \times 2^V$ such that $t(M_i, M_j) = (t, t', t_\nu, t_\sigma)$ if $(M_i, M_j) \in E$ and $M_i[t]M_j$.

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 $\Sigma = (C \cup V, T, F_c \cup F_u \cup F_d, M_0)$ 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

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

A pseudo reachable marking is yielded by operating control places only. We denote $M[ct]M'$ 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 \( \Sigma = (C \cup V, T, F_r \cup F_t \cup F_w \cup F_d, M_0) \) 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 \( \Sigma \), where

1. \( (R(M_0), E, \ell) \) is the reachability graph of \( \Sigma \);
2. \( R'(M_0) \) is the set of all pseudo reachable markings;
3. \( E' = \{(M_i, M_j) | M_i \in R(M_0) \wedge M_j \in R'(M_0) \wedge \exists t \in T: M_i[t] M_j \} \); and
4. \( \ell' : E' \rightarrow T \times 2^V \times 2^V \times 2^V \) such that \( \ell'(M_i, M_j) = (t, r, t', w, d) \) if \( (M_i, M_j) \in E' \) and \( M_i[t] M_j \).

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

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 \( \Sigma = (C \cup V, T, F_r \cup F_t \cup F_w \cup F_d, M_0) \).
Output: All data-flow errors.

1) Initialize
   
   1: Marking set \( MD = \emptyset \); /* The detected markings. */
   2: 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

3: for each \( M \in R(M_0) \) such that \( M \notin MD \) do
   4: \( MD.add(M) \);
   5: \( DetectISData(M, ConHash, ERG(\Sigma)) \);
   6: \( E_1 = \{(M, M') | (M, M') \in E \} \);
7: for each \( (M, M') \in E_1 \) do
8: if \( (M, M') \in E' \) then
9: \( print Missing Data; \)
10: else if \( M[t] M' \) and \( t_m \neq \emptyset \) then
11:   for each \( v \in t_m \) do
12:     \( MT = \emptyset \); /* The traversed markings */
13:     Set \( M_r = M_w = M_d = \emptyset \);
14:     \( FindRWD(v, M', ERG(\Sigma), MT) \);
15:     /* As shown in Procedure_2, it is used to compute \( M_r, M_w \) and \( M_d \) */
16:     if \( M_w \neq \emptyset \) then
17:       \( print Lost Data; \)
18:     end if
19:     if \( M_d \neq \emptyset \) then
20:       \( print Redundant Data; \)
21:     end if
22:     if \( M_r = M_w = M_d = \emptyset \) then
23:       \( print Redundant Data; \)
24:     end if
25:     end for
26: end if
27: end for
28: end for
29: end for

Procedure 1: DetectISData \( (M, ConHash, ERG(\Sigma)) \)

/* Detect inconsistent data */
1: \( T^{oo} = ConHash.get(M) \);
2: /* Get all pairs of concurrent transitions at \( M \) */
3: if \( T^{oo} \neq \emptyset \) then
4:   for each \( (t_1, t_2) \in T_c \) do
5:     if \( (t_1 \cup t_1 \cup d_1) \cap (t_2 \cup d_2) \neq \emptyset \) then
6:       print Inconsistent Data between \( t_1 \) and \( t_2 \);
7:     end if
8: end for
9: end if

Procedure 2: FindRWD \( (v, M', ERG(\Sigma), MT) \)

/* Traverse all markings reachable from \( M' \), and obtain three reachable marking sets \( M_r, M_w, M_d \); \( M_r \) (resp. \( M_w, M_d \)) is the set of reachable markings at which there is a read (resp. write, delete) operation. */
1: if \( M' \notin MT \) then
2:   \( MT.add(M') \);
3: \( E_2 = \{(M', M'') | (M', M'') \in E \} \);
4: if \( E_2 \neq \emptyset \) then
5:   for each \( (M', M'') \in E_2 \) do
6:     if \( M'[t'] M'' \) then
7:       if \( v \in t' \) then
8:         \( M_r.add(M') \);
9:       end if
10:      if \( v \in t' \) then
11:        \( M_w.add(M') \);
12:      end if
13:     if \( v \in d't' \) then
14:       \( M_d.add(M') \);
15:     end if
16:     if \( v \notin t' \cup t' \cup d't' \) then
17:       \( FindRWD(v, M'', ERG(\Sigma), MT) \);
18:     end if
19:   end if
20: end for
21: end if
22: 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.,

\[
ConHash.get(M) = \{(t_i, t_j) | t_i \parallel M t_j \}.
\]

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 $\text{ERG}(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the ERG of a PN-DO $\Sigma$. A prefix of $\text{ERG}(\Sigma)$ 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 $\text{ERG}(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the ERG of a PN-DO $\Sigma$, $G_1 = (R_1, E_1, \ell_1)$ be a prefix of $\text{ERG}(\Sigma)$, and $M_1$ be in $R_1$ be a node of the prefix. $(e, M')$ is a possible extension of the prefix $M_1$ if $e = (M, M') \land e \notin E_1 \land e \in E \cup E'$. A new prefix $G_2 = (R_2, E_2, \ell_2)$ of $\text{ERG}(\Sigma)$ 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 $\text{ERG}(\Sigma) = (R(M_0) \cup R'(M_0), E \cup E', \ell \cup \ell')$ be the ERG of a PN-DO, and $G_1 = (R_1, E_1, \ell_1)$ be a prefix of $\text{ERG}(\Sigma)$, $M \in R_1$, and $(e_1, M_1)$ and $(e_2, M_2)$ be two possible extensions of the prefix at $M$ such that $\ell_1(e_1) = \{t_1, t_1, t'_1, d_1\} \land \ell_1(e_2) = \{t_2, t_2, t'_2, d_2\}$ and $t_1 \parallel M | t_2$. 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 $t_1 \cap y \neq \emptyset$ or $d \cap y \neq \emptyset$.

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 $\Sigma = (C \cup V, T, F_e \cup F_r \cup F_w \cup F_d, M_0)$ and a marking $M$.

**Output:** An extension set $S$.

1: Initialize: A transition pairs $P = \emptyset$, and a transition set $T_S = \emptyset$.
2: $T^* = \{t | M[t], t \in T\}$.
3: Get concurrent transition pair $T^c$ and conflict transition pair $T^\#$, where

$T^c = \{(t_1,t_2)| t_1 | M[t_1], t_2 | M[t_2], t_1 \cap t_2 \neq \emptyset\}$

$T^\# = \{(t_1,t_2)| t_1 \parallel t_2 | M[t_1], t_2 | M[t_2]\}$.

4: for all $(t_1,t_2) \in T^c$ 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.,

$S_1 = \{t | t + M[t_1], t \in T^c, t \in T\}$

$S_2 = \{t | t + M[t_2], t \in T^c, t \in T\}$.

6: Boolean $se = \text{TRUE};$ /* 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
XIANG et al.: DETECTING DATA-FLOW ERRORS BASED ON PETRI NETS WITH DATA OPERATIONS

8: \( P . add(x, y); \)
9: if there exists \((x, y)\) that does not satisfy selection criterion then
10: \( \text{se} = \text{FALSE}; \text{break}; \)
11: end if
12: end for
13: if \( \text{se} == \text{FALSE} \) then
14: \( T_S . add(S_1 \cup \{t_1\} \cup S_2 \cup \{t_1\}); \)
15: else
16: if \( |S_1| \leq |S_2| \) then
17: \( T_S . add(S_1 \cup \{t_1\}); \)
18: else
19: \( T_S . add(S_2 \cup \{t_2\}); \)
20: end if
21: end if
22: end for
23: if \( |T'| = 1 \) or \( T^{co} = \emptyset \) then
24: \( T_S . add(T'); \)
25: end if
26: for each \( t \in T_S \) do
27: if \( M[t].M_1 \) then
28: \( S . add((M, M_1), M_1); \)
29: end if
30: if \( \neg M[v].t \) and \( M[v].t.M_2 \) then
31: \( S . add((M, M_2), M_2); \)
32: end if
33: end for
34: 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.

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

Fig. 7: Different reduced ERGs.
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).](image)

TABLE I

<table>
<thead>
<tr>
<th>Data item</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₀</td>
<td>Email ID</td>
</tr>
<tr>
<td>T₁</td>
<td>Email message</td>
</tr>
<tr>
<td>T₂</td>
<td>Report</td>
</tr>
<tr>
<td>T₃</td>
<td>Employee ID</td>
</tr>
<tr>
<td>T₄</td>
<td>Topic/Subject name</td>
</tr>
<tr>
<td>T₅</td>
<td>GM feedback</td>
</tr>
<tr>
<td>T₆</td>
<td>Contact number</td>
</tr>
<tr>
<td>T₇</td>
<td>Publish in website/newsletter</td>
</tr>
<tr>
<td>T₈</td>
<td>DH guideline/feedback</td>
</tr>
<tr>
<td>T₉</td>
<td>Article No.</td>
</tr>
<tr>
<td>T₁₀</td>
<td>Editorial review</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Transition ID</th>
<th>Business activities</th>
<th>Input data (Read)</th>
<th>Output data (Write)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>Send email</td>
<td>v₀, v₁</td>
<td></td>
</tr>
<tr>
<td>t₂</td>
<td>Submit report</td>
<td>v₁, v₂, v₃, v₄, v₅, v₆</td>
<td></td>
</tr>
<tr>
<td>t₃</td>
<td>Review by GM</td>
<td>v₂, v₃</td>
<td>v₅, v₇</td>
</tr>
<tr>
<td>t₄</td>
<td>Review by DH</td>
<td>v₂, v₃, v₅, v₇</td>
<td>v₇</td>
</tr>
<tr>
<td>t₅</td>
<td>Re-submit report</td>
<td>v₂, v₅</td>
<td>v₂</td>
</tr>
<tr>
<td>t₆</td>
<td>Revise report</td>
<td>v₂, v₈</td>
<td>v₂</td>
</tr>
<tr>
<td>t₇</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>t₈</td>
<td>Editorial review</td>
<td>v₂, v₃, v₇</td>
<td>v₁₀</td>
</tr>
<tr>
<td>t₉</td>
<td>Archive</td>
<td>v₂, v₃, v₄</td>
<td>v₉</td>
</tr>
<tr>
<td>t₁₀</td>
<td>Catalogue</td>
<td>v₂</td>
<td>v₉</td>
</tr>
<tr>
<td>t₁₁, t₁₂</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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


### TABLE III

<table>
<thead>
<tr>
<th>Data-flow errors</th>
<th>Marking</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing data</td>
<td>$M_6$</td>
<td>$v_6$ is missing for $t_0$</td>
</tr>
<tr>
<td>Redundant data</td>
<td>$M_1$, $M_7$, $M_9$</td>
<td>$v_10$ is written by $t_2$ but never read again</td>
</tr>
<tr>
<td>Lost data</td>
<td>$M_{10}$, $M_{11}$, $M_{12}$</td>
<td>$t_9$ and $t_{10}$ overwrite some data into $v_9$ concurrently</td>
</tr>
<tr>
<td>Inconsistent data</td>
<td>$M_8$</td>
<td>$v_8$ is written by $t_2$ but never read again</td>
</tr>
</tbody>
</table>

**Fig. 9:** The ERG of $\Sigma$.

**Fig. 10:** The reduced ERG of $\Sigma$.  


