CHECKING DATA-FLOW ERRORS BASED ON THE GUARD-DRIVEN REACHABILITY GRAPH OF WFD-NET

Dongming Xiang
School of Information Science and Technology
Zhejiang Sci-Tech University
310018 Hangzhou, China
e-mail: flysky_xdm@163.com

Guanjun Liu
Department of Computer Science
Key Laboratory of Embedded System and Service Computing (MOE)
Tongji University
201804 Shanghai, China
e-mail: liuguanjun@tongji.edu.cn

Abstract. In order to guarantee the correctness of workflow systems, it is necessary to check their data-flow errors, e.g., missing data, inconsistent data, lost data and redundant data. The traditional Petri-net-based methods are usually based on the reachability graph. However, these methods have two flaws, i.e., the state space explosion and pseudo states. In order to solve these problems, we use WFD-nets to model workflow systems, and propose an algorithm for checking data-flow errors based on the guard-driven reachability graph (GRG) of WFD-net. Furthermore, a case study and some experiments are given to show the effectiveness and advantage of our method.

Keywords: Petri net, workflow system, data-flow errors, reachability graph

Mathematics Subject Classification 2010: 68-Q60
1 INTRODUCTION

Nowadays, workflow systems have been widely applied to our daily life, e.g., office automation (OA), medical treatment and electronic commerce, etc. In order to guarantee the correctness of workflow systems, we not only need to verify some properties and detect errors in the control-flows, but also model and analyze their data-flows. As we know, the control-flows focus on the partial orders of business activities, while the data-flows mostly include data elements, data operations (i.e., read, write and delete) and data conditions. The existing modeling and analysis methods of workflow systems are mainly concerned with the error detection of control-flows. In fact, data-flows are also greatly important in the design of workflow system. Once its activities conduct an improper operation on data-flows in business processes, some data-flow errors \[15, 19, 28\] easily take place, e.g., missing data, inconsistent data, lost data, redundant data and unsoundness. These errors can lead to some abnormal results, degrade the execution performance, and increase the maintenance cost, or even result in some insecurity problems, e.g., privacy disclosure, illegal user access, and fund loss.

There have been many studies on data-flows of workflow systems. Sadiq et al. \[19\] first proposed seven kinds of data-flow anomalies, but did not provide any detection methods. Sharma et al. \[21\] used BPMN (Business Process Modeling Notation) to model business processes and detected their data-flow errors. Guo et al. \[8\] solved the data exchange problems in the inter-organizational workflows. Sun et al. \[24\] calculated the dependence relationship of business processes in a UML (Unified Modeling Language) diagram, and detected errors in each process instance according to its data association. This work was further generalized in \[15\], where a systematic graph traversal approach was proposed to detect data-flow errors.

Some Petri-net-based methods are also proposed to detect data-flow errors. A Dual Flow Net (DFN) \[27\] was used to model the control- and data-flows in an embedded system. Based on the work in \[21\], Awad et al. \[2\] mapped BPMN into Petri net, and then detected and repaired its errors. In order to model the concurrent read operation, contextual net \[3, 16\] was proposed, and its unfolding technique was utilized to generate the minimal test suites for multi-threaded programs \[13, 12\]. Based on the contextual net, PN-DO (Petri net with data operation) \[31\] was given to detect data-flow errors of workflow systems. All of these methods have an advantage of a great capability to specify parallelism, concurrency and synchronization \[11, 17\]. However, the explicitly modeling of read/write arcs can increase the scales and complexity of Petri nets. By comparison, WFD-net is a workflow net \[18\] (a special Petri net) extended with conceptual data operations. Its transitions are labeled by read, write, delete or guard\[1\] functions \[10, 22\]. Naturally, the scale of WFD-net is much more smaller than the Petri nets with data operation arcs, e.g., contextual net and PN-DO.

\[1\] A guard is a Boolean expression which is formed by some data elements and predicates.
WFD-net has been widely used to check soundness [22], completion requirements [25] and data consistencies [34]. These verification/analysis methods are usually based on the classical reachability graphs [22, 32] of WFD-nets. However, they easily suffer from the state space explosion and pseudo states. On the one hand, a state may have an exponential number of successor states in a reachability graph since every possible value of a guard is considered. On the other hand, the logical relation (e.g., exclusion property) between guards likely generate some pseudo states. In order to solve these problems, we proposed the guard-driven reachability graph (GRG) of a WFD-net in the previous work [30].

In this paper, we use GRG to check data-flow errors in a workflow system, including missing data, inconsistent data, redundant data and lost data. We first define these data-flow errors in a WFD-net, and propose an algorithm for checking them. Furthermore, a case study and some experiments are given to illustrate the effectiveness and advantage of our algorithm.

The rest of this paper is organized as follows. Section 2 presents some basic notations. Section 3 proposes an algorithm to check data-flow errors based on the GRG of WFD-nets. Section 4 gives a case study. Section 5 conducts a group of experiments. The last section sums up the whole work.

2 BASIC NOTATIONS

2.1 WF-Net

A net is a triple \( N = (P, T, F) \), where \( P \) and \( T \) are two disjoint and finite sets that are respectively called place set and transition set, and \( F \subseteq (P \times T) \cup (T \times P) \) is a flow relation. A net \( N \) with an initial marking \( m_0 \) is called a Petri net or net system [14, 33], and denoted as \( \Sigma = (N, m_0) \). For each node \( x \in P \cup T \), its preset and posset are denoted by \( \text{•}x = \{ y | (y, x) \in F \} \) and \( x^* = \{ y | (x, y) \in F \} \), respectively.

Given a net \( N = (P, T, F) \), a transition \( t \in T \) is enabled at a marking \( m \) if \( \forall p \in P : p \in t^* \Rightarrow m(t) \geq 1 \), which is denoted by \( m[t] \). After firing an enabled transition \( t \) at \( m \), a new marking \( m' \) is generated, which is denoted as \( m[t]m' \), where \( \forall p \in P : \)

\[
m'(p) = \begin{cases} 
m(p) - 1, & \text{if } p \in t^* - t^*; \\
m(p) + 1, & \text{if } p \in t^* - t; \\
m(p), & \text{otherwise}. 
\end{cases}
\]

1. there is one source place \( i \) and one sink place \( o \) in \( P \) such that \( i^* = \emptyset \land o^* = \emptyset \); and
2. \( \forall x \in P \cup T : (i, x) \in F^* \) and \( (x, o) \in F^* \) where \( F^* \) is the reflexive-transitive closure of \( F \).
As a particular class of Petri net, WF-net has been widely used to model and verify workflow systems [1].

2.2 WFD-Net

![WFD-net and data-flow errors](image)

Based on WF-net, workflow net with data (WFD-net) [25, 26] is proposed to model the control-flows and data-flows of a workflow system. That is, some data elements, data operations and guards are added into WF-net.

In a WFD-net, $D$ is a finite set of data elements, and $G$ is a set of guards over $D$. $\text{Var}(g)$ denotes the set of variables in the guard $g \in G$. We assume that $\forall g \in G : d \in \text{Var}(g) \Rightarrow d \in D$. Two different guards $g_1$ and $g_2$ are exclusive if $g_1 = \neg g_2$ and $g_2 = \neg g_1$, which is denoted by $g_1 \otimes g_2$. In other words, $g_1$ is TRUE iff $g_2$ is FALSE, and vice versa. For example, given two guards $g_1$ and $g_2$, if $g_1(v_1) : v_1 > 0$ and $g_2(v_1) : v_1 \leq 0$, then they are exclusive and denoted by $g_1 \otimes g_2$.

**Definition 2** (Workflow net with data [22]). A 9-tuple $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ is a WFD-net if

1. $(P, T, F)$ is a WF-net;
2. $D$ is a finite set of data elements;
3. $G$ is a finite set of guards;
4. $Rd: T \rightarrow 2^D$ is a label function of reading data;
5. $Wr: T \rightarrow 2^D$ is a label function of writing data;
6. **De:** \( T \rightarrow 2^D \) is a label function of deleting data; and
7. **Guard:** \( T \rightarrow G \) is a label function of assigning a guard in \( G \) to each transition.

For example, Figure 1 a) is a WFD-net, where \( D = \{v_1, v_2, v_3\} \), \( G = \{g_1(v_1), \overline{g_1(v_1)}\} \), Guard\((t_1) = g_1(v_1)\), Guard\((t_3) = \overline{g_1(v_1)}\), Rd\((t_1) = \{v_1\}\), Wr\((t_2) = \{v_2\}\) and De\((t_1) = \{v_1\}\). Moreover, \( g_1 \otimes \overline{g_1} \) and Var\((g_1) = \operatorname{Var}(\overline{g_1}) = \{v_1\}\).

As for a WFD-net, a state is generally called a **weak configuration**, and it includes a marking and the evaluations of data and guards.

**Definition 3** (Weak configuration). Let \( N = (P, T, F, D, G, Rd, Wr, De, Guard) \) be a WFD-net. \( c = (m, \sigma, \eta) \) is a weak configuration, where

1. \( m \) is a marking of \( (P, T, F) \);
2. \( \sigma : D \rightarrow \{\top, \bot\} \) assigns a defined value (\( \top \)) or an undefined value (\( \bot \)) to each data element; and
3. \( \eta : G \rightarrow \{\text{TRUE}, \text{FALSE}, \bot, \top\} \) assigns \( \text{TRUE}, \text{FALSE}, \) an undefined value (\( \bot \)) or a defined value (\( \top \)) to each guard.

In a weak configuration, \( (\sigma, \eta) \) represents a data state. Besides, we use the guards labeled with \( \ast \) to represent their defined values (\( \top \)). As shown in Figure 1 b) \( c_1 = \langle m_1, \sigma_1, \eta_1 \rangle = \langle [p_1 + p_2], \{v_1\}, \{g_1\} \rangle \) is a weak configuration of the WFD-net in Figure 1 a) where \( \sigma_1(v_1) = \top \) and \( \ast g_1 \) represents that \( \eta_1(g_1) = \top \).

Given the definition of WFD-net, we discuss its weak firing rules of an enabled transition at a weak configuration.

**Definition 4** (Weak enabling/firing rules). Let \( N = (P, T, F, D, G, Rd, Wr, De, Guard) \) be a WFD-net. \( t \in T \) is enabled at a weak configuration \( c = (m, \sigma, \eta) \), which is denoted by \( c[t] \), if

1. \( m[t] \);
2. \( \forall d \in Rd(t) : \sigma(d) = \top \); and
3. \( \forall d \in \operatorname{Var}(\text{Guard}(t)) : \sigma(d) \neq \bot \) and \( \eta(\text{Guard}(t)) \in \{\text{TRUE}, \top\} \).

After firing an enabled transition \( t \) at \( c \), a weak configuration \( c' = (m', \sigma', \eta') \) is generated, where

1. \( m[t]m' \);
2. \( \forall d \in De(t) : \sigma'(d) = \bot \);
3. \( \forall d \in Wr(t) \setminus De(t) : \sigma'(d) = \top \);
4. \( \forall d \in D \setminus (De(t) \cup Wr(t)) : \sigma'(d) = \sigma(d) \);
5. \( \exists g \in \text{Guard}(t) : \operatorname{Var}(g) \cap Wr(t) = \emptyset \Rightarrow \eta'(g) = \text{TRUE} \); and
6. \( \forall g \in G, \forall d \in \operatorname{Var}(g) : (\sigma'(d) = \top \Rightarrow \eta'(g) \land ((g \notin \text{Guard}(t) \land \operatorname{Var}(g) \cap Wr(t) = \emptyset) \Rightarrow \eta'(g) = \eta(g)) \).

It is denoted as \( c[t]c' \).
For example, the transition $t_0$ in Figure 1a is enabled at the initial weak configuration $c_0$ and $c_0[t_0]c_1$, where $c_0 = \langle [i], -, - \rangle$ and $c_1 = \langle [p_1 + p_2], \{v_1\}, \{*g_1\} \rangle$. After firing the transition $t_0$ and writing a new value into the data $v_1$, the evaluations of $g_1$ is not definite because $v_1$ is associated with this guard. We assign a defined value ($\top$) to this guard in $c_1$. Thus, firing $t_0$ generates one unique weak configuration in Figure 1b.

According to the enabling and firing rule of transitions, the may-reachability of WFD-net is defined as follows.

**Definition 5** (May-reachability [22]). Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net. $c_1$ and $c_2$ are two configurations.

1. There is a may-step from $c_1$ to $c_2$, denoted by $c_1 \rightarrow_{\text{may}} c_2$, if there is a transition $t \in T$ and a set of configurations $C$ such that: $c_1[t]C \land c_2 \in C$.
2. $c_2$ is may-reachable from $c_1$ if there exists a sequence of configurations $c^{(1)}, \ldots, c^{(n)}$ such that $c_1 \rightarrow_{\text{may}} c^{(1)} \rightarrow_{\text{may}} \cdots \rightarrow_{\text{may}} c^{(n)} \rightarrow_{\text{may}} c_2$. It is denoted as $c_1 \rightarrow_{\text{may}}^* c_2$.

The set of may-reachable configurations from $c$ is denoted by $R(c)$. For example, there is a may-step from $c_0$ to $c_1$ in Figure 1b and the configuration $c_6 = \langle [p_3 + p_4], \{v_1, v_2\}, \{g_1\} \rangle$ is may-reachable from $c_0$.

### 2.3 Guard-Driven Reachability Graph

Although the classical reachability graph [22] is a fundamental method of analyzing and verifying a WFD-net, it easily suffers from the problems of state space explosion and pseudo configurations due to its guard evaluations and their exclusive relations [30]. Hence, we propose the *guard-driven reachability graph* (GRG) based on the weak configurations and the weak firing rules.

**Definition 6** (Guard-driven reachability graph, GRG). Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net and $c_0$ be its initial weak configuration. $\text{GRG}(N) = (V^+, E^+, \ell^+)$ is the guard-driven reachability graph of $N$, where

1. $V^+ = R(c_0)$;
2. $E^+ = \{(c, c') \mid c, c' \in R(c_0) \land \exists t \in T : c[t]c'\}$; and
3. $\ell^+ : E^+ \rightarrow T \times G$ such that $\ell^+(c, c') = \langle t, \text{Guard}(t) \rangle$ if $(c, c') \in E^+$ and $c[t]c'$.

For example, Figure 1b is the GRG of the WFD-net in Figure 1a where $c_0, c_1 \in V^+$, $e_0 = (c_0, c_1) \in E^+$ and $\ell^+(e_0) = \langle t_0, \text{TRUE} \rangle$.

In the guard-driven reachability graph of a WFD-net, the guard as the condition of enabling a transition determines the unique successor state when firing the

\[\text{If Guard}(t_0) = \emptyset, \text{we use } \langle t_0, \text{TRUE} \rangle \text{ to represent this case.}\]
3 DATA-FLOW ERRORS DETECTION METHOD BASED ON THE GRG OF WFD-NET

3.1 Data-Flow Errors

Data-flow errors are caused by improper data operations in workflow systems, which mainly include missing data, redundant data, lost data, and inconsistent data. We first define these data-flow errors in a WFD-net.

1. Missing Data

Missing data occurs when a business process of workflow systems is reading or deleting some data, but this data is not existing at this time.

**Definition 7** (Missing Data). A WFD-net $N$ with an initial weak-configuration $c_0$ has an error of missing data if $\exists t \in T, \exists c \in R(c_0) : c = \langle m, \sigma, \eta \rangle \land \neg c(t \land m[t]) \land \exists d \in Rd(t) \cup De(t) : \sigma(d) = \bot$.

For example, the transition $t_5$ in Figure 1 is not enabled at the reachable weak-configuration $[p_3 + p_4; v_2; g_1]$ because it cannot read the data from $v_3$ since this data has never been written into any values. At this time, missing data occurs.

2. Inconsistent Data

The error of inconsistent data usually occurs in a concurrent workflow system when one business process is reading or writing or deleting some data, but another process is concurrently writing or deleting this data. Notice that two transitions $t_1$ and $t_2$ in a bounded WFD-net are concurrent at a weak-configuration $c = \langle m, \sigma, \eta \rangle$, if they satisfy that $c[\cdot t_1] \land c[\cdot t_2] \land (\cdot t_1 \cap \cdot t_2 = \emptyset \lor \forall p \in \cdot t_1 \cap \cdot t_2 : m(p) \geq 2)$ [29]. This is denoted by $t_1 \parallel t_2$.

**Definition 8** (Inconsistent Data). The error of inconsistent data takes place in a WFD-net if two concurrent transitions $t_1$ and $t_2$ satisfy $\exists c \in R(c_0) : (Rd(t_1) \cup Wr(t_1) \cup De(t_1)) \cap (Wr(t_2) \cup De(t_2)) \neq \emptyset$.

For example, two transitions $t_2$ and $t_4$ in Figure 1 are concurrently writing into the data $v_2$ at the reachable weak-configuration $[p_1 + p_2; v_1; *g_1]$. At this time, inconsistent data occurs.

3. Redundant Data

Redundant data occurs if a data is never read before it is deleted or the business process terminates.
Definition 9 (Redundant Data). Σ has an error of redundant data if one of the following two conditions holds:

1. \( \exists c_1, c_2 \in R(c_0), \exists t_1 \in T, \exists v \in D : c_1[t_1]c_2 \land v \in Wr(t_1) \land (\forall c_3 \in R(c_2), \forall t_2 \in T : c_3[t_2] \rightarrow v \notin Rd(t_2)) \);
2. \( \exists c_1, c_2 \in R(c_0), \exists t_1, t_2 \in T, \exists \sigma \in T^*, \exists v \in D : c_1[t_1\sigma]c_2[t_2] \land v \in Wr(t_1) \land v \in De(t_2) \land (\forall t_3 \in \sigma : v \notin Rd(t_3)) \).}

For example, the transition \( t_2 \) in Figure 1 is to overwrite the data \( v_2 \) at the reachable weak-configuration \([p_1 + p_2; v_1; *g_1]\). But the data has never been read until the business process terminates. Therefore, there is an error of redundant data.

4. Lost Data

Lost data means that once a data element is written into a value by a transition, it will never be read before it is written again by some follow-up transitions. In other word, the first value of this data element cannot be referenced again by other activities.

Definition 10 (Lost Data). Σ has an error of lost data if \( \exists c_1, c_2 \in R(c_0), \exists t_1, t_2 \in T, \exists \sigma \in T^*, \exists v \in V : c_1[t_1\sigma]c_2[t_2] \land v \in Wr(t_1) \cap Wr(t_2) \land (\forall t_3 \in \sigma : v \notin Rd(t_3)) \).

For example, the transition \( t_0 \) in Figure 1 writes a data value into \( v_1 \) at the initial weak-configuration. But this data is never be read before it is overwritten by \( t_3 \). Therefore, this is an error of lost data.

3.2 The Algorithm for Checking Data-Flow Errors Based on GRG

In order to check the above data-flow errors in a workflow system, we propose an algorithm based on GRG, which is shown in Algorithm 1.

- According to the definition of missing data, we can easily check this data-flow error by traversing each weak-configuration.
- At a weak-configuration, if two concurrent transitions are concurrently conducting some data operations on a data element, we can find out an error of inconsistent data.
- If an enabled transition is to write some value into a data element at a weak-configuration, we can traverse all successors of this weak-configuration. Then, the weak-configurations related to the operations on this data are obtained by the function
  \( \text{FindRWDConfigs}(v, c', GRG(\Sigma), CT) \).

That is, we traverse all weak-configurations reachable from \( c' \), and obtain three reachable weak-configuration sets \( c_r, c_w \) and \( c_d \), where \( c_r \) (resp. \( c_w, c_d \)) is the set
Checking Data-Flow Errors Based on GRG of WFD-Net

Procedure 1 \textbf{FindRWDConfigs}(v, c', GRG(\Sigma), CT)

1: \textbf{if} \ c' \not\in CT \ \textbf{then}
2: \hspace{1em} CT.add(c');
3: \hspace{1em} Get all edges from \ c', \ i.e., \ E_2 = \{(c', c'') \mid (c', c'') \in E\};
4: \hspace{1em} \textbf{if} \ E_2 \neq \emptyset \ \textbf{then}
5: \hspace{2em} \textbf{for each} \ (c', c'') \in E_2 \ \textbf{do}
6: \hspace{3em} \textbf{if} \ c'[t']c'' \ \textbf{or} \ c'[v,t']c'' \ \textbf{then}
7: \hspace{4em} \textbf{if} \ v \in Rd(t') \ \textbf{then}
8: \hspace{5em} c_r.add(c');
9: \hspace{4em} \textbf{end if}
10: \hspace{3em} \textbf{if} \ v \in Wr(t') \ \textbf{then}
11: \hspace{4em} c_w.add(c');
12: \hspace{4em} \textbf{end if}
13: \hspace{3em} \textbf{if} \ v \in De(t') \ \textbf{then}
14: \hspace{4em} c_d.add(c');
15: \hspace{4em} \textbf{end if}
16: \hspace{3em} \textbf{if} \ v \not\in Rd(t') \cup Er(t') \cup De(t') \ \textbf{then}
17: \hspace{4em} \textbf{FindRWD}(v, c', GRG(\Sigma), CT);
18: \hspace{3em} \textbf{end if}
19: \hspace{2em} \textbf{end if}
20: \hspace{1em} \textbf{end for}
21: \hspace{1em} \textbf{end if}
22: \hspace{1em} \textbf{end Procedure 1}

of reachable weak-configurations at which there is a read (resp. write, delete) operation. Finally, according to these weak-configurations, we determine whether there is an error of redundant data or lost data.

It is noted that GRG can effectively reduce the state space and avoid pseudo states since it fully considers the characters of guard functions. As a result, Algorithm 1 only needs to traverse a smaller state space to detect data-flow errors in comparison with the classical reachability graph in [22]. Moreover, it also prevents from some negative influence by pseudo states.

4 CASE STUDY

Our checking method for data-flow errors can be applied in the static program analysis. Figure 2 is a multi-thread program, and it is used to detect the errors of data inconsistency in the related work [9]. As our case study in this paper, we utilize it to check data-flow errors.

We first use a WFD-net to model this program, which is shown in Figure 3a). Tables 1 and 2 list the related transitions and guards. Meanwhile, if we respec-
Algorithm 1 Data-flow error detection algorithm

Require: A WFD-net \( N \)

Ensure: All data-flow errors.

1: Initialize \( C^\# = \emptyset \); /* The detected weak-configurations. */
2: Construct a GRG of \( N \), i.e., \( GRG(N) = (V^+, E^+, \ell^+) \).
3: for each \( c \in R(c_0) \) such that \( c \notin C^\# \) do
4: \( C^\#.add(c); \)
5: if \( \exists t \in T : \neg c[t \wedge m[t] \wedge \exists d \in Rd(t) \cup De(t) : \sigma(d) = \bot \) then
6: print Missing data;
7: end if
8: if \( \exists t_1, t_2 \in T : t_1|t_2 \wedge (Rd(t_1) \cup Wr(t_1) \cup De(t_1)) \cap (Wr(t_2) \cup De(t_2)) \neq \emptyset \) then
9: print Inconsistent Data between \( t_1 \) and \( t_2 \);
10: end if
11: Get all edges from \( c \), i.e., \( E_1 = \{(c, c') \mid (c, c') \in E^+\}; \)
12: for each \( (c, c') \in E_1 \) do
13: if \( c[t]c' \) and \( Wr(t) \neq \emptyset \) then
14: for each \( v \in Wr(t) \) do
15: \( CT = \emptyset \); /* The traversed weak-configurations */
16: Set \( c_r = c_w = c_d = \emptyset \);
17: FindRWDCfgs\( (v, c', GRG(\Sigma), CT) \);
18: /* As shown in Procedure 1, it is used to compute \( c_r, c_w \) and \( c_d */
19: if \( c_w \neq \emptyset \) then
20: print Lost Data;
21: end if
22: if \( c_d \neq \emptyset \) then
23: print Redundant Data;
24: end if
25: if \( c_r = c_w = c_d = \emptyset \) then
26: print Redundant Data;
27: end if
28: end for
29: end if
30: end if
31: end for
32: end if
33: end for
34: end for
Checking Data-Flow Errors Based on GRG of WFD-Net

Figure 2. Pseudo-codes of a multi-thread program

Table 1. Program codes and data operations

<table>
<thead>
<tr>
<th>Transition ID</th>
<th>Codes</th>
<th>Read Data</th>
<th>Write Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$x = y = 0$</td>
<td>–</td>
<td>$x, y$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$a = x$</td>
<td>$x$</td>
<td>$a$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$b = y$</td>
<td>$y$</td>
<td>$b$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$x = 1$</td>
<td>–</td>
<td>$x$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$y = 2$</td>
<td>–</td>
<td>$y$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>if($y &gt; 0$)</td>
<td>$y$</td>
<td>–</td>
</tr>
<tr>
<td>$t_6$</td>
<td>if($x &gt; 0$)</td>
<td>$x$</td>
<td>–</td>
</tr>
<tr>
<td>$t_7$</td>
<td>$y = a + 1$</td>
<td>$a$</td>
<td>$y$</td>
</tr>
<tr>
<td>$t_8$</td>
<td>$y = 0$</td>
<td>–</td>
<td>$y$</td>
</tr>
<tr>
<td>$t_9$</td>
<td>$x = b + 2$</td>
<td>$b$</td>
<td>$x$</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td>$x = 1$</td>
<td>–</td>
<td>$x$</td>
</tr>
<tr>
<td>$t_{11}$</td>
<td>$x = a + 1$</td>
<td>$a$</td>
<td>$x$</td>
</tr>
<tr>
<td>$t_{12}$</td>
<td>$x = 0$</td>
<td>–</td>
<td>$x$</td>
</tr>
<tr>
<td>$t_{13}$</td>
<td>$y = b + 2$</td>
<td>$b$</td>
<td>$y$</td>
</tr>
<tr>
<td>$t_{14}$</td>
<td>$y = 1$</td>
<td>–</td>
<td>$y$</td>
</tr>
<tr>
<td>$t_{15}$</td>
<td>assert1($x == y$)</td>
<td>$x, y$</td>
<td>–</td>
</tr>
<tr>
<td>$t_{16}$</td>
<td>assert2($x == y$)</td>
<td>$x, y$</td>
<td>–</td>
</tr>
<tr>
<td>$t_{17}$</td>
<td>end</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 2. Guards over transitions

<table>
<thead>
<tr>
<th>ID</th>
<th>Guard</th>
<th>ID</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_7$</td>
<td>$g_1(y) : y &gt; 0$</td>
<td>$t_8$</td>
<td>$-g_1(y) : y &lt;= 0$</td>
</tr>
<tr>
<td>$t_9$</td>
<td>$g_2(x) : x &gt; 0$</td>
<td>$t_{10}$</td>
<td>$-g_2(x) : x &lt;= 0$</td>
</tr>
</tbody>
</table>

Initially $x = y = 0$

<table>
<thead>
<tr>
<th>Thread T1</th>
<th>Thread T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 a = x$</td>
<td>$10 b = y$</td>
</tr>
<tr>
<td>$2 x = 1$</td>
<td>$11 y = 2$</td>
</tr>
<tr>
<td>$3 \text{if } (y &gt; 0)$</td>
<td>$12 \text{if } (x &gt; 0)$</td>
</tr>
<tr>
<td>$4 y = a + 1$</td>
<td>$13 x = b + 2$</td>
</tr>
<tr>
<td>$5 x = a + 1$</td>
<td>$14 y = b + 2$</td>
</tr>
<tr>
<td>$6 \text{else}$</td>
<td>$15 \text{else}$</td>
</tr>
<tr>
<td>$7 y = 0$</td>
<td>$16 x = 1$</td>
</tr>
<tr>
<td>$8 x = 0$</td>
<td>$17 y = 1$</td>
</tr>
<tr>
<td>$9 \text{assert}(x == y)$</td>
<td>$18 \text{assert}(x == y)$</td>
</tr>
</tbody>
</table>

Table 2. Guards over transitions

Table 3. Guards over transitions

We can use a PN-DO, a contextual net and a Petri net without read/write arcs to model this program, we can get a comparison between them, which are shown in Figure 3 and Table 3. It is clear that WFD-net needs a smaller space than others.
Figure 3. Different Petri nets for modeling the multi-thread program in Figure 2.
Checking Data-Flow Errors Based on GRG of WFD-Net

<table>
<thead>
<tr>
<th></th>
<th>Nodes (Place &amp; Transition)</th>
<th>Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFD-net</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>PN-DO</td>
<td>40</td>
<td>64</td>
</tr>
<tr>
<td>Contextual net</td>
<td>40</td>
<td>74</td>
</tr>
<tr>
<td>Petri net without read/write arcs</td>
<td>40</td>
<td>86</td>
</tr>
</tbody>
</table>

Table 3. The comparison between WFD-net and another Petri nets

Figure 4. a) GRG graph; b) the classical reachability graph
As for the WFD-net in Figure 3 a), we utilize our tool [29] (see Figure 5) to construct its guard-driven reachability graph (GRG) and classical reachability graph (CRG), which are respectively shown in Figures 4 a) and 4 b). Table 4 gives a comparison between them in terms of state space and running time. Obviously, the former has an advantage over the latter.

Based on the GRG in Figure 4 a), we check the data-flow errors according to Algorithm 1. Table 5 lists these results. From these results, we can see that there exist some errors of inconsistent data because $t_3$ and $t_9$ (resp. $t_{10}$) are concurrent at the weak-configuration $S_{13}$ and they satisfy $Wr(t_3) \cap Wr(t_9) = \{x\}$ (resp. $Wr(t_3) \cap Wr(t_{10}) = \{x\}$). The transitions $t_4$ and $t_7$ (resp. $t_8$) also suffer from the errors of inconsistent data. Moreover, $t_{11}$ (resp. $t_{12}$) overwrites some data into $x$ at weak-configurations $S^+_1$ after firing $t_3$, and $t_{13}$ (resp. $t_{14}$) overwrites some data into $y$ at weak-configurations $S^+_2$ after firing $t_4$. Therefore, there are some errors of lost data.
Data-Flow Errors | Weak-Configurations | Illustration
--- | --- | ---
Missing Data | – | –
Inconsistent Data | $S_{13}$, $S_{15}$ | $t_3$ and $t_9$ (or $t_{10}$) concurrently write some data into $x$

Redundant Data | – | –
Lost Data | $S_1^+ : \{S_{16}, S_{17}, S_{24}, S_{31}, S_{32}, S_{39}, S_{40}, S_{45}-S_{48}, S_{52}, S_{53}, S_{57}, S_{58}\}$, $S_2^+ : \{S_{11}, S_{12}, S_{19}, S_{20}, S_{28}, S_{29}, S_{37}, S_{38}, S_{45}-S_{48}, S_{54}, S_{55}, S_{60}, S_{61}\}$ | $t_{11}$ (resp. $t_{12}$) overwrites some data into $x$

$t_4$ and $t_7$ (or $t_8$) concurrently write some data into $y$

$t_{13}$ (resp. $t_{14}$) overwrites some data into $y$

Table 5. The result of checking data-flow errors

5 EXPERIMENTS

We do a set of experiments in order to compare RG- and GRG-based methods for checking data-flow errors in terms of state space and runtime.

In our experiments, a tool is developed to generate RGs and GRGs of any bounded WFD-nets. It is based on PIPE (Platform Independent Petri Net Editor) [6], which is an open source tool of Petri net. Our tool can draw, edit, import and export a WFD-net.

Our experimental benchmarks are listed as follows:

- **KIT**[^3] is a data set of BPMN 2.0 process models that describes 11 scenarios (including the business processes $BP_1 \sim BP_{11}$) with data specifications.
- **SystemC, blanc2010race** illustrates a SystemC (a modeling language) module.
- **AddGlobal, Sinha2010Staged** is an example of concurrency bugs when multithreads access shared variables.

We utilize a PC with Intel Core i5-2400 CPU (3.10 GHz) and 4.0 GB memory to do experiments. We first use WFD-nets to model these benchmarks in our tool, and then respectively obtain their RGs and GRGs.

Based on GRGs, we can check data-flow errors. Table 6 presents the results of our experiments for all benchmarks. Obviously, the scale of GRG is much smaller than RG. Meanwhile, our GRG-based method spends less time to produce a GRG than the RG-based method. Naturally, the former has an advantage over the latter in terms of checking data-flow errors.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>RG</th>
<th>GRG</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nos. of States</td>
<td>Nos. of Arcs</td>
<td>Time of RG</td>
</tr>
<tr>
<td>BP₁</td>
<td>13</td>
<td>13</td>
<td>75.2</td>
</tr>
<tr>
<td>BP₂</td>
<td>17</td>
<td>18</td>
<td>70.6</td>
</tr>
<tr>
<td>BP₃</td>
<td>21</td>
<td>28</td>
<td>85.2</td>
</tr>
<tr>
<td>BP₄</td>
<td>15</td>
<td>14</td>
<td>70.3</td>
</tr>
<tr>
<td>BP₅</td>
<td>10</td>
<td>11</td>
<td>67.7</td>
</tr>
<tr>
<td>BP₆</td>
<td>23</td>
<td>43</td>
<td>73.3</td>
</tr>
<tr>
<td>BP₇</td>
<td>12</td>
<td>13</td>
<td>51.4</td>
</tr>
<tr>
<td>BP₈</td>
<td>103</td>
<td>103</td>
<td>318.5</td>
</tr>
<tr>
<td>BP₉</td>
<td>16</td>
<td>15</td>
<td>55.5</td>
</tr>
<tr>
<td>BP₁₀</td>
<td>36</td>
<td>40</td>
<td>73.3</td>
</tr>
<tr>
<td>BP₁₁</td>
<td>111</td>
<td>218</td>
<td>1042.3</td>
</tr>
<tr>
<td>SystemC</td>
<td>33</td>
<td>62</td>
<td>76.6</td>
</tr>
<tr>
<td>AddGlobal</td>
<td>50</td>
<td>101</td>
<td>125.1</td>
</tr>
</tbody>
</table>

¹ Time: (ms).
² Errors: “I” denotes inconsistent data, “L” represents lost data, and “R” means redundant data.

Table 6. Experimental results

6 CONCLUSION

Petri net is widely used to check data-flow errors in workflow systems. As a special kind of Petri net, WFD-net is prominent in the modeling of control- and data-flows of business processes. Hence, we use a WFD-net to model workflow systems and its reachability graph to check data-flow errors in this paper. However, the classical reachability graphs of WFD-nets easily suffer from the problems of state space explosion and pseudo states. In order to avoid these problems, we propose a GRG-based method for checking data-flow errors. On one hand, our modeling method of WFD-net takes a smaller space than contextual net and PN-DO. On the other hand, our GRG-based method can effectively reduce the state space and avoid pseudo states in comparison with the classical reachability graph.

In the future work, we plan to do the following studies:

1. we utilize some existing techniques to reduce the scale of GRG, e.g., binary decision diagram (BDD) [5], abstraction [20] and partial order reduction [7]; and

2. we explore the unfolding-based technique of WFD-net [30] to check data-flow errors.

Acknowledgements

This paper was supported by Zhejiang Provincial Natural Science Foundation of China (Grant No. LQ20F020002), and in part by the Key Laboratory of Embedded
System and Service Computing (Ministry of Education) (Grant No. ESSCKF 2019-02).

REFERENCES


Dongming Xiang received his Ph.D. degree in computer science and technology from Tongji University, Shanghai, China, in 2018. He is currently Lecturer with the Department of Computer Science and Technology, Zhejiang Sci-Tech University. His research interests include model checking, Petri nets, business process management, and service computing.

Guanjun Liu received his Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2011. He was Post-Doctoral Research Fellow with the Singapore University of Technology and Design, Singapore, from 2011 to 2013. He was Post-Doctoral Research Fellow with the Humboldt University zu Berlin, Germany, from 2013 to 2014, supported by the Alexander von Humboldt Foundation. He is currently Professor with the Department of Computer Science and Technology, Tongji University. He has authored over 80 papers including 15 papers in IEEE/ACM Transactions and one book entitled Liveness of Petri Nets and Its Application (Tongji University Press, 2017). His research interests include Petri net theory, model checking, Web service, workflow, discrete event systems, and information security.