Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2022.018872

ARTICLE

Detecting and Repairing Data-Flow Errors in WFD-net Systems

Fang Zhao[1], Dongming Xiang[2,*], Guanjun Liu[1], Changjun Jiang[1] and Honghao Zhu[3]

[1]School of Electronics and Information Engineering, Tongji University, Shanghai, 201804, China

[2]School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, 310018, China

[3]School of Computer Engineering, Bengbu University, Bengbu, 233030, China

[*]Corresponding Author: Dongming Xiang. Email: flysky_xdm@163.com

Received: 22 August 2021 Accepted: 03 December 2021

ABSTRACT

Workflow system has become a standard solution for managing a complex business process. How to guarantee its correctness is a key requirement. Many methods only focus on the control-flow verification, while they neglect the modeling and checking of data-flows. Although some studies are presented to repair the data-flow errors, they do not consider the effect of delete operations or weak circulation relations on the repairing results. What's more, repairing some data-flow errors may bring in new errors. In order to solve these problems, we use workflow net with data (WFD-net) systems to model and analyze a workflow system. Based on weak behavioral relations and order relations in a WFD-net system, we formalize four kinds of data-flow errors. After then, we reveal the relations between these errors and organize them into a hierarchy. Furthermore, we propose some new methods to repair data-flow errors in a WFD-net system based on system requirements and repair strategies. Finally, a case study of campus-card recharging shows the applicability of our methods, and a group of experiments show their advantages and effectiveness.

KEYWORDS

Data-flow errors; WFD-net; weak behavioral relations; system requirements

Nomenclature

$\mathbb{N}$	Set of Positive Integers
$\lesseqgtr$	$\in$ or $\notin$
RD	Redundant Data
MD	Missing Data
LD	Lost Data
IND	Inconsistent Data

1 Introduction

A business process is considered as "the specific ordering of work activities across time and place, with a beginning, an end, and clearly identified input and output" [1]. As an information technology of process automation, workflow systems have become a standard solution for managing complex business process models, such as loan approval management [2], supply chain management [3], and knowledge management [4]. In other words, workflow systems can be used to automate, manage, and optimize different kinds of business processes. In general, the design of a workflow system needs to describe control-/data-flows, and then detect their errors [5]. A successful workflow system depends on error-free modeling and analyzing methods. Therefore, it is necessary to keep the correctness of control-/data-flows in a workflow system. Control-flows mainly focus on the partial orders of business activities, while data-flows mainly involve data operations (e.g., read, write, and delete) and guard functions [6–8].

There are many modeling methods that can be used to describe workflow systems, such as Business Process Modeling Notation (BPMN) [9], Business Process Execution Language (BPEL) [10], Event-Driven Process Chains (EPCs) [11], UML Activity Diagrams [11], and Workflow Net with Data (WFD-net) [12]. Compared with these methods, WFD-nets are much suitable to describe and analyze control-/data-flows due to their data labeling functions and guards.

In order to guarantee the correctness of workflow systems, some methods are developed to detect and repair errors. Unfortunately, most of them only focus on the verifications of control-flows while ignoring the checking of data-flows. In fact, data-flows play an important role in workflow design. This is because routing constraints in a workflow are usually based on data operations and guard functions. Moreover, data-flow errors (e.g., redundant data, missing data, lost data, and inconsistent data) may arise even if control-flows are correct [5,13]. Once a workflow model suffers from missing data, it may need high costs to repair unexpected process interruptions. To repair data-flow errors in workflow systems, many studies are done. Awad et al. [9] presented a method to repair missing data in BPMN. After then, Song et al. [10] provided some methods to preserve data-flow correctness in BPEL. Later on, Sharma et al. [14] presented a method to repair data-flow errors in workflows. However, these methods do not consider data-flow errors caused by delete operations, and their workflow models are usually simple (e.g., no loop structures). In order to remove a kind of data-flow error, they may bring in another one. Therefore, they need multiple attempts to repair these errors. What's worse, they may get an incorrect workflow model.

In this paper, we use WFD-net systems to analyze the correctness of control-/data-flows in workflow systems. It is assumed that our control-flows in WFD-net systems are sound (i.e., no deadlocks [9,15,16] or livelocks [17–19]), without taking data operations into consideration [12,20,21]. The aim of this paper is to detect and repair four kinds of data-flow errors (i.e., redundant data, missing data, lost data, and inconsistent data) in WFD-net systems.

The contributions of this paper are given as follows:

1) Based on weak behavioral relations (i.e., weak sequence relation, weak exclusiveness relation, weak circulation relation, and weak concurrency relation) and order relation, we formalize four kinds of data-flow errors in WFD-net systems.
2) We reveal the relations between different kinds of data-flow errors, and further organize them into a hierarchy, which can contribute to correctly repairing data-flow errors without repeated work.
3) We present some methods to repair data-flow errors in WFD-net systems according to several system requirements and repair strategies.

The rest of this paper is organized as follows. In Section 2, we present some related work about data-flow errors. Then, we introduce workflow system modeling and a motivation example in Section 3. In Section 4, we formalize data-flow errors in WFD-net systems, and provide some new methods to repair them. In Section 5, a case study of campus-card recharging is conducted. In Section 6, we do a group of experiments to show the effectiveness of our methods. Finally, conclusion and future work are given in Section 7.

2 Related Work

2.1 Detecting Methods for Data-Flow Errors

There have been many methods to detect data-flow errors. Sun et al. [2] provided three basic types of data-flow errors in UML activity diagram, namely missing data, conflicting data, and redundant data. Sundari et al. [22] presented a graph traversal algorithm called GT to detect data-flow errors in workflows. It systematically traversed every workflow route to detect errors like redundant data, missing data, and lost data. Based on the work provided in [2], Sun et al. [23] analyzed the dependencies among different transitions based on the input and output data of activities in a workflow model. Meda et al. [24] provided missing data, inconsistent data, lost data, and redundant data. They also presented a graph traversal algorithm to detect the data-flow errors in unstructured and nested workflows. Sidorova et al. [25] defined the may/must-soundness of a WFD-net system. Haddar et al. [26] provided a data-centric method to manage business processes. Dolean et al. [27] presented a survey of researches on modeling data-flows of business processes, and pointed out that the control-flows cannot be executed without data operations. Kabbaj et al. [5] pointed out that the role of data in a workflow is very important because routing constraints are closely related to data elements. Therefore, a workflow easily leads to an interruption once it contains data-flow errors. In order to solve this problem, they provided a method to detect redundant data, missing data, and lost data (or conflict data). Xiang et al. [6] proposed Petri Net with Data Operations (PN-DO), such as read, write, and delete operations. They formalized redundant data, missing data, lost data, and inconsistent data. Dramski [28] presented missing data in event logs and gave some ways to make some data recovery. Trcka et al. [29,30] put forward six kinds of data-flow errors in WFD-nets, namely missing data, strongly redundant data, weakly redundant data, strongly lost data, weakly lost data, and inconsistent data. von Stackelberg et al. [31] proposed a method to detect strongly redundant data, weakly redundant data, missing data, strongly lost data, weakly lost data, and inconsistent data in BPMN 2.0. Song et al. [32] utilized supervised learning algorithms to predict data-flow errors in an unseen BPEL process. Moreover, they provided an empirical study on data-flow errors in BPEL processes. Mülle et al. [33] considered that the correctness of data-flows in BPMN 2.0 is a challenge. As for the above studies, they did not consider the relations between different kinds of data-flow errors.

2.2 Repairing Methods for Data-Flow Errors

There are many studies on repairing data-flow errors. Awad et al. [9] presented a method to repair missing data. Song et al. [10] provided preserve data-flow correctness in process adaptation. They proposed three criteria to make the process free of missing data, redundant data, and lost data. Technically, they assumed that each activity has 0/1 output. Later on, Sharma et al. [14] presented methods to repair redundant data, missing data, lost data, and inconsistent data in workflows. After then, Jovanovikj et al. [34] provided methods to compare and merge two different models with different data operations. The methods are activity inserted, activity deletion, data object creation, data object to read, data object updated, and data object deletion. These methods can contribute to providing possible ways for the designer to repair data-flow errors in a WFD-net system. As mentioned above, these studies did not consider data-flow errors caused by delete operations, and their workflow models are usually without loop structures. What's more, repairing some kinds of data-flow errors may bring in new data-flow errors. In order to solve these problems, we propose some new methods to repair different kinds of data-flow errors according to several system requirements in this paper.

3 Preliminaries

3.1 Workflow System Modeling

A Petri net is a 3-tuple $N=(P,T,F)$, where P is a limited non-empty set of places, T is a limited non-empty set of transitions, $P\cap T=\emptyset$, and $F\subseteq(P\times T)\cup(T\times P)$ is the flow relationship in N [35–38]. A Petri net system is a net $N=(P,T,F)$ with an initial marking M_0. A marking function of a net is a mapping $M:P\to\mathbb{N}_1$, where $\mathbb{N}_1$ is the set of non-negative integers. Given a node $x\in P\cup T$, its pre-set and post-set are denoted by $^\cdot x=\{y|(y\in P\cup T)\wedge(y,x)\in F\}$ and $x^\cdot=\{y|(y\in P\cup T)\wedge(x,y)\in F\}$, respectively [39]. If x is a source place, then $^\cdot x=\emptyset$; If x is a sink place, then $x^\cdot=\emptyset$. Given a set of nodes $X\subseteq P\cup T$, and $x\in X$, its pre-set and post-set are denoted by $^\cdot X=\{Y|(Y\subseteq P\cup T)\wedge(y\in Y)\wedge(y,x)\in F\}$ and $X^\cdot=\{Y|(Y\subseteq P\cup T)\wedge(y\in Y)\wedge(x,y)\in F\}$, respectively. For the example of Fig. 1, if $X=\{t_2,t_3\}$, then we have $^\cdot X={}^\cdot t_2\cup{}^\cdot t_3=\{p_2,p_3\}$, and $X^\cdot=t_2^\cdot\cup t_3^\cdot=\{p_3,p_4\}$. Furthermore, we use k to denote the pre-set number for convenience. Therefore, we have $^{(\cdot)_1}t_3={}^\cdot t_3=\{p_3\}$, $^{(\cdot)_3}t_3={}^\cdot({}^\cdot({}^\cdot t_3))=\{p_2\}$, $^{(\cdot)_5}t_3={}^\cdot({}^\cdot({}^\cdot({}^\cdot({}^\cdot t_3))))=\{p_1,p_4\}$, and $^{(\cdot)_1}t_3\cup{}^{(\cdot)_3}t_3\cup{}^{(\cdot)_5}t_3=\{p_1,p_2,p_3,p_4\}$, where $k=1,3,5$, respectively. With respect to the set of post-sets, it is defined analogously. A Petri net N is called a safe Petri net if all places are marked at most one token. In this paper, our Petri nets are safe by default.

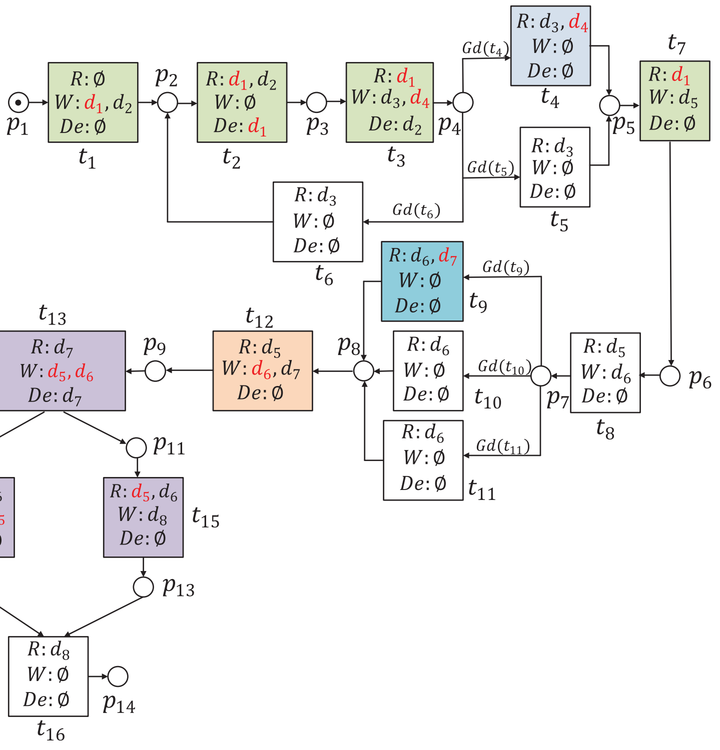

p_1
t_1 R:∅ W:d_1,d_2 De:∅
p_2
t_2 R:d_1,d_2 W:∅ De:d_1
p_3
t_3 R:d_1 W:d_3,d_4 De:d_2
p_4
t_4 R:d_3,d_4 W:∅ De:∅
$Gd(t_4)$
t_5 R:d_3 W:∅ De:∅
$Gd(t_5)$
t_6 R:d_3 W:∅ De:∅
$Gd(t_6)$
p_5
t_7 R:d_1 W:d_5 De:∅
p_6
t_8 R:d_5 W:d_6 De:∅
p_7
t_9 R:d_6,d_7 W:∅ De:∅
$Gd(t_9)$
t_{10} R:d_6 W:∅ De:∅
$Gd(t_{10})$
t_{11} R:d_6 W:∅ De:∅
$Gd(t_{11})$
p_8
t_{12} R:d_5 W:d_6,d_7 De:∅
p_9
t_{13} R:d_7 W:d_5,d_6 De:d_7
p_{10}
p_{11}
t_{14} R:d_6 W:d_5 De:∅
t_{15} R:d_5,d_6 W:d_8 De:∅
p_{12}
p_{13}
t_{16} R:d_8 W:∅ De:∅
p_{14}

Figure 1: A WFD-net system of campus-card recharging

We model the control-flows of a workflow system as a dedicated class of Petri nets, thereby modeling activities by transitions, conditions by places, and cases as tokens. A typical workflow has a well-defined starting point, and a well-defined ending point imposes syntactic restrictions on Petri nets that result in the following definition of a workflow net (WF-net).

Definition 1 (WF-net [40,41]**)** A Petri net $N = (P, T, F)$ is a workflow net (WF-net) if it satisfies:

(1) There is only one source place $p_I \in P$ and only one sink place $p_o \in P$ in N; and
(2) $\forall x \in P \cup T : (p_I, x) \in F^*$ and $(x, p_o) \in F^*$, where F^* is the reflexive-transitive closure of F.

A WF-net is used as a basic description of control-flows in a workflow process. A WF-net system (N, M_0) is a WF-net extended with an initial marking M_0. In order to model the actual process (including control-/data-flows) of workflow systems, we define a workflow net with data (*WFD-net*) [29,30] as follows.

Definition 2 (WFD-net [29,30]**)** A Petri net $ND = (P, T, F, D, Guard, R, W, De, Gd)$ is a WF-net with data (WFD-net) iff:

(1) (P, T, F) is a WF-net;
(2) D is a set of data items;
(3) $Guard$ is a set of guards over D;
(4) $R : T \rightarrow 2^D$ is a labeling function of reading data;
(5) $W : T \rightarrow 2^D$ is a labeling function of writing data;
(6) $De : T \rightarrow 2^D$ is a labeling function of deleting data; and
(7) $Gd : T \rightarrow Guard$ is a function of assigning guards to transitions.

A WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$ is a WFD-net with a marking M_0 in the source place. The function $Type : D \rightarrow \{Cons, Vari\}$ denotes the value type of data items. That is, if $d \in D$ is a constant, then we have $Type(d) = Cons$ (resp. if $d \in D$ is a variable, then we have $Type(d) = Vari$). A guard function is a boolean expression related to data items. In this paper, we suppose that guard functions in each branch are related to the same data items. For the sake of readability, when saying "data item d is read", it means "data item d is read or used for the evaluation of a guard function" [30]. Therefore, a guard function can be considered as read operations on a transition. For example, we have $Gd(t_4) =< high(d_3) >$, $Gd(t_5) =< middle(d_3) >$, $Gd(t_6) =< low(d_3) >$, $Gd(t_9) =< high(d_6) >$, $Gd(t_{10}) =< middle(d_6) >$, and $Gd(t_{11}) =< low(d_6) >$ in Fig. 1. The data item d_3 (resp. d_6) can be considered as read operations on transitions t_4, t_5, and t_6 (resp. t_9, t_{10}, and t_{11}).

3.2 Motivation Example

This part introduces a campus-card recharging process as a motivation example. As shown in Fig. 1, there are 14 places, 16 transitions, 8 data items, 34 data operations (including 19 read operations, 12 write operations, and 3 delete operations), and 6 guard functions. Table 1 illustrates the transitions and data items.

Table 1: Transitions and data items

Transition	Name	Transition	Name	Data item	Name
t_1	Log into the system	t_9	Choose bank card	d_1	ID
t_2	Click on application	t_{10}	Choose wechat	d_2	Login password
t_3	Click on campus wallet	t_{11}	Choose alipay	d_3	Balance of account
t_4	Recharge card	t_{12}	Submit orders	d_4	Card information
t_5	Pay the electricity	t_{13}	Pay	d_5	Payment ID
t_6	Go back	t_{14}	Enter account number	d_6	Transaction amount
t_7	Submit orders	t_{15}	Enter payment password	d_7	Merchant name
t_8	Choose payment method	t_{16}	Close the deal	d_8	Payment succeeds

Fig. 1 describes a WFD-net system as follows. Firstly, users log into the system, and produce two data items, i.e., *ID* and *Login Password*. Then, they click on the application and campus wallet. After these operations, there are three transitions in weak exclusiveness relations. If the balance of the account is high, users can recharge the campus smart cart. If the balance of the account is middle, they can pay the electricity. Otherwise, they go back to the application. Then, they need to submit orders. After submitting orders, users can choose payment methods, e.g., bank card, Wechat, or Alipay. If the payment amount is high, they can choose bank cards. If the payment amount is middle, they can choose Wechat. Otherwise, they only choose Alipay. After then, they submit and pay for orders. Users need to input account numbers and payment passwords. Finally, they close this deal.

There are some data-flow errors in Fig. 1. For example, the data item d_4 on the write operation of t_3 is weakly redundant, and the data item d_1 on the read operation of t_3 is strongly missing. More details about data-flow errors are given in Sections 4 and 5.

Although some methods are used to repair these errors, e.g., Criteria 1−3 [10] and CorrDF [14], they have some limitations. The method of Criteria 1−3 can repair data-flow errors with transition pairs in weak sequence relations. Each transition reads/writes 0/1 data item. The method of CorrDF can repair data-flow errors caused by incorrect read and write operations in process models with transition pairs in weak sequence relations, weak exclusiveness relations, and weak concurrency relations (Section 4). Unfortunately, they cannot repair data-flow errors caused by delete operations, and their workflow models are usually simple (e.g., no loop structures). What's more, repairing one error may bring in some new kinds of errors. For the example of Fig. 4h, when we repair the error of missing data, we may bring in some redundant data. Therefore, we need to reveal the scope of different kinds of data-flow errors, and present new methods to check and repair data-flow errors according to the real system requirements.

4 Data-flow Errors Checking and Repairing Methods

In this section, we first define *enabled conditions* of WFD-net systems, and their weak behavioral relations. Then, we present some methods to check and repair data-flow errors.

4.1 Firing Transitions and Weak Behavioral Relations

Definition 3 (Enabled Conditions) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, $t \in T$ is enabled at a marking M if it satisfies the following conditions:

(1) t is control-enabled at M such that $\forall p \in {}^{\cdot}t : M(p) \geq 1$, denoted by $M[_c t >$; and

(2) t is data-enabled at M such that each data item read by t must be defined and its guards are evaluated to true, which is denoted by $M[_d t>$.

The firing of an enabled transition t (i.e., control-enabled [42] and data-enabled) at a marking M does not only affect the values of data items and guard functions but also yields a new marking M' [25], which is denoted by $M[t > M'$. That is,

$$M'(p) = \begin{cases} M(p)+1, & \textit{if} \;\; p \in t^{\cdot} - {}^{\cdot}t, \\ M(p)-1, & \textit{if} \;\; p \in {}^{\cdot}t - t^{\cdot}, \\ M(p), & \textit{otherwise}. \end{cases}$$

A new marking M_k is reachable from the initial marking M_0, if and only if there exists a firing sequence $\sigma_k = t_1 t_2 \cdots t_k$ such that $M_0[t_1 > M_1[t_2 > M_2 \cdots M_{k-1}[t_k > M_k$.

Notice that if a transition $t \in T$ is control-enabled at M, we say it is weakly enabled, and its related firing sequence is a weak firing sequence.

Due to the fact that the modeling of a workflow system is process-oriented in reality, we need to analyze its *weak order relation* [43].

Definition 4 (Weak Order Relation [43]**)** Let $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$ be a WFD-net system. The weak order relation $\succ_c \subseteq T \times T$ contains all pairs of transitions (t_α, t_β), such that there exists a weak firing sequence (without taking the data operations and guard functions into consideration) $\sigma_x = t_1 t_2 \cdots t_x$ satisfying $(ND, M_0)[\sigma_x >$, $\alpha \in \{1, \cdots, x-1\}$, $\alpha < \beta \leq x$, and $x - 1 \in \mathbb{N}$.

In this paper, the set of all weak firing subsequences is denoted by T^*, and $T^* = \{\sigma_{(1)}\} \cup \{\sigma_{(2)}\} \cup \cdots \cup \{\sigma_{(\varepsilon)}\}$, where $\sigma_{(\varepsilon)} = t_{a_1} t_{a_2} \cdots t_{a_\varepsilon}$, and $t_{a_1}, t_{a_2}, \cdots, t_{a_\varepsilon} \in T$.

Based on the weak order relation, we define *weak behavioral relations* as follows.

Definition 5 (Weak Behavioral Relations) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, the transition pair (t_α, t_β) is in one of the following weak behavioral relations, where $k \in \{1, 3, \cdots, n\}$, $n = 2m+1$, and $m+1 \in \mathbb{N}$:

(1) t_α and t_β are in weak sequence relation, which is denoted by $t_\alpha \rightarrow_c t_\beta$, iff $(t_\alpha \succ_c t_\beta) \wedge (t_\beta \not\succ_c t_\alpha)$;
(2) t_α and t_β are in weak exclusiveness relation, which is denoted by $t_\alpha \otimes_c t_\beta$, iff $(t_\alpha \not\succ_c t_\beta) \wedge (t_\beta \not\succ_c t_\alpha)$;
(3) t_α and t_β are in weak circulation relation, which is denoted by $t_\alpha \leftrightarrow_c t_\beta$, iff $(t_\alpha \succ_c t_\beta) \wedge (t_\beta \succ_c t_\alpha) \wedge (t_\alpha^{\cdot} \cap (\cup_{k=1}^{n} {}^{(\cdot)k} t_\beta) \neq \emptyset) \wedge (t_\beta^{\cdot} \cap (\cup_{k=1}^{n} {}^{(\cdot)k} t_\alpha) \neq \emptyset)$; or
(4) t_α and t_β are in weak concurrency relation, which is denoted by $t_\alpha \oplus_c t_\beta$, iff $(t_\alpha \succ_c t_\beta) \wedge (t_\beta \succ_c t_\alpha) \wedge ((t_\alpha^{\cdot} \cap (\cup_{k=1}^{n} {}^{(\cdot)k} t_\beta) = \emptyset) \vee (t_\beta^{\cdot} \cap (\cup_{k=1}^{n} {}^{(\cdot)k} t_\alpha) = \emptyset))$.

Generally speaking, $t_\alpha \rightarrow_c t_\beta$ represents the transition t_β cannot execute before t_α in any traces. $t_\alpha \otimes_c t_\beta$ means the transitions t_α and t_β execute exclusively in any traces. $t_\alpha \leftrightarrow_c t_\beta$ illustrates the transitions t_α and t_β are in a loop structure. $t_\alpha \oplus_c t_\beta$ denotes the transitions t_α and t_β can execute in different orders. Obviously, these four kinds of weak behavioral relations are mutually exclusive. If t_α and t_β are in weak sequence relation and they are in the same branch (i.e., $(t_\alpha \rightarrow_c t_\beta) \wedge (t_\alpha^{\cdot} = {}^{(\cdot)k_1} t_\beta) \wedge (|t_\alpha^{\cdot}| = |{}^{(\cdot)k_1} t_\beta| = 1)$, where $k_1 \in \{1, 3, \cdots, n_1\}$, $n_1 = 2m_1 + 1$, and $m_1 + 1 \in \mathbb{N}$), then we denote it as $t_\alpha \twoheadrightarrow_c t_\beta$. For the example of Fig. 1, we have $t_1 \rightarrow_c t_2$, $t_4 \otimes_c t_5$, $t_3 \leftrightarrow_c t_6$, $t_{12} \twoheadrightarrow_c t_{13}$, and $t_{14} \oplus_c t_{15}$, respectively.

Definition 6 (Concurrency Relation) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, the transitions t_α and t_β are in *concurrency relation*, which is denoted by $t_\alpha \oplus_{cd} t_\beta$, iff:

(1) $t_\alpha \oplus_c t_\beta$; and
(2) The guard functions related to t_α and t_β are satisfied at the same time.

For the example of Fig. 2a, we can find that $t_\alpha \oplus_{cd} t_\beta$ if $Gd(t_\alpha)$ and $Gd(t_\beta)$ are evaluated to TRUE at the same time.

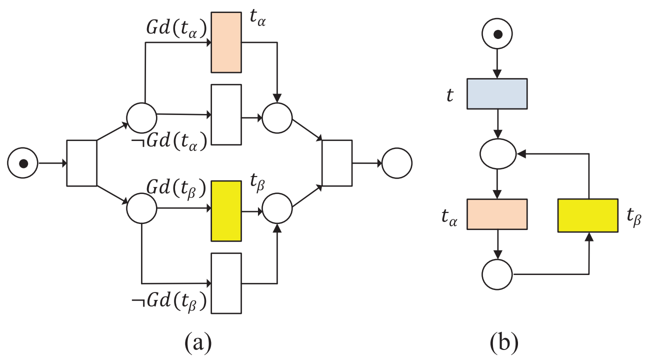


Figure 2: Two WFD-net systems. (a) The behavioral relation of t_α and t_β depends on their guard functions; (b) t_α and t_β are in weak circulation relation (i.e., $t_\alpha \leftrightarrow_c t_\beta$)

Definition 7 (Distance of Transitions in Weak Circulation Relation) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, $\exists t_\alpha, t_\beta \in T$: $t_\alpha \leftrightarrow_c t_\beta$, $t_\alpha \in t^{(\cdot)k_1}$, and $t_\beta \in t^{(\cdot)k_2}$, where t is a transition outside the weak circulation relation, k_1 and k_2 are the minimum number of post-sets, such that $k_1, k_2 \in \{2, 4, \cdots, 2m\}$, and $m \in \mathbb{N}$. The distance between t_α and t_β is defined as $\delta(t_\beta, t_\alpha) = |k_2 - k_1|$.

Based on the definitions of weak behavioral relations, concurrency relation, and distance of transitions in weak circulation relation, we further formalize the *order relation* of transition pairs in a WFD-net system.

Definition 8 (Order Relation) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, the order relation $t_\alpha \succ t_\beta$ contains all pairs of transitions (t_α, t_β), and they satisfy one of the following conditions:

(1) $t_\alpha \rightarrow_c t_\beta$;
(2) $t_\alpha \oplus_{cd} t_\beta$; or
(3) $(t_\alpha \leftrightarrow_c t_\beta) \wedge (k_2 \geq k_1)$.

For the example of Fig. 2b, we have $t \rightarrow_c t_\alpha$, $t \rightarrow_c t_\beta$, $t_\alpha \leftrightarrow_c t_\beta$, and $\delta(t_\beta, t_\alpha) = |k_2 - k_1| = 2$.

4.2 Four Kinds of Data-Flow Errors

Based on order relation, we formalize *redundant data*, *missing data*, *lost data*, and *inconsistent data*. Before we define these errors, we use $A \lesseqgtr B$ to denote $A \in B$ or $A \notin B$.

Definition 9 (Redundant Data, *RD*) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, T^* is the set of all weak firing subsequences, and $\exists \sigma \in T^*$, $\exists t_o \in \sigma : p_o \in t_o^{\cdot}$. A data item $d \in D$ is redundant if it satisfies $\exists t \in T$, $\forall t' \in \sigma : (t \succ t') \wedge (d \in W(t)) \wedge \left[\left(d \lesssim De(t)\right) \vee \left(d \lesssim W(t') \cup De(t')\right)\right] \wedge (d \notin R(t'))$, which is denoted by $d \lessdot RD$.

Notice that if there exist $t, t' \in T$ such that $(t \succ t') \wedge [d \in W(t) \cap R(t')] \wedge (d \notin De(t)) \wedge \left[d \lesssim W(t') \cup De(t')\right]$, then d may be redundant. For the example of Fig. 3e, we have $(t_1 \succ t_2) \wedge [d \in W(t_1) \cap R(t_2)] \wedge (d \notin De(t_1)) \wedge [d \notin W(t_2) \cup De(t_2)]$, and $d \lessdot RD$.

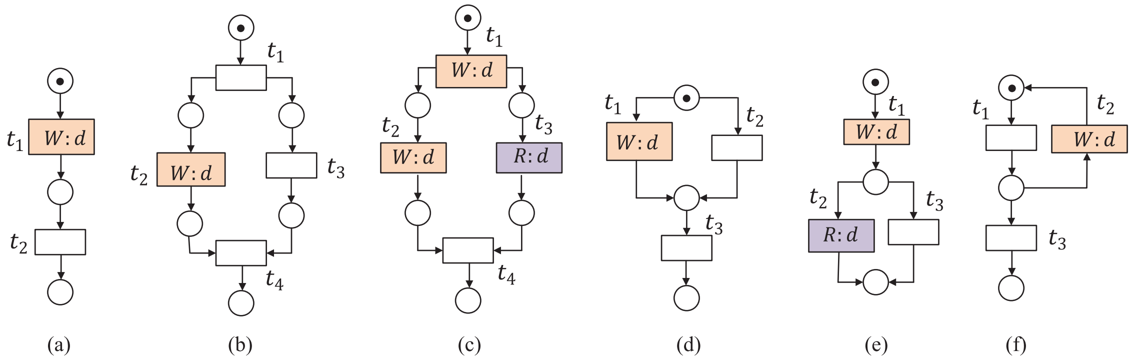


Figure 3: The data item d is redundant, where $Type(d) = Cons$ or $Type(d) = Vari$. (a)–(c) $d \lessdot SRD$; (d)–(f) $d \lessdot WRD$

A data item d is strongly redundant data (SRD) if the written values of d in all possible execution paths are never read before it is deleted, or the workflow process is completed. A data item d is weakly redundant data (WRD) if there exist some execution paths in which it is written but never read afterward, i.e., before it is deleted, or the workflow process is completed [30]. SRD and WRD belong to redundant data (RD). As shown in Fig. 3, the data item d is redundant, where (a)–(c) suffer from strongly redundant data, and (d)–(f) suffer from weakly redundant data.

Definition 10 (Missing Data, *MD*) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, T^* is the set of all weak firing subsequences, $\exists \sigma \in T^*$, and $\exists t_i \in \sigma : p_I \in {}^{\cdot}t_i$. A data item $d \in D$ is missing if it satisfies one of the following conditions:

(1) $\exists t' \in T$, $\forall t \in \sigma : (t \succ t') \wedge [((d \notin W(t)) \wedge (d \in De(t) \cup R(t'))) \vee ((d \notin W(t) \cup W(t')) \wedge (d \in De(t')))]$, which is denoted by $d \lessdot MD$; or

(2) $\exists t \in \sigma$, $\exists t'' \in T, \forall t' \in T : (t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge [d \in W(t) \cap (De(t) \cup De(t')) \cap (R(t'') \cup De(t''))] \wedge \left(d \lesssim R(t')\right) \wedge \left[d \notin W(t') \cup W(t'')\right]$, which is denoted by $d \lessdot MD$.

Notice that if there exist $t, t' \in T$ such that $(t \succ t') \wedge [d \in W(t) \cap (R(t') \cup De(t'))] \wedge [d \notin De(t) \cup W(t')]$, then d may be missing, e.g., Figs. 4c, 4d and 4f.

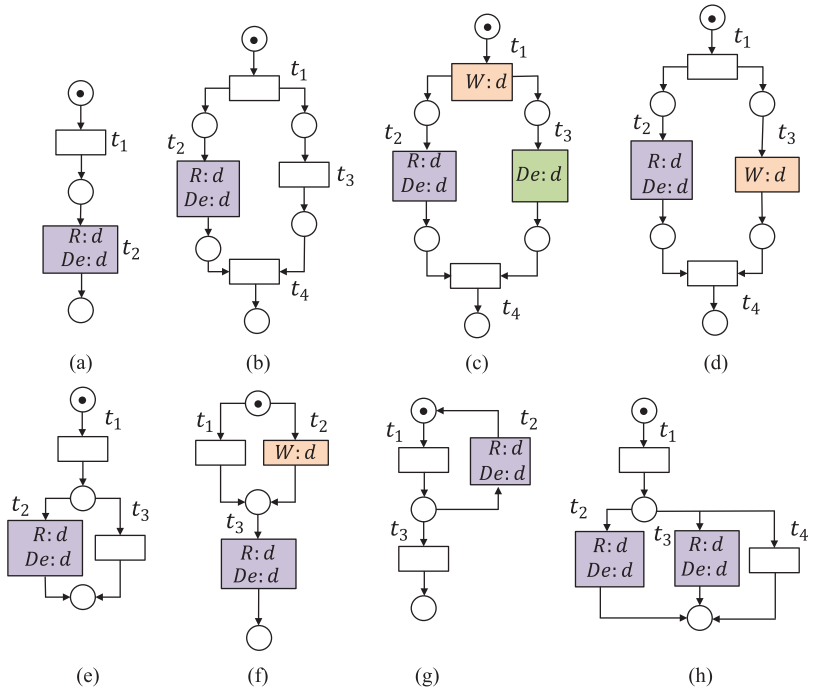


Figure 4: The data item d is missing, where $Type(d) = Cons$ or $Type(d) = Vari$. (a)–(c) $d \lessdot SMD$; (d)–(h) $d \lessdot WMD$

A data item d is strongly missing data (SMD) if the read or deleted values of d in all possible execution paths are never written before. A data item d is weakly missing data (WMD) if there exist some execution paths where it is read or deleted but never written before [30]. SMD and WMD belong to the missing data (MD). It is a terminal error for the execution of workflow systems [24]. For example, the data item d is strongly missing in Figs. 4a–4c, and it is weakly missing in Figs. 4d–4h.

Definition 11 (Lost Data, *LD*) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, a data item $d \in D$ is lost if it satisfies one of the following conditions:

(1) $\exists t, t' \in T,\ t^{\cdot} \cap {}^{\cdot}t' \neq \emptyset : [d \in W(t) \cap W(t')] \wedge [d \notin De(t) \cup R(t')]$, which is denoted by $d \lessdot LD$; or
(2) $\exists t, t'' \in T, \forall t' \in T:\ (t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge [d \in W(t) \cap W(t'')] \wedge [d \notin De(t) \cup R(t') \cup W(t') \cup De(t') \cup R(t'')]$, which is denoted by $d \lessdot LD$.

A data item d is strongly lost data (SLD) if the written values of d in all possible execution paths are overwritten without being read or deleted first. A data item d is weakly lost data (WLD) if there is an execution path where it is overwritten without being read or deleted first [30]. The SLD and WLD all belong to the lost data (LD). In Figs. 5a–5c, the data item d is strongly lost and strongly redundant. Sometimes, weakly redundant in weak circulation relation can lead to weakly lost, as shown in Fig. 5d. In Fig. 5e, the data item d is weakly lost and weakly redundant. In fact, if $d \lessdot LD$, then $d \lessdot RD$ [24]. The lost data is also called conflicting data [2].

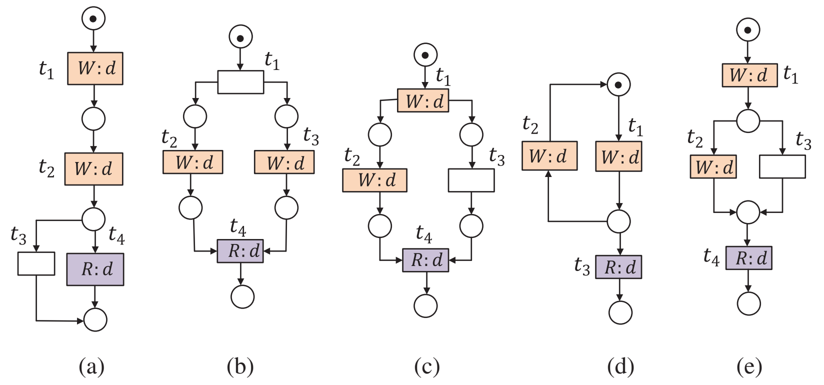


Figure 5: The data item d is lost, where $Type(d) = Cons$ or $Type(d) = Vari$. (a)–(c) $d \lessdot SLD$; (d)–(e) $d \lessdot WLD$

Definition 12 (Inconsistent Data, *IND*) Given a WFD-net system $(ND, M_0) = (P, T, F, M_0, D, Guard, R, W, De, Gd)$, a data item $d \in D$ is inconsistent if it satisfies $\exists t, t' \in T : (t \oplus_{cd} t') \wedge (Type(d) = Vari) \wedge [d \in (R(t) \cup W(t) \cup De(t)) \cap (W(t') \cup De(t'))]$, which is denoted by $d \lessdot IND$.

In Fig. 6, the data item d is inconsistent when $Type(d) = Vari$. Meanwhile, the data item d is weakly lost and strongly redundant in Figs. 6a and 6c, and d is also strongly lost and strongly redundant in Fig. 6b. The error of inconsistent data comes up when a transition accesses a data item while its concurrent transitions write or delete the same data item (there is no locking mechanism [19]). Notice that, if $Type(d) = Cons$, there is no inconsistent data on the data item d.

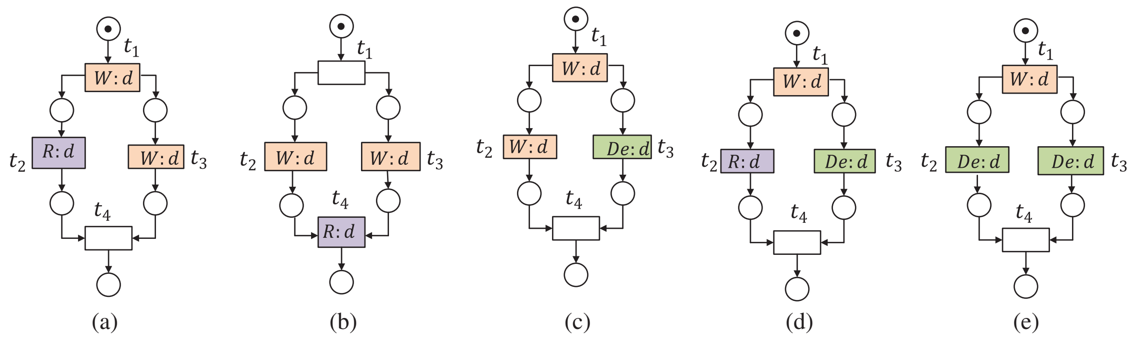


Figure 6: The data item d is inconsistent, where $Type(d) = Vari$.(a) – (e) $d \lessdot$ IND

A data item written by one transition may belong to several kinds of data-flow errors. The errors of inconsistent data sometimes lead to redundant, missing, or lost data. Lost data causes redundant data. In order to distinguish their relations, we reveal their scopes, as shown in Fig. 7a.

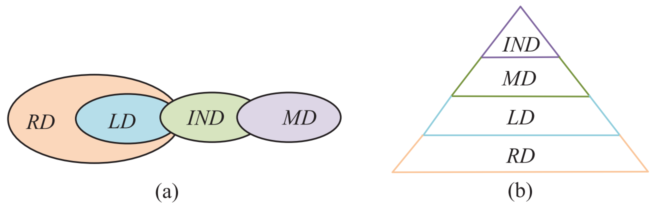


Figure 7: Four kinds of data-flow errors, i.e., redundant data (*RD*), missing data (*MD*), lost data (*LD*), and inconsistent data (*IND*). (a) The scopes of data-flow errors; (b) The hierarchy of these errors

According to the scopes of data-flow errors, we propose Algorithm 1 to detect them. This algorithm is an iterative procedure, and it consists of seven conditions. It can be used as a road map for implementing data-flow errors detection in a WFD-net system [2].

Algorithm 1: Data-flow errors detection algorithm

```
Require: A WFD-net system $(ND, M_0)$, a data item $d \in D$
Ensure: Data-flow errors in $(ND, M_0)$.
(1) For each $d \in D$ Do
(2) Traverse the firing subsequences $\sigma \in T^*$ in the state space of $(ND, M_0)$;
(3) If there exist $t, t' \in T$ such that $(t \oplus_{cd} t') \wedge (Type(d) = Vari) \wedge [d \in (R(t) \cup W(t) \cup De(t)) \cap (W(t') \cup De(t'))]$ Then
(4) Print $d \lessdot IND$;
(5) Else if there exist $t' \in T$ and $t_i \in \sigma$ satisfying $p_I \in {}^{\cdot}t_i$ Then
(6) For each $t \in \sigma$ such that $(t \succ t') \wedge [((d \notin W(t)) \wedge (d \in De(t) \cup R(t'))) \vee ((d \notin W(t) \cup W(t')) \wedge (d \in De(t')))]$ Do
(7) Print $d \lessdot MD$;
(8) End for
(9) Else if there exist $\in \sigma, t'' \in T$, and $t_i \in \sigma$ satisfying $p_I \in {}^{\cdot}t_i$ Then
(10) For each $t' \in T$ such that $(t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge [d \in W(t) \cap (De(t) \cup De(t')) \cap (R(t'') \cup De(t''))] \wedge \left(d \lesssim R(t')\right) \wedge [d \notin W(t') \cup W(t'')]$ Do
(11) Print $d \lessdot MD$;
(12) End for
(13) Else if there exist $t, t' \in T$, and $t^{\cdot} \cap {}^{\cdot}t' \neq \emptyset$ such that $[d \in W(t) \cap W(t')] \wedge [d \notin De(t) \cup R(t')]$ Then
(14) Print $d \lessdot LD$ and $d \lessdot RD$;
(15) Else if there exist $t, t'' \in T$ Then
(16) For each $t' \in T$ such that $(t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge [d \in W(t) \cap W(t'')] \wedge [d \notin De(t) \cup R(t') \cup W(t') \cup De(t') \cup R(t'')]$ Do
(17) Print $d \lessdot LD$ and $d \lessdot RD$;
(18) End for
(19) Else if there exist $t \in T$ and $t_o \in \sigma$ satisfying $p_o \in t_o^{\cdot}$ Then
(20) For each $t' \in \sigma$ such that $(t \succ t') \wedge (d \in W(t)) \wedge \left[\left(d \lesssim De(t)\right) \vee \left(d \lesssim W(t') \cup De(t')\right)\right] \wedge (d \notin R(t'))$ Do
(21) Print $d \lessdot RD$;
(22) End for
```

(Continued)

Algorithm 1: (Continued)

```
(23) Else
(24) Print No data-flow errors;
(25) End if
(26) End for
```

4.3 Repairing Data-flow Errors

In order to guarantee the correctness of WFD-net systems, we should provide effective methods to repair different kinds of data-flow errors. However, it is unnecessary to delete the read operation in an original WFD-net system because some transitions can fire only through reading these data items [10]. More importantly, we should not change the value we need to repair all kinds of data-flow errors. When a data item in a WFD-net system only consists of a read, write, or delete operation, we need to add a write operation, remove a write operation, or remove a delete operation. How to place these read, write, and delete operations means configuring the data-flow reasonably. In detail, we need to consider the positions of a data item. If the read and write operations are in inappropriate positions, there may appear new data-flow errors. Therefore, we give the following repair strategies according to the real system requirements to repair different kinds of data-flow errors.

System requirements:

(1) Take the value of data items into consideration;
(2) Avoid bringing in any new data-flow errors as far as possible;
(3) Do not lead to any control-flow errors, e.g., deadlock and livelock;
(4) Do not remove the read operation in an original WFD-net system;
(5) Do not bring in more data operations in order to repair the weakly redundant;
(6) Do not repair some kinds of weakly lost data so as to satisfy the non-determinacy of some read operations afterward;
(7) Avoid some data-flow errors caused by the delete operation; and
(8) Make sure that there is no inconsistent data by taking the locking mechanism.

Repair strategies:

(1) Change the firing sequence of transitions;
(2) Combine several transitions into one transition;
(3) Remove redundant write operations;
(4) Remove some undesirable delete operations;
(5) Divide one transition into several transitions; and
(6) Create new transitions with write operations.

As we know, inconsistent data often exists in concurrent programs. We can adopt the locking mechanism [18] to block the executions of multiple threads, e.g., a thread runs in a concurrent program without intervening in other threads. That is, the data item read/deleted by a thread only comes from the main thread or itself. Therefore, we do not provide related algorithms to repair this kind of error in this paper. After then, we propose Algorithms 2–4 to repair missing data, lost data, and redundant data, respectively. In the following algorithms, when we add or remove a transition, we need to add or remove its related places and arcs.

4.3.1 Repairing Missing Data (MD)

In order to repair missing data, we need to add a write operation, or remove a delete operation. It is unnecessary to remove read operations in an actual net as some transitions can fire only through reading these data items. Therefore, we propose Algorithm 2 to repair this kind of error. Its basic solutions are to introduce a transition that can write a required data item d at some point where the error occurs (i.e., Steps (1)–(12) and Steps (23)–(28) in Algorithm 2). Meanwhile, we remove some undesirable d in a delete operation (i.e., Steps (1)–(28) in Algorithm 2), and put the existing d just before all the branches where the error occurs (i.e., Steps (18)–(22) and (27)–(36) in Algorithm 2). After Steps (1)–(12), the missing data in Figs. 4a, 4b, 4d, 4e, 4g and 4h can be repaired. As for the errors in Fig. 4c (resp. Fig. 4f), we can take Steps (18)–(22) (resp. Steps (34)–(36)) to repair them. Fig. 8 shows the repairing results of missing data in Fig. 4.

Algorithm 2: Repairing missing data (MD) in a WFD-net system

```
Require: A WFD-net system $(ND, M_0)$, a data item $d \in D$
Ensure: A WFD-net system without missing data ($MD$).
(1) For each $t \in \sigma$, where $\sigma \in T^*$ and $t_i \in \sigma$ satisfying $p_I \in {}^{\cdot}t_i$ Do
(2)   If there exists $t' \in T$ such that $(t \succ t') \wedge (d \notin W(t)) \wedge [d \in De(t) \cup R(t')]$ Then
(3)     Analyze the value type of $d$;
(4)     If there exist $t'' \in T$ and $d$ such that $(Type(d) = Cons) \wedge (t' \otimes_c t'') \wedge (d \in R(t''))$ Then
(5)       Remove $d$ from $De(t)$ (if it exists), and add a new transition $t_k$ into $T$ satisfying $(t_k \rightarrow_c t') \wedge (t_k \rightarrow_c t'') \wedge (d \in W(t_k))$;
(6)     Else if there exist $t'' \in T$ and $d$ such that $(Type(d) = Vari) \wedge (t' \otimes_c t'') \wedge (d \in R(t''))$ Then
(7)       Remove $d$ from $De(t)$ (if it exists), and add two new transitions $t_k$ and $t_{k'}$ into $T$ satisfying $(t_k \twoheadrightarrow_c t') \wedge (t_{k'} \twoheadrightarrow_c t'') \wedge [d \in W(t_k) \cap W(t_{k'})]$;
(8)     Else
(9)       Remove $d$ from $De(t)$ (if it exists), and add a new transition $t_k$ into $T$ satisfying $(t_k \twoheadrightarrow_c t') \wedge (d \in W(t_k))$;
(10)    End if
(11)  End if
(12) End for
(13) For each $t \in \sigma$ Do
(14)   If there exists $t' \in T$ such that $(t \succ t') \wedge [d \notin W(t) \cup W(t')] \wedge (d \in De(t'))$ Then
(15)     Remove $d$ from $De(t')$;
(16)   End if
(17) End for
(18) For each $t' \in T$ Do
(19)   If there exist $t \in \sigma$ and $t'' \in T$ such that $(t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge \left[d \in W(t) \cap (De(t) \cup De(t')) \cap (R(t'') \cup De(t''))\right] \wedge \left(d \leq R(t')\right)$
       $\wedge [d \notin W(t') \cup W(t'')]$ Then
(20)     Analyze the value type of $d$ and data operations on $t$, $t'$, and $t''$;
(21)     If $(Type(d) = Cons) \wedge [d \in W(t) \cap (De(t) \cup De(t')) \cap (R(t'') \cup De(t''))] \wedge \left(d > R(t')\right) \wedge [d \notin W(t') \cup W(t'')]$ Then
(22)       Remove $d$ from $De(t)$ and $De(t')$ (if there exist), and divide $t$ into two transitions $t_k$ and $t_{k'}$, such that $\{t_{k'}\} =^{(\cdot)x_1} t' =^{(\cdot)x_2} t''$, where $x_1, x_2 \in \{1, 3, \cdots, n_1\}$, $n_1 = 2m_1 - 1$, and $m_1 \in \mathbb{N}$, $R(t_k) = R(t) - \Gamma(d)$, $W(t_k) = W(t) - \{d\}$, $De(t_k) = De(t) - \Gamma(d)$, $R(t_{k'}) = \Gamma(d)$, $W(t_{k'}) = \{d\}$, and $De(t_{k'}) = \Gamma(d)$; /*$\Gamma(d)$ is a set of data items only related to $d$ on $t$ */
```

(Continued)

Algorithm 2: (Continued)

```
(23) Else if $(Type(d) = Vari) \wedge [d \in W(t) \cap De(t) \cap R(t') \cap (R(t'') \cup De(t''))] \wedge (d \lesssim De(t')) \wedge [d \notin W(t') \cup W(t'')]$ Then
(24) Remove $d$ from $W(t)$ and $De(t)$, and add two new transitions $t_{k_1}$ and $t_{k_2}$ into $T$ satisfying $(t_{k_1} \twoheadrightarrow_c t') \wedge (t_{k_2} \twoheadrightarrow_c t'') \wedge [d \in W(t_{k_1}) \cap W(t_{k_2})]$;
(25) Else if $(Type(d) = Vari) \wedge [d \in W(t) \cap (De(t) \cup De(t')) \cap (R(t'') \cup De(t''))] \wedge [d \notin R(t') \cup W(t') \cup W(t'')]$ Then
(26) Remove $d$ from $W(t)$, $De(t)$, and $De(t')$(if it exists), and add a new transition $t_{k_1}$ into $T$ satisfying $(t_{k_1} \twoheadrightarrow_c t'') \wedge (d \in W(t_{k_1}))$;
(27) Else if $(Type(d) = Vari) \wedge [d \in W(t) \cap R(t') \cap De(t') \cap (R(t'') \cup De(t''))] \wedge [d \notin De(t) \cup W(t') \cup W(t'')]$ Then
(28) Divide $t$ into two transitions $t_k$ and $t_{k'}$, and add a new transition $t_{k_1}$ into $T$, with $t_{k'} \twoheadrightarrow_c t'$, $t_{k_1} \twoheadrightarrow_c t''$, $R(t_k) = R(t) - \Gamma(d)$, $W(t_k) = W(t) - \{d\}$, $De(t_k) = De(t) - \Gamma(d)$, $R(t_{k'}) = \Gamma(d)$, $W(t_{k'}) = \{d\}$, $De(t_{k'}) = \Gamma(d)$, and $d \in W(t_{k_1})$;
(29) Else
(30) No missing data;
(31) End if
(32) End if
(33) End for
(34) If there exist $t, t' \in T$ such that $(d \lessdot MD) \wedge (t \succ t') \wedge [d \in W(t) \cap (R(t') \cup De(t'))] \wedge [d \notin De(t) \cup W(t')]$ Then
(35) Divide $t$ into two transitions $t_k$ and $t_{k'}$, with $t_{k'} \twoheadrightarrow_c t'$, $R(t_k) = R(t) - \Gamma(d)$, $W(t_k) = W(t) - \{d\}$, $De(t_k) = De(t) - \Gamma(d)$, $R(t_{k'}) = \Gamma(d)$, $W(t_{k'}) = \{d\}$, and $De(t_{k'}) = \Gamma(d)$;
(36) End if
```

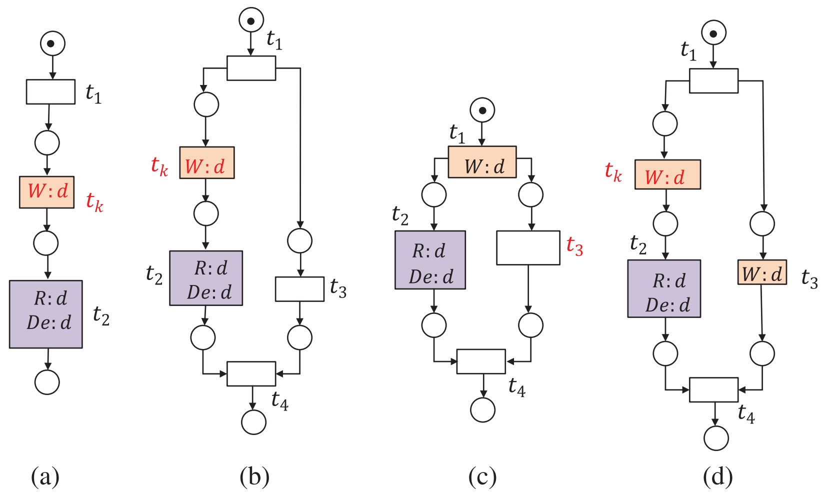


Figure 8: (Continued)

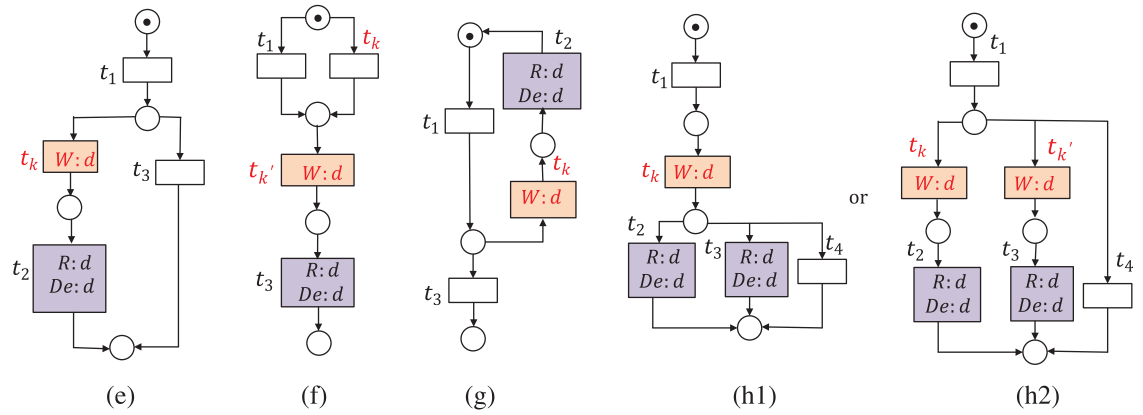


Figure 8: The repairing results of Figs. 4a–4g are shown in (a)–(g), and the repairing result of Fig. 4h is shown in (h1) if $Type(d) = Cons$ (resp. (h2) if $Type(d) = Vari$)

4.3.2 Repairing Unnecessary Lost Data (LD)

In order to repair lost data, we need to remove a write operation [10]. However, some kinds of lost data should not be repaired, especially when the type of a data item d is a variable.

If a data item is lost, it must be redundant, but not vice versa. Therefore, we propose Algorithm 3 to repair lost data before repairing redundant data. Figs. 5a–5e illustrates the cases of lost data. Its basic solutions are to remove the unnecessary data item in a write operation (i.e., Steps (1)–(4), (7)–(8), (11)–(15), and (18)–(19) in Algorithm 3) or not repair (i.e., Steps (5)–(6) and (16)–(17) in Algorithm 3). If a data item d satisfies $Type(d) = Cons$, the lost data in Figs. 5a–5e can be repaired after Steps (1)–(4). If a data item d satisfies $Type(d) = Vari$, the lost data in Figs. 5a,5c and 5d can be repaired after Steps (7)–(8). However, if a data item d satisfies $Type(d) = Vari$, the lost data in Figs. 5b and 5e should not be repaired because the data item d read by a transition t_4 depends on the branch to fire (Steps (5)–(6) in Algorithm 3). Fig. 9 shows the repairing results of lost data in Fig. 5.

Algorithm 3: Repairing unnecessary lost data (LD) in a WFD-net system

```
Require: A WFD-net system $(ND, M_0)$, and a data item $d \in D$
Ensure: A WFD-net system without any unnecessary lost data ($LD$).
(1) If there exist $t, t' \in T$, and $t^{\bullet} \cap {}^{\bullet}t' \neq \emptyset$ satisfying $[d \in W(t) \cap W(t')] \wedge [d \notin De(t) \cup R(t')]$ Then
(2) Analyze the value type of $d$;
(3) If $Type(d) = Cons$ Then
(4) Remove $d$ from $W(t)$ or $W(t')$ without bringing in a missing data;
(5) Else if $Type(d) = Vari$ and there exists $t'' \in T$ such that $(t \succ t'') \wedge (t' \succ t'')$, where $d \in W(t) \cap R(t'')$ or $d \in W(t') \cap R(t'')$ Then
(6) Not repair;
(7) Else
(8) Remove $d$ from $W(t)$;
(9) End if
(10) End if
```

(Continued)

Algorithm 3: (Continued)

```
(11) For each $t' \in T$ Do
(12) If there exist $t, t'' \in T$ such that $(t \succ t'') \wedge (t \succ t') \wedge (t' \succ t'') \wedge [d \in W(t) \cap W(t'')] \wedge [d \notin De(t) \cup R(t') \cup W(t') \cup De(t') \cup R(t'')]$ Then
(13) Analyze the value type of $d$;
(14) If $Type(d) = Cons$, and there exists $t''' \in T$ such that $(t \rightarrow_c t''') \wedge (t'' \oplus_{cd} t''') \wedge (d \in R(t''') \cup De(t'''))$ Then
(15) Remove $d$ from $W(t'')$;
(16) Else if $Type(d) = Vari$, and there exists $t''' \in T$ satisfying $(t \rightarrow_c t''') \wedge (t'' \oplus_{cd} t''') \wedge [d \in R(t''') \cup De(t''')]$ Then
(17) $d \lessdot IND$, and we should adopt the locking mechanisms and not repair;
(18) Else
(19) Remove $d$ from $W(t)$;
(20) End if
(21) End if
(22) End for
```

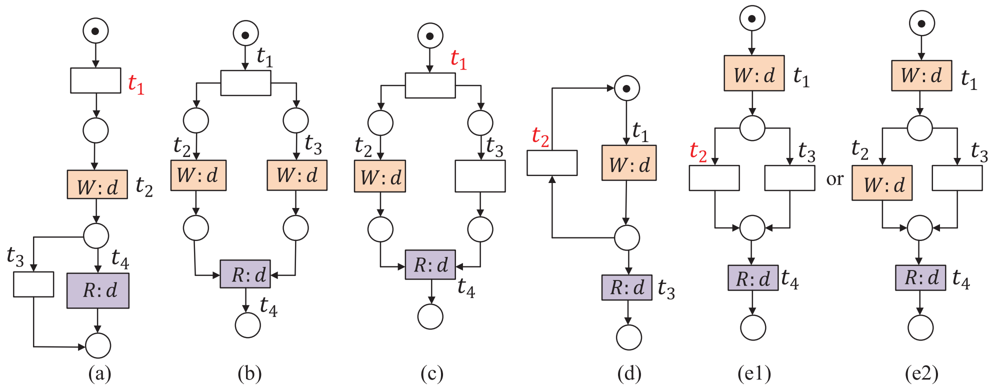


Figure 9: The repairing results of Figs. 5a–5d, are shown in (a)–(d), and the repairing result of Fig. 5e is shown in (e1) if $Type(d) = Cons$ (resp. (e2) if $Type(d) = Vari$)

4.3.3 Repairing Unnecessary Redundant Data (RD)

If a data item d is redundant, it may reduce the efficiency of programs. In order to repair this error, we need to remove a write operation [10]. However, some kinds of weakly redundant may make programs consume less memory and time. That is to say, the existence of some weakly redundant may be more convenient, and a moderate amount of weakly redundant are required, as shown in Figs. 10a–10b. Notice that we should not change the value we need when we repair this kind of error, as shown in Fig. 10c. Therefore, the following three kinds of weakly redundant should not be repaired, as shown in Fig. 10.

(1) A data item d is weakly redundant, but it is also read by more than one branch. We use $n(R)$ to denote the number of branches that can read d. For the example of Fig. 10a, we have $n(R) = 2$ corresponding to the transitions t_2 and t_3;
(2) The write operation of d is not in a weak circulation relation while the read operation is in a weak circulation relation. For the example of Fig. 10b, we use $\varsigma(W(t_1))$ to denote the number of times of transitions like t_1 that can write d, and we use $\varsigma(R(t_2))$ to denote the

number of times of transitions like t_2 that can read d. Therefore, we have $\varsigma(W(t_1)) = 1$ and $\varsigma(R(t_2)) \geq 1$; and

(3) The data item d satisfies $Type(d) = Vari$, and the transitions which write and read d are in weak circulation relation. For the example of Fig. 10c, we have $(t_2 \leftrightarrow_c t_3) \wedge (d \in W(t_1) \cap R(t_2) \cap W(t_2) \cap R(t_3))$ and the value of d may change as $d \in W(t_1) \cap W(t_2)$. If we put the write operation of t_2 into the same branch just before t_3, we may change the value of d because t_3 may not fire.

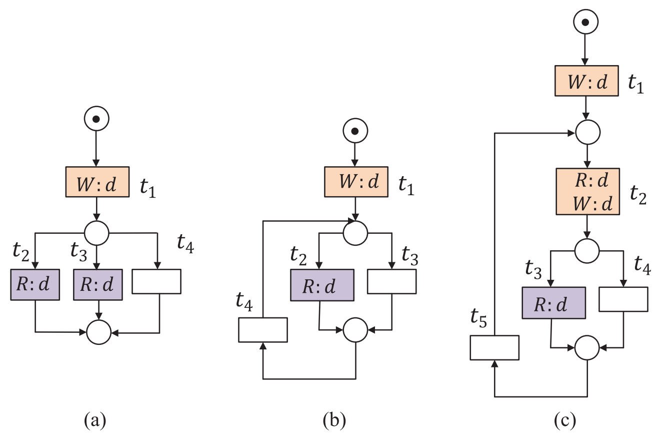


Figure 10: Three kinds of weakly redundant shouldn't be repaired. (a) $d \in W(t_1) \cap R(t_2) \cap R(t_3)$; (b) t_1 is not in a weak circulation relation, while t_2 is in a weak circulation relation; (c) $Type(d) = Vari$

Figs. 3a–3f illustrates the cases of redundant data. We propose Algorithm 4 to repair this kind of data-flow error. Its basic solutions are to remove some undesirable data item d in a write operation and delete operation (i.e., Steps (1)–(5) in Algorithm 4). Meanwhile, we divide the transition that produces d into some branches, the transition with d in a write operation just before the transition read it (i.e., Steps (6)–(9) in Algorithm 4), or not repair (i.e., Steps (10)–(11) in Algorithm 4). After Steps (1)–(5), the redundant data in Figs. 3a–3d and 3f can be repaired. As for the errors in Fig. 3e, we can take Steps (6)–(9) to repair them. Fig. 11 shows the repairing results of redundant data in Fig. 3.

Algorithm 4: Repairing unnecessary redundant data (RD) in a WFD-net system

```
Require: A WFD-net system $(ND, M_0)$, a data item $d \in D$
Ensure: A WFD-net system without any unnecessary redundant data ($RD$).
(1) For each $t' \in \sigma$, where $\sigma \in T^*$ and $t_o \in \sigma$ satisfying $p_o \in t_o^{\cdot}$ Do
(2) If there exists $t \in T$ satisfying $(t \succ t') \wedge (d \in W(t)) \wedge \left[\left(d \lesssim De(t)\right) \vee \left(d \lesssim W(t') \cup De(t')\right)\right] \wedge (d \notin R(t'))$ Then
(3) Remove $d$ from $W(t)$, $De(t)$, and $W(t') \cup De(t')$ if there exist;
(4) End if
(5) End for
```

(Continued)

Algorithm 4: (Continued)

```
(6) If there exist $t, t' \in T$ such that $(d \lessdot RD) \wedge (t \succ t') \wedge [d \in W(t) \cap R(t')] \wedge (d \notin De(t)) \wedge [d \lesssim W(t') \cup De(t')]$ Then
(7) Analyze $d$, $n(R)$, $\varsigma(W(t))$, and $\varsigma(R(t'))$;
(8) If $(n(R) = 1) \wedge [(Type(d) = Cons) \vee ((Type(d) = Vari) \wedge ((t \rightarrow_c t') \vee (t \oplus_{cd} t')))] \wedge [\varsigma(W(t)) \geq \varsigma(R(t'))]$, and there is no $t'' \in T$ such that $(t \succ t'') \wedge (t' \succ t'') \wedge (d \in De(t''))$ Then
(9) Divide $t$ into two transitions $t_k$ and $t_{k'}$, with $t_{k'} \twoheadrightarrow_c t'$, $R(t_k) = R(t) - \Gamma(d)$, $W(t_k) = W(t) - \{d\}$, $De(t_k) = De(t) - \Gamma(d)$, $R(t_{k'}) = \Gamma(d)$, $W(t_{k'}) = \{d\}$, and $De(t_{k'}) = \Gamma(d)$;
(10) Else
(11) Not repair;
(12) End if
(13) End if
```

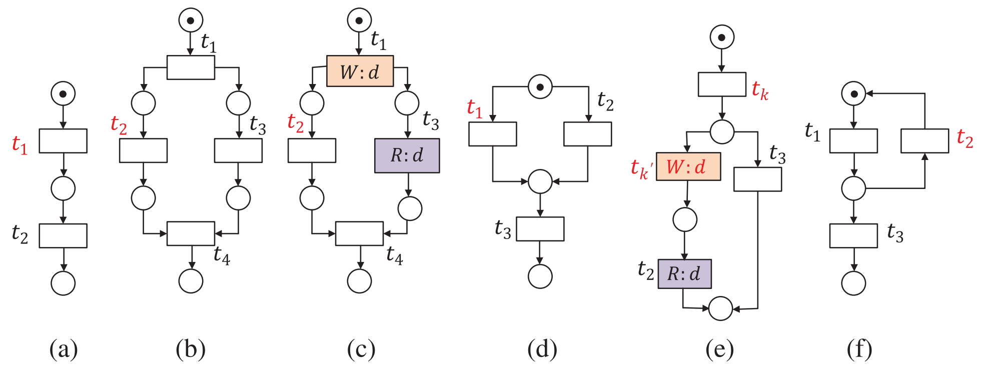


Figure 11: The repairing results of Figs. 3(a)–(f) are shown in (a)–(f)

As we know, the error of inconsistent data exists in weak concurrency relations, and can be avoided by taking a locking mechanism. It needs to be repaired firstly. The errors of redundant data, missing data, and lost data may exist in weak concurrency relations, weak sequence relations, weak exclusiveness relations, and weak circulation relations. For the example of Fig. 4h, we have $d \lessdot WMD$. If $Type(d) = Cons$, we need to add a new transition t_k into T satisfying $(t_k \rightarrow_c t_2) \wedge (t_k \rightarrow_c t_3) \wedge (d \in W(t_k))$ (i.e., Steps (4)–(5) in Algorithm 2), as shown in Fig. 8h1. However, this would bring in weakly redundant data (i.e., $d \lessdot WRD$). Based on the Steps (1)–(36) in Algorithm 2 and Steps (1)–(22) in Algorithm 3, when we repair missing data, we haven't brought in lost data, and vice versa. For the example of Fig. 5a, we have $d \lessdot SLD$. We need to remove the data item d from $W(t_1)$(i.e., Steps (1)–(10) in Algorithm 3). However, after taking this algorithm to repair, there are still exist weakly redundant data (i.e., $d \lessdot WRD$), as shown in Fig. 9a. The lost data belongs to redundant data. After repairing lost data, the redundant data may still exist. Based on Steps (1)–(13) in Algorithm 4, when we repair redundant data, we have not brought in lost data or missing data. As we know, the transitions with missing data have no right to fire, and possibly making WFD-net systems stop at a nonterminal state. In other words, missing data may lead to terminal errors [24]. Compared with missing data, lost data and redundant data are not terminal errors. Therefore, the missing data needs to be repaired secondly and the lost data needs to be repaired thirdly. Based on these analyses, we organize the data-flow errors into a hierarchy, as shown in Fig. 7b. This hierarchy can help us avoid bringing in new unnecessary data-flow errors as far as possible when we repair one kind of error.

5 Case Study

This paper aims to detect and repair data-flow errors, and guarantees the correctness of a WFD-net system. For the example of Fig. 1, we can find that the data item d_7 is a missing data in $R(t_9)$, i.e., $d_7 \lessdot MD$. Obviously, the transition t_9 cannot fire if we do not repair this kind of error.

Based on Definitions 9–12, we analyze the data-flow errors in Fig. 1. By taking Algorithms 2–4, we repair these errors, and their results are shown in Fig. 12.

(1) Data items d_2, d_3, d_8 are not in any data-flow errors;
(2) Due to the fact that $(d_1 \lessdot SMD) \wedge (Type(d) = Cons)$ in $R(t_3)$ and $R(t_7)$, we can remove $d_1 \in De(t_2)$ (Steps (18)–(22) in Algorithm 2);
(3) Since $d_7 \lessdot WMD$ in $R(t_9)$, we can add a transition t_k with $d_7 \in W(t_k)$ and related places as well as arcs in the same branch just before t_9 (Steps (1)–(12) in Algorithm 2);
(4) Given $d_5 \lessdot IND \wedge SLD \wedge SRD$ in $W(t_{13}) \cap W(t_{14}) \cap R(t_{15})$, we can adopt the locking mechanism to avoid of IND and SLD (Steps (16)–(17) in Algorithm 3). Therefore, the data item d_5 read by t_{15} can only come from $W(t_{13})$. After then, we have $d_5 \lessdot SRD$ in $W(t_{14})$, and we should remove $d_5 \in W(t_{14})$(Steps (1)–(5) in Algorithm 4);
(5) Considering $d_6 \lessdot SLD \wedge SRD$ in $W(t_{12}) \cap W(t_{13}) \cap R(t_{14}) \cap R(t_{15})$, we can remove $d_6 \in W(t_{12})$ to avoid SLD and SRD (Steps (1)–(10) in Algorithm 3 or Steps (1)–(5) in Algorithm 4); and
(6) Given $d_4 \lessdot WRD$ in $W(t_3)$, we can divide the transition t_3 into two transitions t_{3k} and $t_{3k'}$. The transition $t_{3k'}$ is in the same branch just before t_4 (Steps (6)–(9) in Algorithm 4).

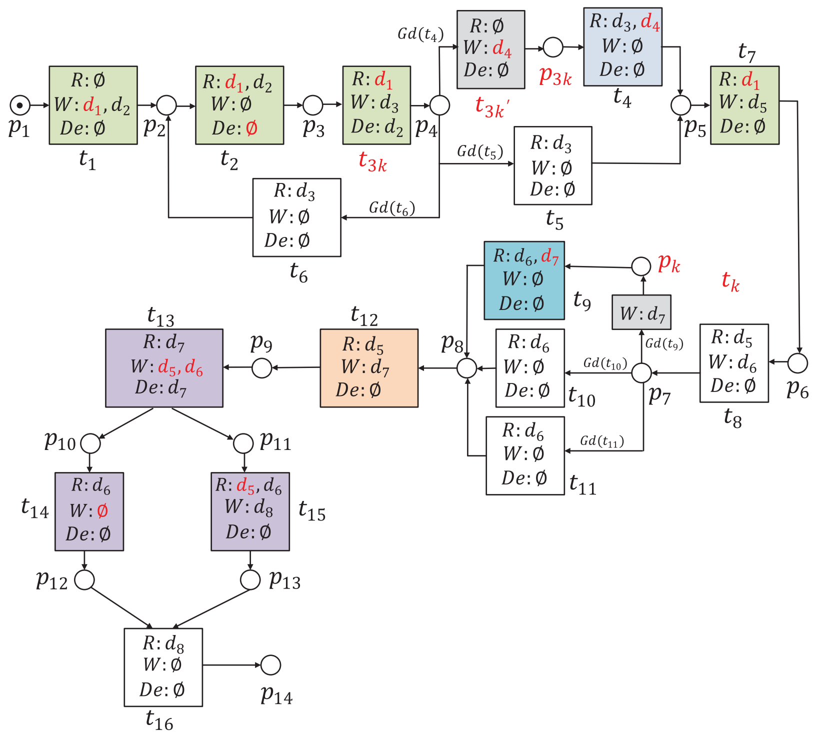


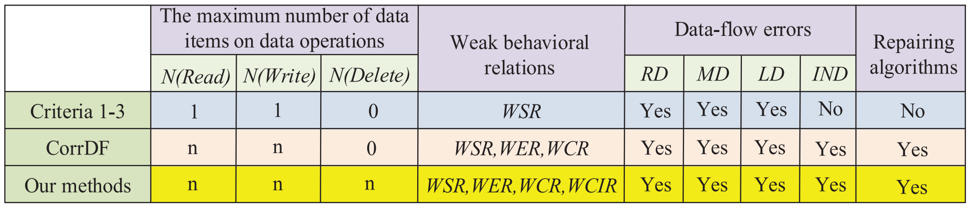
Figure 12: A repaired WFD-net system of Fig. 1

After taking these methods, a repaired WFD-net system is shown in Fig. 12. It is a new WFD-net system without any control-flow or data-flow errors.

6 Evaluation and Experiment

6.1 Functional Evaluation

In this subsection, we compare our methods with some state-of-the-art methods in terms of repairing functions for data-flow errors (Fig. 13). The method of Criterias 1–3 [10] can repair data-flow errors caused by transition pairs only in weak sequence relations, and its transitions read/write at most one data item. The method of CorrDF [14] can repair data-flow errors caused by transition pairs in weak sequence relations, weak exclusiveness relations, or weak concurrency relations. Unfortunately, these methods cannot repair data-flow errors in weak circulation relations and errors caused by delete operations. By comparison, our methods overcome these deficiencies, as shown in Fig. 13.



	The maximum number of data items on data operations			Weak behavioral relations	Data-flow errors				Repairing algorithms
	N(Read)	*N(Write)*	*N(Delete)*		*RD*	*MD*	*LD*	*IND*	
Criteria 1-3	1	1	0	*WSR*	Yes	Yes	Yes	No	No
CorrDF	n	n	0	*WSR, WER, WCR*	Yes	Yes	Yes	Yes	Yes
Our methods	n	n	n	*WSR, WER, WCR, WCIR*	Yes	Yes	Yes	Yes	Yes

Figure 13: A comparison with some state-of-the-art methods (e.g., Criterias 1–3 [10] and CorrDF [14]) in terms of repairing functions for data-flow errors. In this figure, $N(Read)$(resp. $N(Write)$ or $N(Delete)$) denotes the maximum number of data items on the read (resp. write or delete) operations of a transition, where $n \in \mathbb{N}$. WSR denotes weak sequence relation, WER stands for weak exclusiveness relation, WCR represents weak concurrency relation, and $WCIR$ means weak circulation relation

6.2 Experiments

In reality, a simple pseudo-code easily suffers from different kinds of data-flow errors. In this part, some experimental cases are given to show the advantages of our methods. Their detailed procedures are proceeding as follows. Firstly, we use a WFD-net system to model the real pseudo-code. Secondly, we check the data-flow errors based on Algorithm 1. After that, we use Algorithms 2–4 to repair these data-flow errors in a WFD-net system. Finally, we can obtain the repaired WFD-net system. According to the repaired WFD-net system, we can further repair the errors in the pseudo-code.

6.2.1 Repairing Missing Data (MD)

In the first experiment, we use a WFD-net system to model a simple pseudo-code with missing data, as shown in Figs. 14a and 15a. Due to the fact that the data item a read by t_{10} is deleted by t_4, we can find $(a \lessdot MD) \wedge (Type(a) = Cons)$. Therefore, it is important to repair this kind of data-flow error in a suitable position. We utilize Algorithm 2 (Steps (18)–(22)) to do this work, and its result is shown in Fig. 15b. Furthermore, we can repair this pseudo-code according to the WFD-net system without missing data, as shown in Fig. 14b.

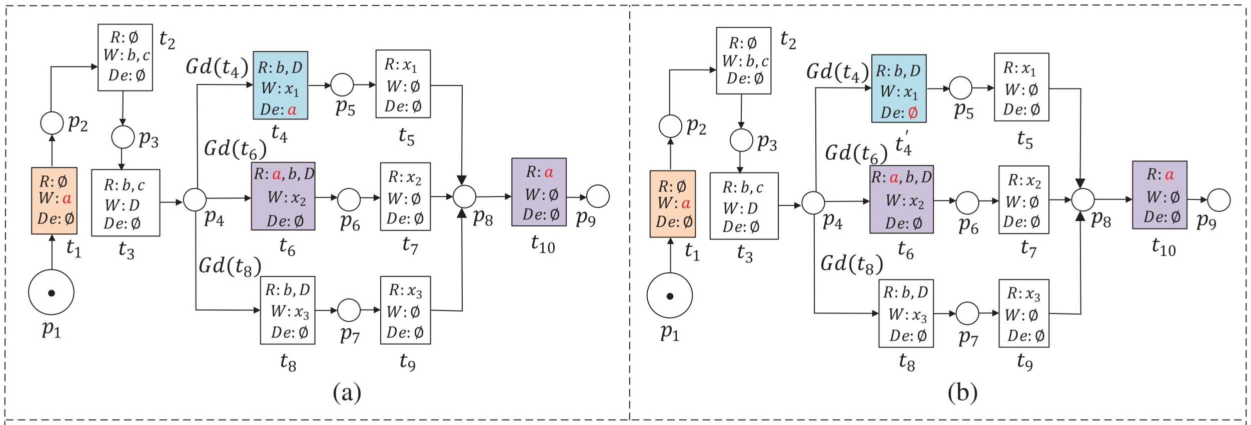
```
Begin
a=3;
Input b and c;
D=b*b-4*c;
If b>=0 and D>=0 Then
x1=D-b and delete(a);
Output x1;
Else if b>=0 and D<0 Then
x2=(D+b)/a;
Output x2;
Else
x3=(D-b)/2;
Output x3;
End if
Output a;
End
```

(a)

```
Begin
a=3;
Input b and c;
D=b*b-4*c;
If b>=0 and D>=0 Then
x1=D-b;
Output x1;
Else if b>=0 and D<0 Then
x2=(D+b)/a;
Output x2;
Else
x3=(D-b)/2;
Output x3;
End if
Output a;
End
```

(b)

Figure 14: (a) Pseudo-code with missing data (*MD*) and (b) after being repaired


$R:\emptyset$ $W:b,c$ $De:\emptyset$ t_2
$Gd(t_4)$ $R:b,D$ $W:x_1$ $De:a$ t_4
p_5 $R:x_1$ $W:\emptyset$ $De:\emptyset$ t_5
p_2 p_3 $Gd(t_6)$
$R:\emptyset$ $W:a$ $De:\emptyset$ t_1
$R:b,c$ $W:D$ $De:\emptyset$ t_3
p_4 $R:a,b,D$ $W:x_2$ $De:\emptyset$ t_6
p_6 $R:x_2$ $W:\emptyset$ $De:\emptyset$ t_7
p_8 $R:a$ $W:\emptyset$ $De:\emptyset$ t_{10} p_9
$Gd(t_8)$ $R:b,D$ $W:x_3$ $De:\emptyset$ t_8
p_7 $R:x_3$ $W:\emptyset$ $De:\emptyset$ t_9
p_1
(a)
$R:b,D$ $W:x_1$ $De:\emptyset$ t_4'
(b)

t_1: a=3;
t_2: Input b and c;
$t_3: D = b * b - 4 * c;$
$t_4: x_1 = D - b$ and delete(a);
t_5: Output x_1;
$t_6: x_2 = (D + b)/a;$
t_7: Output x_2;
$t_8: x_3 = (D - b)/2;$
t_9: Output x_3;
t_{10}: Output a;
$t_4': x_1 = D - b;$
$Gd(t_4): b \geq 0 \text{ and } D \geq 0;$
$Gd(t_6): b \geq 0 \text{ and } D < 0;$
$Gd(t_8): b < 0.$

Figure 15: (a) and (b) are the WFD-net systems of Figs. 14a and 14b

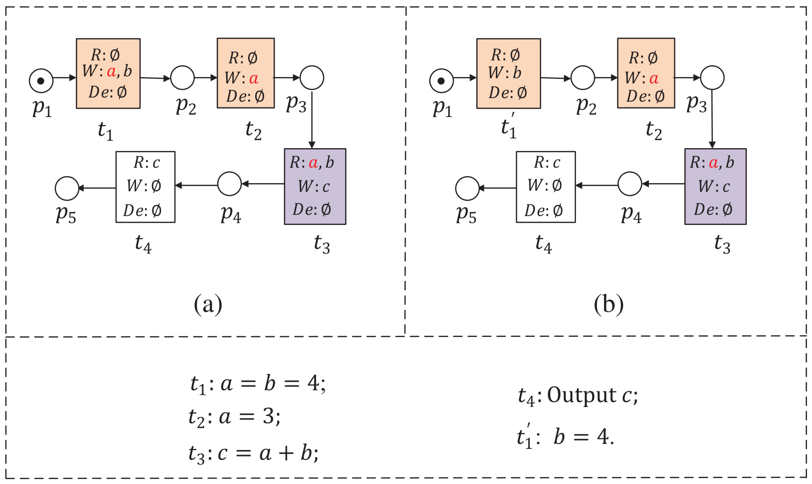
6.2.2 *Repairing Unnecessary Lost Data (LD)*

In the second experiment, we use a WFD-net system to model a simple pseudo-code with unnecessary lost data, as shown in Figs. 16a and 17a. Due to the fact that the data item a is overwritten without being read or deleted first. Therefore, we have $a \lessdot LD$. We utilize Algorithm 3 (Steps (1)–(10)) to do this work, and its result is shown in Fig. 17b. Furthermore, we can repair this pseudo-code according to the WFD-net system without unnecessary lost data, as shown in Fig. 16b.

```
Begin
a=b=4;
a=3;
c=a+b;
Output c;
End
```
(a)

```
Begin
b=4;
a=3;
c=a+b;
Output c;
End
```
(b)

Figure 16: (a) Pseudo-code with Lost data (*LD*) and (b) after being repaired



$R:\emptyset$ $W:a,b$ $De:\emptyset$
p_1 t_1 p_2
$R:\emptyset$ $W:a$ $De:\emptyset$
t_2 p_3
$R:a,b$ $W:c$ $De:\emptyset$ t_3
p_4
$R:c$ $W:\emptyset$ $De:\emptyset$ t_4
p_5
(a)
$R:\emptyset$ $W:b$ $De:\emptyset$ t_1'
(b)
$t_1: a = b = 4;$
$t_2: a = 3;$
$t_3: c = a + b;$
t_4: Output c;
t_1': $b = 4$.

Figure 17: (a) and (b) are the WFD-net systems of Figs. 16a and 16b

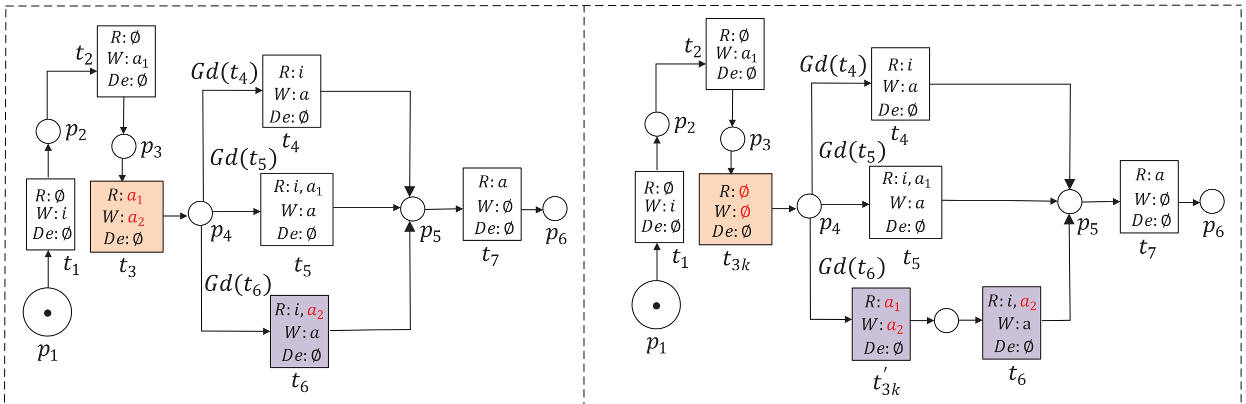
6.2.3 *Repairing Unnecessary Redundant Data (RD)*

In the third experiment, we use a WFD-net system to model a simple pseudo-code with redundant data, as shown in Figs. 18a and 19a. Due to the fact that the data item a_2 wrote by t_3 is only read by t_6, we can find $a_2 \lessdot RD$. We utilize Algorithm 4 (Steps (6)–(9)) to do this work, and its result is shown in Fig. 19b. Furthermore, we can repair this pseudo-code according to the WFD-net system without redundant data, as shown in Fig. 18b.

```
Begin
Input i;
a=1;
a2=a1+6;
If i<=100 Then
a=i*0.1;
Else if 100<i<=200 Then
a=a1+(i-10) *0.75;
Else
a=a2+(i-20) *0.75;
End if
Output a;
End
```
(a)

```
Begin
Input i;
a=1;
If i<=100 Then
a=i*0.1;
Else if 100<i<=200 Then
a=a1+(i-10) *0.75;
Else
a2=a1+6;
a=a2+(i-20) *0.75;
End if
Output a;
End
```
(b)

Figure 18: (a) Pseudo-code with redundant data (*RD*) and (b) after being repaired


t_2 $R:\emptyset$ $W:a_1$ $De:\emptyset$
p_2 p_3
$R:\emptyset$ $W:i$ $De:\emptyset$ t_1
$R:a_1$ $W:a_2$ $De:\emptyset$ t_3
p_1 p_4 p_5 p_6
$Gd(t_4)$ $R:i$ $W:a$ $De:\emptyset$ t_4
$Gd(t_5)$ $R:i,a_1$ $W:a$ $De:\emptyset$ t_5
$Gd(t_6)$ $R:i,a_2$ $W:a$ $De:\emptyset$ t_6
$R:a$ $W:\emptyset$ $De:\emptyset$ t_7
$R:\emptyset$ $W:\emptyset$ $De:\emptyset$ t_{3k}
$R:a_1$ $W:a_2$ $De:\emptyset$ t'_{3k}

(a) A WFD-net system with missing data

(b) A repaired WFD-net system

t_1: Input i;	t_7: Output a;
t_2: $a_1 = 1$;	t_{3k}: τ;
t_3: $a_2 = a_1 + 6$;	t'_{3k}: $a_2 = a_1 + 6$;
t_4: $a = i * 0.1$;	$Gd(t_4)$: $i \leq 100$;
t_5: $a = a_1 + (i - 10) * 0.75$;	$Gd(t_5)$: $100 < i \leq 200$;
t_6: $a = a_2 + (i - 20) * 0.75$;	$Gd(t_6)$: $i > 200$.

Figure 19: (a) and (b) are the WFD-net systems of Figs. 18a and 18b

6.2.4 Repairing Results of Existing Examples

In the following experiment, we consider examples from the existing references. We present the advantages of our methods in repairing four kinds of data-flow errors. Fig. 20 is the result of our experiment. It shows the numbers of data items on read/write/delete operations, the numbers of guards and transitions, and the numbers of data-flow errors before and after repairing. We can find that after taking our Algorithms 2–4, most of the data-flow errors can be repaired. However, as shown in Algorithm 4, some kinds of redundant data should not be repaired, e.g., redundant data item e shown in Fig. 1 in [30]. Therefore, our methods are more effective.

	Number of data items			Number of guards	Number of transitions	Number of data-flow errors before repairing				Repairing Algorithms (Alg)	Number of data-flow errors after repairing			
	Read	*Write*	*Delete*			*RD*	*MD*	*LD*	*IND*		*RD*	*MD*	*LD*	*IND*
Fig.2 in [5]	5	13	2	0	5	9	2	2	0	Alg 2,3,4	0	0	0	0
Fig.3 in [13]	30	14	0	6	11	1	5	0	0	Alg 2,4	0	0	0	0
Fig.1 in [14]	7	6	0	0	5	2	2	0	0	Alg 2,4	0	0	0	0
Fig.1 in [22]	27	20	0	8	13	3	0	1	0	Alg 3,4	0	0	0	0
Fig.1 in [24]	22	18	0	8	11	3	0	1	0	Alg 3,4	0	0	0	0
Fig.1 in [25]	11	6	0	4	9	0	0	0	0	\	0	0	0	0
Fig.1 in [30]	10	15	5	2	8	8	2	5	3	Alg 2,3,4	1	0	0	0

Figure 20: Our experiment result in terms of the numbers of data items on read/write/delete operations, the numbers of guards and transitions, and the numbers of data-flow errors before and after repairing

7 Conclusion and Future Work

WFD-net is a formal functional method to describe the control-/data-flows of workflow systems. It extends WF-net systems with three kinds of data operations (i.e., read, write, and delete operations) and guard functions. A good modeling method and efficient detecting and repairing technique are crucial for the correctness of workflow systems. Based on weak behavioral relations (i.e., weak sequence relation, weak exclusiveness relation, weak circulation relation, and weak concurrency relation) and order relation, we formalize four kinds of data-flow errors (e.g., redundant data, missing data, lost data, and inconsistent data) in a WFD-net system. Then, we reveal the relations between these data-flow errors, and organize them into a hierarchy, which is conducive to correctly repairing data-flow errors without repeated work. Furthermore, some algorithms are developed to detect and repair data-flow errors in WFD-net systems according to system requirements and repair strategies. Compared with the existing methods, our methods can repair data-flow errors in weak circulation relations and errors caused by delete operations. What's more, our methods can avoid bringing in new unnecessary errors when repairing some kinds of data-flow errors.

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

(1) We analyze the behavioral consistency of WFD-net systems, and study what kind of data-flow error affects the behavioral consistency degree;
(2) We develop a tool to repair some unnecessary data-flow errors automatically based on system requirements; and
(3) We intend to consider the process mining with timestamps in the workflow processes, and use the unfolding-based technique [7] to detect and repair their data-flow errors.

Funding Statement: This work was supported in part by the Shanghai Science and Technology Innovation Action Plan (Grant No. 19511101300), in part by the Key Laboratory of EMBEDded System and Service Computing (Ministry of Education) (Grant Nos. ESSCKF201902 and ESSCKF202102), and in part by the National Nature Science Foundation of China (Grant Nos. 62172299 and 62032019).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Davenport, T. H. (1993). *Process innovation: Reengineering work through information technology*. Boston, MA, USA: Harvard Business School Press.
2. Sun, S. X., Zhao, J. L., Nunamaker, J. F. (2006). Formulating the data-flow perspective for business process management. *Information Systems Research, 17(4),* 374–391. DOI 10.1287/isre.1060.0105.
3. Stohr, E., Zhao, J. L. (2001). Workflow automation: Overview and research issues. *Information Systems Frontiers, 3(3),* 281–296. DOI 10.1023/A:1011457324641.
4. Sarnikar, S., Zhao, J. L., Kumar, A. (2004). Organizational knowledge distribution: An experimental evaluation. *Americas Conference on Information Systems,* pp. 2305–2341. New York.
5. Kabbaj, M. I., Abdelkader, B., Bakkoury, Z., Rharbi, A. (2015). Towards an active help on detecting data-flow errors in business process models. *International Journal of Computer Science and Applications, 12(1),* 16–25.
6. Xiang, D. M., Liu, G. J., Yan, C. G., Jiang, C. J. (2018). Detecting data-flow errors based on petri nets with data operations. *IEEE/CAA Journal of Automatica Sinica, 5(1),* 251–260. DOI 10.1109/JAS.2017.7510766.

7. Xiang, D. M., Liu, G. J. (2020). Checking data-flow errors based on the Guard-driven reachability graph of WFD-net. *Computing and Informatics, 39,* 193–212. DOI 10.31577/cai_2020_1-2_193.
8. Xiang, D. M., Liu, G. J., Yan, C. G., Jiang, C. J. (2021). A Guard-driven analysis approach of workflow net with data. *IEEE Transactions on Services Computing, 14(6),* 1650–1661. DOI 10.1109/TSC.2019.2899086.
9. Awad, A., Decker, G., Lohmann, N. (2009). Diagnosing and repairing data anomalies in process models. *International Conference on Business Process Management. Business Process Management Workshops*, pp. 5–16. Germany.
10. Song, W., Ma, X. X., Cheung, S. C., Hu, H., Lü, J. (2010). Preserving data-flow correctness in process adaptation. *IEEE International Conference on Services Computing*, pp. 9–16. Miami, USA.
11. Weidlich, M., Polyvyany, A., Mendling, J., Weske, M. (2011). Causal behavioral profiles-efficient computation, applications, and evaluation. *Fundamental Informaticae, 113(3–4),* 399–435. DOI 10.3233/FI-2011-614.
12. Koehler, J., Hauser, R., Kuster, J. M. (2008). The role of visual modeling and model transformations in business-driven development. *Electronic Notes Theory Computer Science, 211,* 5–15. DOI 10.1016/j.entcs.2008.04.025.
13. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C. (2003). Data flow and validation in workflow modeling. *Conferences in Research and Practice in Information Technology*, pp. 207–214. Dunedin, New Zealand.
14. Sharma, D., Pinjala, S., Sen, A. K. (2014). Correction of data-flow errors in workflows. *25th Australasian Conference on Information Systems*, pp. 1–10. Auckland, New Zealand.
15. Best, E., Wimmel, H. (2013). Structure theory of Petri nets. In: *Transactions on Petri nets and other models of concurrency VII,* vol. 7480, pp. 162–224. DOI 10.1007/978-3-642-38143-0_5.
16. Wang, S. G., Gan, M. D., Zhou, M. C., You, D. (2015). A reduced reachability tree for a class of unbounded petri nets. *IEEE/CAA Journal of Automatica Sinica, 2(4),* 345–352. DOI 10.1109/JAS.2015.7296528.
17. Fang, X. W., Jiang, C. J., Yin, Z. X., Fan, X. Q. (2011). The trustworthiness analyzing of interacting business process based on the induction information. *Computer Science and Information Systems, 8(3),* 843–867. DOI 10.2298/CSIS100411031F.
18. Clempner, J. (2014). Verifying soundness of business processes: A decision process Petri nets approach. *Expert Systems with Applications, 41,* 5030–5040. DOI 10.1016/j.eswa.2014.03.005.
19. Lourenco, J. M., Fiedor, J., Krena, B., Vojnar, T. (2018). Discovering concurrency errors. In: *Lecture on runtime verification,* vol. 10457, pp. 34–60. Germany: Springer.
20. Henkel, M., Zdravkovic, J., Johannesson, P. (2004). *Service-based processes: Design for business and technology,* pp. 21–29. USA: ACM Digital Library.
21. Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., Johannesson, P. (2005). A declarative foundation of process models. *Lecture Notes in Computer Science, 3520,* 233–247. DOI 10.1007/11431855_17.
22. Sundari, M. H., Sen, A. K., Bagchi, A. (2007). Detecting data-flow errors in workflows: A systematic graph traversal approach. *17th Workshop on Information Technology and Systems,* pp. 1–6. Montreal.
23. Sun, S. H., Zhao, J. L. (2008). Developing a workflow design framework based on data-flow analysis. *Proceedings of the 41st Hawaii International Conference on System Sciences,* pp. 7–10. Waikoloa.
24. Meda, H. S., Sen, A. K., Bagchi, A. (2010). On detecting data-flow errors in workflows. *ACM Journal of Data and Information Quality, 2(1),* 1–31. DOI 10.1145/1805286.1805290.
25. Sidorova, N., Stahl, C., Trcka, N. (2011). Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Information Systems, 36,* 1026–1043. DOI 10.1016/j.is.2011.04.004.
26. Haddar, N., Tmar, M., Gargouri, F. (2016). A Data-centric approach to manage business processes. *Computing, 98,* 375–406. DOI 10.1007/s00607-015-0440-2.
27. Dolean, C. C., Petrusel, R. (2012). Data-flow modeling: A survey of issues and approaches. *Informatica Economica, 16(4),* 117–130.
28. Dramski, M. (2017). Missing data problem in the event logs of transport processes. *International Conference on Transport Systems Telematics*, pp. 110–120. Katowice.

29. Trcka, N., van der Aalst, W. M. P., Sidorova, N. (2008). Analyzing control-flow and data-flow in workflow processes in a unified way. *Computer Science Reports,* 1–23. DOI 10.1.1.501.519.
30. Trcka, N., van der Aalst, W. M. P., Sidorova, N. (2009). Data-flow anti-patterns: Discovering data-flow errors in workflows. *International Conference on Advanced Information Systems Engineering*, vol. 5565, pp. 425–439. Amsterdam, Netherlands.
31. von Stackelberg, S., Putze, S., Mvlle, J., Bohm, K. (2014). Detecting data-flow errors in BPMN 2.0. *Open Journal of Information Systems, 1(2),* 1–19.
32. Song, W., Zhang, C., Jacobsen, H. (2018). An empirical study on data flow bugs in business processes. *IEEE Transactions on Cloud Computing, 9(1),* 1–14. DOI 10.1109/TCC.2018.2844247.
33. Mülle, J., Tex, C., Bohm, K. (2019). A practical data-flow verification scheme for business processes. *Information Systems, 81,* 136–151. DOI 10.1016/j.is.2018.12.002.
34. Jovanovikj, I., Yigitbas, E., Gerth, C., Sauer, S., Engels, G. (2019). Detection and resolution of data-flow differences in business process models. *CAiSE Forumn 2019,* vol. 350, pp. 145–157. Germany: Springer.
35. Wang, M. M., Liu, G. J., Zhao, P. H., Yan, C. G., Jiang, C. J. (2018). Behavior consistency computation for workflow nets with unknown correspondence. *IEEE/CAA Journal of Automatica Sinica, 5(1),* 281–291. DOI 10.1109/JAS.2017.7510775.
36. Qi, L., Zhou, M. C., Luan, W. J. (2018). A two-level traffic light control strategy for preventing incident-based urban traffic congestion. *IEEE Transactions on Intelligent Transportation Systems, 19(1),* 13–24. DOI 10.1109/TITS.2016.2625324.
37. Song, W., Chang, Z., Jacobsen, H., Zhang, P. W. (2021). Discovering structural errors from business process event logs. *IEEE Transactions on Knowledge and Data Engineering (Early Access),* 1–14. DOI 10.1109/TKDE.2021.3052927.
38. Wang, S. G., Gan, M. D., Zhou, M. C. (2015). Macro liveness graph and liveness of w-independent unbounded nets. *Science in China Series F: Information Sciences, 58(3),* 1–10. DOI 10.1007/s11432-014-5239-9.
39. Song, W., Jacobsen, H., Chen, F. F. (2019). Scientific workflow protocol discovery from public event logs in clouds. *IEEE Transactions on Knowledge and Data Engineering, 32(12),* 2453–2466. DOI 10.1109/TKDE.2019.2922183.
40. Liu, C., Zeng, Q. T., Cheng, L., Duan, H., Zhou, M. C. et al. (2021). Privacy-preserving behavioral correctness verification of cross-organizational workflow with task synchronization patterns. *IEEE Transactions on Automation Science and Engineering, 18(3),* 1037–1048. DOI 10.1109/TASE.2020.2993376.
41. Zhao, F., Xiang, D. M., Liu, G. J., Jiang, C. J. (2021). A new method for measuring the behavioral consistency degree of WF-net systems. *IEEE Transactions on Computational Social Systems (Early Access),* 1–14. DOI 10.1109/TCSS.2021.3099475.
42. Guang, M. J., Yan, C. G., Wang, J. L., Qi, H. D., Jiang, C. J. (2021). Benchmark datasets for stochastic petri net learning. *International Joint Conference on Neural Networks*, pp. 1–8. Shenzhen, China. DOI 10.1109/IJCNN52387.2021.9533785.
43. Weidlich, M., Mendling, J., Weske, M. (2011). Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering, 37(3),* 410–429. DOI 10.1109/TSE.2010.96.