Locating and Controlling Unsound Transitions in Workflow Systems Based on Workflow Net with Data Constraints

YAQIONG HE\textsuperscript{1,2,3}, GUANJUN LIU\textsuperscript{1,2,3}, (Member, IEEE), CHUNGA N YAN\textsuperscript{1,2,3}, CHANGJUN JIANG\textsuperscript{1,2,3}, AND JIACUN WANG\textsuperscript{4}, (Senior Member, IEEE)

\textsuperscript{1}Key Laboratory of Embedded System and Service Computing, Ministry of Education
\textsuperscript{2}Shanghai Electronic Transactions and Information Service Collaborative Innovation Center
\textsuperscript{3}Department of computer science, Tongji University, Shanghai, 201804
\textsuperscript{4}Computer Science and Software Engineering Department, Monmouth University, West Long Branch NJ 07764, USA

Corresponding author: Changjun Jiang (e-mail: cjjiang@tongji.edu.cn), Guanjun Liu (e-mail: liuguanjun@tongji.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1001804, in part by the National Natural Science Foundation of China under Grant 61332008 and Grant 61572360.

\section*{Abstract}
Data has a great influence on the analysis of the correctness of workflow systems. How to ensure that a system runs without data errors is very important. Workflow nets with data constraints (WFDC-nets) are a kind formal model that is good at specifying data oriented functional requirements in workflow systems. A sound WFDC-net can guarantee that the system is not only functionally correct but also meets all data requirements. Methods and tools for the soundness verification of WFDC-nets have been designed. However, how to locate unsound transitions in a WFDC-net and then effectively control them is still an open topic. This paper attempts to solve the issue and gives methods to optimize unsound workflow systems in design stage. In our solution, we first construct a minimal complete configuration tree (MCC-tree) based on the configuration graph with data constraints (C-graph) of the WFDC-net, and then we define four kinds of paths and algorithms to characterize unsound transitions. Furthermore, a pseudo intermediate set (PIS) is constructed according to different paths and unsound transitions. Based on it, a 4-step controlling strategy is taken to optimize the system and finally makes the controlled system satisfy the data requirements. An e-commerce system is used to illustrate the feasibility and effectiveness of the proposed methods.

\section*{Index Terms} Workflow systems, WFDC-nets, unsound transitions, controlling strategy.

\section*{I. INTRODUCTION}

Data has a great influence on the running of a workflow system. When abnormal data that does not satisfy the functional requirements of a system (i.e., data constraints) is written into the system, the system can face some security risks. For example, a payment request is delivered from an attacker to a merchant in a third-party payment platform \cite{1}. After the attacker pays to the third-party cashier, the merchant will receive a payment notification from the cashier. If the merchant does not verify it, the attacker can modify the amount of the to-be-paid money before the payment is conducted. That will bring a loss in money to the merchant if the attacker reduces the amount. This business lacks an action of checking whether the final payment is equal to the price of goods. Although there is no deadlock or livelock in the control-flows of the system, it is unsound in the view of data-flows. Therefore, the correctness of a workflow system should also consider the satisfaction of data constraints.

Data in a workflow system can be modified or updated through various activities, and should be transferred or stored in view of specific rules. Otherwise, the system has some potential risks suffering from abnormal data \cite{2}, \cite{3}. These specific rules are called data constraints which can be understood as some semantic requirements between functionality and data of a system \cite{4}. Therefore, it is necessary to analyze whether every data constraint is satisfied. To solve this issue, a formal model called workflow net with data constraints (WFDC-net) is proposed in \cite{5}. It adds data constraints into workflow net with data (WFD-net) \cite{6}. Soundness of WFDC-net is proposed that reflects the correctness of control-flows.
and the satisfaction of data constraints. A method is proposed to verify the soundness of WFDC-nets. A so-called C-graph (configuration graph with data constraints) is used to detect soundness since it can represent all possible executing cases of a WFDC-net. However, there are no studies on how to control an unsound workflow system such that the controlled system is sound. The unsound transitions in WFDC-nets remain unsolved. Therefore, lower-cost and high-efficiency methods of locating unsound transitions and controlling them in design stage are deserved to be studied.

In this paper we utilize C-graph to locate those transitions that make a system unsound and then propose a policy to control them such that the controlled system is sound. To find an unsound transition, we transform a C-graph into a C-tree (configuration tree with data constraints) in order to extract five different types of executing paths. The five types of executing paths reflect five different kinds of termination. Based on these paths, the unsound transitions can be obtained. If the workflow system can be guaranteed correct to the maximum extent during the design stage, the risk that may occur during the real-time operation of the system can also be significantly reduced.

The rest of this paper is organized as follows. Section II introduces the related work. Section III introduces basic concepts of WFDC-nets. Section IV proposes a method of locating unsound transitions. Section V presents a strategy of controlling WFDC-nets. Section VI introduces an example of e-commerce transaction system in order to illustrate the usefulness of our methods. Section VII concludes this paper.

II. RELATED WORKS

There are many studies of analyzing the correctness of workflow systems, i.e., formal specifications [7] and model checking techniques [8], [9]. As a formal method, Petri nets can well model and analyze concurrent systems [10], [11]. Workflow nets (WF-nets), which are a class of Petri nets, are suitable to formally describe the business processes of workflow systems [12]. As an important property of WF-nets, the soundness requires that a system can always terminate, and thus guarantees that the system has neither deadlock nor livelock [13].

Data plays an important role in workflow systems, e.g., orders in e-commerce systems [14], addresses in delivery systems [15], and case reports in medical service systems [16]. In these systems, once data abnormalities occur, the consequences will be serious and immeasurable. Abnormal data include conflict data, missing data, inconsistent data and so on [17]. A business process could be correct when only control-flows are considered, but incorrect when data-flows are considered as well. Prototype Petri nets [18], Logical Petri nets [19], and Colored Petri nets [20] are often used to verify the correctness of workflow systems involving data. Some similar studies are also introduced in [21] and [22].

WF-nets-based models are also used to deal with data. Wang et al. [23] propose a Resource-Oriented Workflow nets (ROWN) to analyze workflow resource requirements. Zeng et al. [16] regard data as resource and message, and extend WF-nets to Resource and Message Workflow nets (RM-WF-nets) in order to formalize a medical system with different departments. To detect data-flow errors, Sidorova et al. [15] add data elements and functions (e.g., read, write and delete) into WF-nets and thus form workflow nets with data (WF-D-nets). Wang et al. [24] refine WFD-net by considering conditions of firing transitions. The soundness of these models is also redefined.

Although many models are provided to detect the errors in data-flows, the value of data is still not considered. Some errors of business processes can be reflected only if the value of data are taken into account. Yu et al. [25] focus on data in the e-commerce transaction process, and propose e-commerce business process net (EBPN) to validate rationality and transaction consistency. But the shortcoming is that they do not consider data constraints and thus their study is limited to some specific data problems and application areas.

WFDC-nets are a general analysis method of verifying the correctness of workflow systems by considering data constraints. Four levels of soundness including soundness, control-soundness, data-soundness and non-soundness are proposed in order to reflect different requirements of correctness. WFDC-nets can be used to analyze not only the structure of a workflow system but also the data abnormality. However, they do not solve the problem: how to locate and control the unsound transitions caused by abnormal data?

The current studies for locating and controlling unsound transitions mainly focus on control-flows, such as discovering vulnerabilities in code-level [26], fault diagnosis in discrete event systems [27], fault locating [28], [29] and deadlock control [30], [31] in system structure without data, and repairing consistency in processing mining [32], [33]. There is a lack of methods for locating unsound transitions and controlling them that specifically focus on data-flows with abnormal data.

III. BASIC CONCEPTS

In this section we introduce WFDC-nets. WFDC-nets are based on the WFD-nets [15].

Definition 3.1. A 9-tuple $WD = (P, T, F, D, Rd, Wt, De, grd, G_{11})$ is a workflow net with data (WFD-net) where

1. $(P, T, F)$ is a WF-net;
2. $D$ is a finite set of data elements;
3. $Rd:T \rightarrow 2^D$, $Wt:T \rightarrow 2^D$ and $De:T \rightarrow 2^D$ are label functions of reading data, writing data and deleting data; and
4. $grd:T \rightarrow G_{11}$ assigns a guard in $G_{11}$ to each transition $t \in T$, where $G_{11}$ is a set of guards.

$Rd$, $Wt$ and $De$ represent read, write and delete operations, respectively. A WFD-net extends a workflow net with conceptual data operations on the refined data including read, write (or update) and delete. A refined data is a conceptual notion in a business process that a concrete specification can be substituted for an abstract one if its behavior is consistent.
with the abstract one. Update operation is a special write operation since they both change the value of data. We use these refined data operations to only focus on the function of a system and ignore the data repository. However, these refined data do not reflect the values of data.

Parallel execution of Rd, Wt and/or De may bring data inconsistency [34]. Our methods mainly focus on repairing a workflow system and prevent abnormal value of data from occurring. Therefore, in this paper we assume that the workflow systems we study on have no data inconsistency.

WFDC-nets consider not only routing paths but also data constraints. We assume that $Ψ = \{ψ^d_i | i ∈ N_k, d ∈ D\}$ is a formula set of unary atomic propositions where $N_k = \{1, 2, \cdots, k\}$. $Π ⊆ Ψ$ includes all atomic proposition formulas (e.g., predicates) in guards and $Φ ⊆ Ψ$ is a data constraint set. For convenience, we denote $Π = \{ψ^m_n | n ∈ N_k, d ∈ D\}$ and $Φ = \{ψ^d_ℓ | n ∈ N_k, d ∈ D\}$ where $k_1 ≤ k$ and $k_2 ≤ k$. For any $d ∈ D$, if $ψ^d_ℓ = ψ^m_n$ and $ψ^d_ℓ = ψ^m_n$, then $ψ^d_ℓ = ψ^m_n$.

**Definition 3.2.** A triple Pa($Φ$) = ($D$, $Φ$, $Γ_D$) is a constraint pattern where

1. $D$ is the set of data elements;
2. $Φ$ is the set of data constraints; and
3. $Γ_D : D → 2^Φ$ maps each data element to a subset of $Φ$.

**Definition 3.3 (WFDC-net).** An ordered pair $N = (WD, Pa(Φ))$ is a workflow net with data constraints (WFDC-net) where

1. $WD$ is a WFD-net; and
2. $Pa(Φ)$ is a constraint pattern.

An example of WFDC-net is shown in Fig. 1(a) which includes one data element $a$. We assume that the system has one data constraint $ψ_x$ w.r.t. $a$, i.e., $Φ = \{ψ_x\}$ and $Γ_D(a) = \{ψ_x\}$, the value of $a$ should satisfy the data constraint $ψ_x$. $Π = \emptyset$ means that there is no guard in this WFDC-net to prevent the abnormal data. When firing $t_3$, a value is written into $a$ . When firing $t_4$, the value of $a$ is deleted.

Before introducing configuration, some notations need to be formalized.

- $ρ_D : D → \{T, \bot\}$ denotes that each $d ∈ D$ is assigned with $T$ or $\bot$ where $T$ means a defined value and $\bot$ means an undefined value.
- $ρ_Π : π → \{\text{TRUE, FALSE, } \bot\}$ denotes that each $π ∈ Π$ is assigned with $\text{TRUE, FALSE, } \bot$ (undefined).

$ℓ_G : G_Π → 2^Π$ is a mapping function where $ℓ_G(\text{grd})$ represents all atomic formulas associated with guard $\text{grd}$.

It is assumed that for each $d ∈ D$, if $∃π ∈ Π : ℓ_Π(π) = d \land ρ_D(d) = \bot$, then $ρ_Π(π) = \bot$. Similarly, for each $d ∈ D$, if $∃ϕ ∈ Φ : ϕ = d \land ρ_D(d) = \bot$, then $ρ_Φ(ϕ) = \bot$.

A triple $ρ = (ρ_D, ρ_Φ, ρ_Π)$ is a data state. All data states generated during system running form a data state set $P$.

**Definition 3.4 (Configuration).** Let $N = (WD, Pa(Φ))$ be a WFDC-net. $c = (M, ρ) = (M, ρ_D, ρ_Φ, ρ_Π)$ is a configuration of $N$, where

1. $M$ is a marking of $WD$; and
2. $ρ_Φ : Φ → \{\text{TRUE, FALSE, } \bot\}$ and $ρ_Π : Π → \{\text{TRUE, FALSE, } \bot\}$ are defined above.

The initial configuration of $N$ is denoted as $c_0 = ([i], ρ_D, ρ_Φ, ρ_Π)$ where $∀d ∈ D, ∀ϕ ∈ Φ, ∀π ∈ Π : ρ_D(d) = \bot \land ρ_Φ(ϕ) = \bot \land ρ_Π(π) = \bot$. The terminal configuration set is denoted as $C_f = \{([o], ρ_D, ρ_Φ) | ρ_Π ∈ P\}$.

**Definition 3.5 (Enabling Conditions).** Let $N = (WD, Pa(Φ))$ be a WFDC-net. A transition $t ∈ T$ is enabled at the configuration $c = (M, ρ_D, ρ_Φ, ρ_Π)$, denoted as $c[t]$, if

1. $M[t]$;
2. $∀d ∈ Rd(t), De(t) : ρ_D(d) = T$;
3. $∀ϕ ∈ ϵ_G(\text{grd}(t)) : ρ_Φ(ϕ) \neq \bot$; and
4. $ρ_Π(\text{grd}(t)) = \text{TRUE}$.

**Definition 3.6 (Firing Rule).** Let $N = (WD, Pa(Φ))$ be a WFDC-net and $t ∈ T$ be an enabled transition at $c = (M, ρ_D, ρ_Φ, ρ_Π)$. A set of configurations $C'$ is generated after firing $t$ where $C' = \{(M', ρ_D', ρ_Φ', ρ_Π') | M[t]^M'$

$\land \forall d ∈ Wt(t) : ρ_D'(d) = T$

$\land \forall d ∈ De(t) : ρ_D'(d) = \bot$

$\land \forall d ∈ D \setminus (Wt(t) \cup De(t)) : ρ_D'(d) = ρ_D(d)$

$\land \forall ϕ ∈ Φ : ϕ ∈ ϵ_G(\text{grd}(t)) \Rightarrow ρ_Φ'(ϕ) = ρ_Φ(ϕ) ∈ \{\text{TRUE, FALSE}\}$

$\land \forall ϕ ∈ Φ : ϕ ∈ De(t) \Rightarrow ρ_Φ'(ϕ) = ρ_Φ(ϕ) = \bot$

$\land \forall ϕ ∈ Φ : ϕ ∈ D \setminus (Wt(t) \cup De(t)) \Rightarrow ρ_Φ'(ϕ) = ρ_Φ(ϕ)$

---

**FIGURE 1.** An example of WFDC-net (a) and its C-graph (b).
\(\forall \pi \in \Pi : (\forall d \in \ell_{\Pi}(\pi) \cap W(t) : \Gamma_D(d) = \emptyset) \Rightarrow \rho_0'(\pi) \in \{\text{TRUE}, \text{FALSE}\}\)
\(\land (\forall \pi \in \Pi : (\forall d \in \ell_{\Pi}(\pi) \cap W(t) : \Gamma_D(d) \neq \emptyset) \Rightarrow \rho_0'(\pi) = \omega)\)
\(\land (\forall \pi \in \Pi : f_{\Pi}(\pi) \in D e(t) \Rightarrow \rho_0'(\pi) = \bot)\)
\(\land (\forall \pi \in \Pi : \ell_{\Pi}(\pi) \in D \setminus (W(t) \cup D e(t)) \Rightarrow \rho_0'(\pi) = \rho_0(\pi))\)

where 
\[
\omega \in \begin{cases} 
\{\text{TRUE}\}, & \text{if } \exists \varphi \in \Gamma_D(d) : \pi = \varphi \\
\{\text{FALSE}\}, & \text{if } \exists \varphi \in \Gamma_D(d) : \pi = \varphi \\
\{\text{TRUE}, \text{FALSE}\}, & \text{if } \exists \varphi \in \Gamma_D(d) : \pi \neq \varphi.
\end{cases}
\]

It is denoted by \(c[t]C'\).

The firing rules of WFDC-nets focus on the changing of four elements in a configuration: marking, data elements, data constraints and predicates. Let \(t\) be an enabled transition at configuration \(c\). After \(t\) is fired:

1. for marking \(M'\), it is obtained according to firing rules of WF-\(\pi\)-nets;
2. for \(\rho_D'(d)\), when write, delete or read is operated on data \(d\), \(\rho_D(d)\) is changed to \(\top, \bot\), or remains unchanged, respectively;
3. for \(\rho_{\Phi}'(\varphi)\), if the related data is written, then \(\rho_{\Phi}'(\varphi)\) belongs to \(\text{TRUE}\) or \(\text{FALSE}\); if the related data is deleted, then \(\rho_{\Phi}'(\varphi)\) is assigned with \(\bot\); if the related data is read, then the value of \(\varphi\) remains unchanged; and
4. for \(\rho_0'(\pi)\), if the related data \(d\) is written and \(\Gamma_D(d) = \emptyset\), then \(\rho_0'(\pi)\) belongs to \(\text{TRUE}\) or \(\text{FALSE}\); if \(d\) is written and \(\Gamma_D(d) \neq \emptyset\), then \(\rho_0'(\pi)\) is the same with \(\rho_0'(\varphi)\); if \(d\) is deleted, then \(\rho_0'(\pi)\) is changed to \(\bot\); if \(d\) is read, then the value of \(\varphi\) remains unchanged when \(d\) read.

**Definition 3.7 (Reachability).** Let \(N = (W D, Pa(\Phi))\) be a WFDC-net, \(c, c'\) and \(c''\) are configurations of \(N\). \(C, C'\) and \(C''\) are configuration sets of \(N\).

1. A **may-step** exists from \(c\) to \(c'\) if \(\exists t \in T : c[t]c\)\. This is denoted as \(c \xrightarrow{\text{may}} c'\).
2. \(c''\) is **may-reachable** from \(c\) if there is a configuration sequence \(c_0, \ldots, c_t, \ldots, c_n\) such that \(c_i \xrightarrow{\text{may}} c_{i+1}\), where \(0 \leq i < n\), \(c_0 = c\) and \(c_n = c''\). This is denoted as \(c \xrightarrow{\text{may}} \top c''\).
3. A **must-step** exists from \(c\) to \(C\) if \(\exists t \in T : c[t]c' \land c' \in C\). This is denoted by \(c \xrightarrow{\text{must}} C\). Furthermore, a must-step also exists from \(C\) to \(C''\), denoted by \(C \xrightarrow{\text{must}} C''\), if \(C'' = \bigcup_{c \in C} C_c\) where \(C \xrightarrow{\text{must}} C_c\) or \(C_c = \{c\}\) (if \(c\) is a dead configuration).
4. \(C''\) is **must-reachable** from \(C\) if there is a sequence of configurations \(C_0, C_1, \ldots, C_n\) of \(N\) such that \(C_i \xrightarrow{\text{must}} C_{i+1}\), where \(0 \leq i < n\), \(C_0 = C\) and \(C_n = C''\). It is denoted by \(C \xrightarrow{\text{must}} C''\).

The set of may-reachable configurations from \(c_0\) is denoted by \(R(c_0)\). The configuration graph with data constraints is defined based on the definitions above:

**Definition 3.8 (C-graph).** Let \(N = (W D, Pa(\Phi))\) be a WFDC-net. A triple \(G = (C, \mathcal{E}, s)\) is a configuration graph with data constraints (C-graph) of \(N\) where

1. \(\mathcal{C} = R(c_0)\);
2. \(\mathcal{E} = \{(c, c') \mid c \in C \land c' \in 2^{R(c_0)} \land (\exists t \in T : c[t] c')\}\); and
3. \(s : \mathcal{E} \rightarrow T\) satisfies that \((c, c') \in \mathcal{E} \land c[t] C'\), then \(s(c, c') = t\).

Note that a configuration graph with data constraints is denoted as CDC-graph in \([5]\), and for convenience we denote it as C-graph in this paper. Fig. 1(b) is the C-graph of the WFDC-net in Fig. 1(a). A branched arrow with multiple heads represents an arc connecting a configuration with its successors. The initial configuration is \(c_0 = \{\text{start}\}, \{\rho_D(a) = \top\}, \{\rho_{\Phi} (\psi_\varphi) = \top, \rho_0 = \emptyset\}\). It satisfies the firing conditions of \(t_1\). After firing \(t_3\) and then \(t_2\), data \(a\) is written. Since there is data constraint \(\psi_\varphi\) w.r.t. \(a\), we have that \(\rho_D(a) = \top\) and \(\rho_{\Phi}(\psi_\varphi) \in \{\text{TRUE}, \text{FALSE}\}\), which result in two successors \(c_2\) and \(c_3\).

Based on the C-graph, must-/may-termination and normal/abnormal termination are defined in order to respectively reflect the correctness of control-flows and data-flows.

**Definition 3.9 (Must-/May-termination).** Let \(N = (W D, Pa(\Phi))\) be a WFDC-net and \(G = (C, \mathcal{E}, s)\) be a C-graph of \(N\). \(c_0\) and \(C_f\) are the initial configuration and the terminal configuration set, respectively. \(N\) is **must-terminated** if \(G\) satisfies

1. \(\forall c \in R(c_0), \exists C \subseteq C_f : c \xrightarrow{\text{must}} C \land C \subseteq C_f\); and
2. \(\forall c \in R(c_0) : M \geq [a] \Rightarrow M = [a]\) where \(M\) is the marking of \(c\); and
3. \(\forall t \in T, \exists c \in R(c_0) : c[t]\).

\(N\) is **may-terminated** if \(G\) does not satisfy the above conditions.

**Definition 3.10 (Normal and abnormal termination).** Let \(N = (W D, Pa(\Phi))\) be a WFDC-net and \(G = (C, \mathcal{E}, s)\) be the C-graph of \(N\). \(C_f\) be the set of terminal configurations of \(N\). \(N\) satisfies **normal termination** if for each \(d \in D\), \(G\) satisfies the following two conditions:

1. \(\forall c \in R(c_0), \forall \varphi \in \Gamma_D(d) : c \xrightarrow{\text{must}} C \land C \subseteq C_f, \text{ then we have that}\)
   1. \(\forall c \in C : \varphi \in \{\text{TRUE}, \bot\}; \text{ and}\)
   2. \(\exists c' \in C : \varphi \in \{\text{TRUE}, \bot\}\).
2. \(\forall c \in R(c_0), \forall \varphi \in \Gamma_D(d) : \rho_{\Phi}(\varphi) = \text{FALSE}, \text{ then we have that}\)
   1. \(\exists c' \in C : \varphi \in \{\text{TRUE}, \bot\}\).

\(N\) satisfies **abnormal termination** if \(G\) does not satisfy the above conditions.

Based on terminations about control-flows and data-flows, the hierarchical soundness is defined.

**Definition 3.11 (Soundness).** Let \(N = (W D, Pa(\Phi))\) be a WFDC-net and \(G = (C, \mathcal{E}, s)\) be the C-graph of \(N\). \(N\) is **sound** if \(G\) satisfies must-termination and normal termination;
(2) control-sound if $G$ satisfies must-termination;
(3) data-sound if $G$ satisfies normal termination; or
(4) non-sound if $G$ satisfies may-termination and abnor-
mal termination.

The C-graph in Fig. [1]b) satisfies must-termination but
does not satisfy normal termination because $\rho_\phi(\psi_\phi) =\nFALSE$ at the terminal configuration. As a result, this example
is not sound in data-flow.

IV. LOCATING UNSOUND COMPONENTS BASED ON
C-TREES

After verifying a system, it is also necessary to provide
a solution for incorrectness if the system is not sound. In
this paper, we propose a method to locate those unsound
transitions (i.e., transitions) of WFDC-nets.

A. C-TREES

In order to find out all unsound transitions, we must analyze
every path. Usually, a system is complicated due to the
existence of circuits. These circuits can generate the same
running status. In order to look for unsound transitions, we
must avoid to check circuits repeatedly. In other words,
circuits in a C-graph are needed to be removed while the
properties of C-graph are still preserved. In this paper, the
process of removing circuits in a C-graph is called circuit-
removing. After circuit-removing, a C-tree is generated.

C-graphs and C-trees are all directed graphs according to
their structure. Therefore, before defining C-trees and their
properties, we first introduce some notations which are based
on directed graph.

Let $a = (v, v')$ and $a' = (v', v''')$ be two continuous
arcs in a directed graph where $v$ and $v''$ are vertices. We
denote that $\text{input}(a) = v$, $\text{output}(a) = \text{input}(a') = v'$,
$\text{output}(a') = v''$; $a \in \text{input}(v')$, $a' \in \text{input}(v'')$, $a \in
\text{output}(v)$, $a' \in \text{output}(v'')$; $v' \in v''$; $v' \in v''$; $v'' \in v'''$; $v' \in v''$; and $a \in a'$, $a \in a''$.

A path is a continuous arc sequence connecting vertex $v$
and vertex $v'$, and is denoted as $\text{path}(v, v')$. A circuit is a
path satisfying that its first and last vertices are the same and
the rest are different from each other [35].

Definition 4.1 (Re-circuit path). Let $\text{path}(v, v') = a_1 \cdots
a_k \cdots a_n$, where $a_k$ is an arc, $k \leq n$, $n \in \mathbb{N}^+$,
$a_1 \in \text{input}(v)$, and $a_n \in \text{input}(v')$. If there exists
a sequence $a_i \cdots a_j a_{j+1} \cdots a_{2j-i} a_{2j-i+1}$ such that $a_i \cdots a_j = a_{j+1} \cdots a_{2j-i}$ and $1 \leq i < j \leq \frac{n-1}{2}$,
then it is a re-circuit path and denoted as $\text{path}_{rc}(v, v')$; if a path does not
satisfy the condition above, it is not a re-circuit path, denoted as
$\text{path}_{nc}(v, v')$.

For example in Fig. [1]b), $t_1 t_2 t_4 t_2 t_4$ is a re-circuit path
and $t_1 t_2 t_4$ is not a re-circuit path. The circuit-removing pro-
cess of $\text{path}(v, v')$ is to generate a path in which the order
of arcs is the same as $\text{path}(v, v')$, but there are no subsequences
continuously repeated in it.

Definition 4.2. Let $\text{path}(v, v')$ be a re-circuit path, then

Step 1: Let $\text{path}(v, v')' := \text{path}(v, v')$;
Step 2: Assume that the repeated subsequence is $\sigma = a_1 \cdots a_i \cdots a_j a_{j+1} \cdots a_{2j-i} \cdots a_n$ and $1 \leq i < j \leq \frac{(n-1)+i}{2}$.
Let $\text{path}(v, v')'' := a_1 \cdots a_i \cdots a_{j+1} \cdots a_{2j-i+2} \cdots a_n$ and $n \geq -(j-i+1)$;
Step 3: If $\text{path}(v, v')''$ is still a re-circuit path, then repeat
step 2; otherwise, the process ends.

The circuit-removing process is denoted as $\text{Cr}(\text{path}(v, v'))$.
For vertices $v$ and $v'$.

The set of all paths from $v$ to $v'$ is denoted as $\text{PATH}(v, v')$. Among them all re-circuit paths comprise
a set of re-circuit path, denoted as $\text{PATH}_{rc}(v, v')$, and
$\text{PATH}_{nc}(v, v')$ denoted all paths that are not re-circuit paths.
We can easily get the follow proposition based on Definition
4.2.

Proposition 4.1. Let $v$ and $v'$ be two vertices in a directed
graph, then we have the following conclusions:

(1) If there is a re-circuit path $\text{path}_{rc}(v, v')$, then
there exists $\text{path}_{nc}(v, v')$ such that $\text{path}_{nc}(v, v') =
\text{Cr}(\text{path}_{rc}(v, v'))$;
(2) $\text{PATH}(v, v') = \text{PATH}_{rc}(v, v') \cup \text{PATH}_{nc}(v, v')$;
and
(3) $\text{Cr}(\text{PATH}_{rc}(v, v')) \subseteq \text{PATH}_{nc}(v, v')$.

Based on the notations above, a C-tree is defined as fol-
ows.

Definition 4.3 (C-tree). Let $G = (C, E, s)$ be a C-graph.
$\text{CT} = (C, E, s, \delta)$ is a configuration tree with data constraints
(C-tree) of $G$ if

(1) $C$ is the set of nodes where each node is a configuration
and $\forall c \in C : |c| \leq 1$;
(2) $E$ is the set of arcs; and
(3) $\delta : C \cup E \rightarrow G$ is a mapping function that satisfies
$\delta(c) \subseteq C$ and $\delta(e) \subseteq E$. ($\delta$ preserves the nature of nodes
and arcs, i.e., $\forall e \in E, \delta(e) \in C$ and $\forall e \in E, \delta(e) \in E$.)

Notice that in the normal situation, a transition sequence
is by default to represent a path in a Petri net or a high-
level Petri net, as shown in Fig. [1]b). However, due to the
special form of C-graph, there may be the same transitions
on multiple directed arcs, but the input and output config-
urations of these directed arcs are different due to different
data states. For example, in Fig. [1]b), $s(c_1, c_2) = s(c_1, c_4) = s(c_2, c_4) = s(c_3, c_5)$ = $s_3$, but $c_2 \neq c_3$ and
$c_4 \neq c_5$. Therefore, we can not know which one the path
$t_2 t_3$ represents. In order to reflect different sequences, we
use arc sequences instead of transition sequences to represent
paths in this paper. For example, directed arcs $e_2 = (c_1, c_2)$,
$e_3 = (c_1, c_3)$, $e_4 = (c_2, c_4)$ and $e_5 = (c_3, c_5)$, then the paths
$e_2 e_4$ and $e_3 e_5$ can be distinguished.

For instance, Fig. [2]a) is a simplified C-graph correspond-
ing to Fig. [1]b), and Fig. [2]b) is one C-tree that is generat-
ed. In this figure, a configuration with a double-line circle
represents a terminal configuration. A C-tree is directed and
acyclic. A C-graph can be unfolded into different C-trees ac-
according to different unfolding and cycle-removing. The minimal C-tree is the tree with only one configuration $c_0$ such that $\delta(c_0) = c_0$. Three different C-trees are shown in Fig. 2(b), (c) and (d) in which we mark arcs and configurations using the corresponding ones in C-graph for simplification. Through C-trees, we can verify the reachability of a WFDC-net.

**Definition 4.4.** Let $a_1$ and $a_2$ are two arcs in a directed graph. $\sigma$ is an arc sequence. If there exists an arc sequence $a_1\sigma a_2$, then $(a_1, a_2)$ is in a casual order relationship that is denoted as $a_1 \prec a_2$. If $a_1 = a_2$, then $a_1 \prec a_2$.

According to Definition 4.4, we can easily draw the following property in a C-tree:

**Property 4.1.** Let $CT = (C, E, \delta)$ be a C-tree, then we have that $\forall e_1, e_2, e_3 \in E:
\begin{align*}
(1) & \quad e_1 \prec e_2 \land e_2 \prec e_3 \Rightarrow e_1 \prec e_3; \\
(2) & \quad e_1 \prec e_2 \land e_2 \prec e_1 \Rightarrow e_1 = e_2; \text{ and} \\
(3) & \quad e_1 \prec e_2 \Rightarrow \delta(e_1) \prec \delta(e_2).
\end{align*}$

**Theorem 4.1.** Let $CT = (C, E, \delta)$ be a C-tree. $\tilde{c}$ and $\tilde{c}'$ are configurations in $CT$. $\tilde{C}, \tilde{C}'$ and $\tilde{C}''$ are sets of configurations.

(1) $\tilde{c}'$ is may-reachable from $\tilde{c}$ if $\forall \tilde{e} \in \tilde{C} : \tilde{c} \in \text{output}(\tilde{e}) \Rightarrow (\exists \tilde{e}' \in \tilde{E} : \tilde{c} \prec \tilde{e}' \land \text{output}(\tilde{e}') = \tilde{c}')$;

(2) $\tilde{C}''$ is must-reachable from $\tilde{C}$ if $\tilde{C}'' = \bigcup_{\tilde{c} \in \tilde{C}} \tilde{C}'$ where $\forall \tilde{e} \in \tilde{E} : \tilde{c} = \text{output}(\tilde{e}) \Rightarrow \exists \tilde{e}' \in \tilde{E}, \forall \tilde{e}' \in \tilde{C}', \tilde{e} \prec \tilde{e}' \land \text{output}(\tilde{e}') = \tilde{c}'$.

Proof: (1) Assume that $\tilde{c} = \tilde{c}'$. Obviously a may-step $\tilde{c} \rightarrow_{\text{may}} \tilde{c}'$ exists since $\exists s(\delta(\tilde{e})) \in T : \tilde{c}(s(\delta(\tilde{e}))) \tilde{c}'$ where $\tilde{e} = \text{input}(\tilde{e})$ and $\tilde{c}' = \text{output}(\tilde{e})$. Assume that $\tilde{c} \neq \tilde{c}'$. Then, according to Definition 4.4, there exists a sequence of configurations satisfying input($\tilde{e}$), $\cdots$, $\tilde{c}_i$, $\cdots$, output($\tilde{e}'$), $0 \leq i \leq n$. Because $\text{input}(\tilde{e}) = \tilde{c}$ and $\text{output}(\tilde{e}') = \tilde{c}'$, we can get $\tilde{c} \rightarrow_{\text{may}} \tilde{c}'$ according to Definition 3.7, i.e., $\tilde{c}'$ is may-reachable from $\tilde{c}$.

(2) Assume that $\forall \tilde{e}' \in \tilde{E}' : \text{input}(\tilde{e}') \in \tilde{C}$. Then we have that $\forall \tilde{e} \in \tilde{E} : \tilde{c} = \text{input}(\tilde{e})$ and $\tilde{c}' = \text{output}(\tilde{e})$. According to Definition 3.7, there exists a must-step $\tilde{c} \rightarrow_{\text{must}} \tilde{C}'$ since $\exists \tilde{e} \in \tilde{E} : \tilde{c}(s(\delta(\tilde{e}))) \tilde{c}' \land \tilde{c}' \in \tilde{C}'$. Therefore, $\tilde{C}'' = \bigcup_{\tilde{c} \in \tilde{C}} \tilde{C}'$ is a must-step from $\tilde{C}$ (denoted as $\tilde{C} \rightarrow_{\text{must}} \tilde{C}'$) since $\forall \tilde{e} \in \tilde{E} : \tilde{c} \rightarrow_{\text{must}} \tilde{C}'$. $\tilde{C}''$ is must-reachable from $\tilde{C}$.

If $\exists \tilde{e}' \in \tilde{E}' : \text{input}(\tilde{e}') \notin \tilde{C}$, then we can obtain a sequence set $\Sigma = \{\sigma | \sigma = \tilde{e}' \land \text{input}(\tilde{e}) \in \tilde{C} \land \text{output}(\tilde{e}') \in \tilde{C}''\}$ according to Definition 4.4, we let $\tilde{C} = \tilde{C}_0$.
then we get a configuration set sequence $\hat{C}_0, \ldots, \hat{C}_i, \ldots, \hat{C}_n$, such that $\hat{C}_i \rightarrow_{\text{must}} \hat{C}_{i+1}$, where $0 \leq i \leq n$. Therefore, according to Definition 3.7, $\hat{C}''$ is must-reachable from $\hat{C}$.

In a C-tree, a configuration is may-reachable from $\hat{c}$ if it is in a path that starts from $\hat{c}$. The set of all configurations may-reachable from $\hat{c}$ is denoted as $R(\hat{c})$. Similarly, a multiple configuration set $\hat{C}'$ is must-reachable from configuration set $\hat{C}$ if every subset of $\hat{C}'$ is must-reachable from $\hat{c} \in \hat{C}$. They are denoted as $\delta R(\hat{c})$.

Although a C-graph can probably be unfolded to infinitely many C-trees, we can obtain such a C-tree that can describe all configurations, arcs and their order relationship. Such a C-tree is called complete.

**Definition 4.5.** Let $G = (C, E, s)$ be a C-graph and $CT = (C, E, \delta)$ be a C-tree of G. $CT$ is complete if
1. $\forall \hat{c} \in C$, $\exists \hat{c} \in C : \delta(\hat{c}) = \hat{c}$;
2. $\forall e \in (c_1, c_2) \subseteq \hat{E}, \exists \hat{e} \in \hat{E} : \delta(\hat{e}) = e$;
3. $\forall c_1, c_2 \in \hat{C} : c_2 \in R(c_1) \Rightarrow \exists \hat{c}_1, \hat{c}_2 \in \hat{C} : \delta(\hat{c}_1) = c_1 \land \delta(\hat{c}_2) = c_2 \land \delta(\hat{e}) = e$;
4. $\forall \hat{e}_1, \hat{e}_2 \in \hat{E} : \hat{e}_1 \leq \hat{e}_2 \Rightarrow \exists \hat{e}_1, \hat{e}_2 \in \hat{E} : \hat{e}_1 = \delta(\hat{e}_2) = \hat{e}_2 \land \hat{e}_2 \leq \hat{e}_1$;
5. $\forall \hat{c} \in C, \forall path_{nc}(\hat{c}_0, \hat{c}) \in \hat{PATH}(\hat{c}_0, \hat{c}) \subseteq C, \exists \hat{c}_1, \hat{c}_2 \in \hat{C} : \hat{c} \in \hat{c}_1 \land \delta(\hat{c}_1) = c_1 \land \delta(\hat{c}_2) = c_2 \land \delta(\hat{e}) = e$ and

Based on the above definition, we know that if a C-tree is complete, then a bigger one containing it is still complete, but its subtrees may not be complete. Thus we need to define the minimal complete C-tree (MCC-tree).

**Definition 4.6.** Let $G = (C, E, s)$ be a C-graph, and $CT = (C, E, \delta)$ and $CT' = (C', E', \delta)$ be two complete C-trees generated from G. CT is the subtree of CT' if $C' \subseteq C \land E \subseteq E'$. This is denoted as $CT \subseteq CT'$.

**Proposition 4.2.** Let $G = (C, E, s)$ be a C-graph and $S_{CCT}$ be the set of all complete C-trees generated from G. Then $(S_{CCT}, \subseteq)$ is a partially ordered set.

Proof: A partially ordered set is a binary relation satisfying reflexivity, antisymmetry, and transitivity. According to Definition 4.6, it is easy to know that $(S_{CCT}, \subseteq)$ is a partially ordered set.

**Definition 4.7.** Let $G = (C, E, s)$ be a C-graph and $S_{CCT}$ be the set of all complete C-trees generated from G. $CT \in S_{CCT}$ is the minimal complete C-tree (MCC-tree) if $\forall CT' \subseteq S_{CCT} : CT \subseteq CT'$.

**Theorem 4.2.** Let $G = (C, E, s)$ be a C-graph and $CT = (C, E, \delta)$ be the complete C-tree of G. $CT$ is a MCC-tree of G iff $\forall \hat{c} \in C : \delta(\hat{PATH}(\hat{c}_0, \hat{c})) = Cr(\delta(\hat{PATH}(\hat{c}_0, \hat{c})))$.

Proof: (Sufficiency). Assume that $CT$ is not a MCC-tree. Let $\delta(\hat{PATH}(\hat{c}_0, \hat{c})) = \hat{e}_1 \cdots \hat{e}_1 \cdots \hat{e}_n$ be an arbitrary path where $n \in \mathbb{N}^+$. $\delta(\hat{PATH}(\hat{c}_0, \hat{c})) = \hat{e}_1$ and $\delta(\hat{PATH}(\hat{c}_0, \hat{c})) = \hat{e}_n$. Then according to Definitions 4.4 and 4.6, there exists a $CT' = (C', E', \delta)$ such that $C' \subseteq C$ and $E' \subseteq E$. This indicates that there exists a path in $CT'$ such that $\delta(\hat{PATH}(\hat{c}_0, \hat{c})) = \hat{e}_1 \cdots \hat{e}_1 \cdots \hat{e}_n \cdots \hat{e}_n$. Therefore, $CT'$ is not complete. Therefore, the sufficiency holds.

(Necessity). Assume that there exists a configuration $\hat{c}$ and a path $\delta(\hat{PATH}(\hat{c}_0, \hat{c})) = \hat{e}_1 \cdots \hat{e}_1 \cdots \hat{e}_n$ in a complete C-tree $CT = (C, E, \delta)$ such that $\delta(\hat{PATH}(\hat{c}_0, \hat{c}))$ is a re-circuit path. Then according to Definition 4.4, there must exist a complete C-tree $CT' = (C', E', \delta)$ such that $\forall \hat{c} \in C, \forall \hat{e} \in C' : \delta(\hat{e}) = \hat{e} \Rightarrow \hat{c}$ and $Cr(\delta(\hat{PATH}(\hat{c}_0, \hat{c})))$ is not complete. Therefore, we can get $CT' \subseteq C \land E' \subseteq E$. According to Definition 4.7, $CT'$ is not the minimal complete C-tree. Therefore, the necessity also holds.

Fig. 2(b) is a C-tree of the C-graph in Fig. 2(a). Obviously, it is not complete since it lacks several paths which can occur in Fig. 2(a). For example, the path $e_1 e_3 e_7 e_2 e_4$ represents an executed case in which data $a$ is written twice until a terminal configuration is reached. At the first time it is written with a value which does not satisfy its data constraint but returns to rewrite. At the second time it is written with a satisfied value and finally the terminal configuration is reached. Obviously, this path is not in the C-tree in Fig. 2(b).

Fig. 2(c) is an MCC-tree. We can observe all running status from it. In this MCC-tree, there does not exist the situation that the same arc sequence appears twice continuously, i.e., redundancy is avoided.

Fig. 2(d) is a C-tree. It is complete but not minimal. Although it can reflect all running status of the related C-graph, there is redundancy in it. For example, sequence $e_2 e_6 e_3 e_7 e_2 e_4$ is a re-circuit path since $e_2 e_4$ continuously repeats twice, i.e., $Cr(e_2 e_6 e_3 e_7 e_2 e_4) = e_2 e_6 e_3 e_7$. Algorithm 1 is a function to backtrack and locate a pair of configuration and arc which satisfies some special conditions. The form of function Backtrack() is recursive, then time complexity of Algorithm 1 is $O(W)$, where $W$ is the number of configurations in the current MCC-tree. By recalling the backtrack function, Algorithm 2 shows the method of constructing a MCC-tree. The algorithm is based on the depth-first search and Theorem 4.2 guarantees its correctness. Its time complexity is $O(C \times W)$, where $C$ is the number of configurations in the C-graph.

**B. LOCATING UNSOUND COMPONENTS**

To locate the unsound transitions via an MCC-tree, we need to extract the normal and abnormal executing paths from it.
and focus on the abnormal nodes. In this paper, an executing path (e-path) starts from initial configuration and ends at a dead configuration or a terminal configuration. For example in Fig. 2(c), e1e2e4 and e1e2e6e2e4 are two e-paths in the C-tree.

To effectively distinguish different e-paths and reduce unnecessary calculation, we classify all e-paths to five types based on two aspects: whether it is must-terminated; and whether it ends with abnormal data. The four kinds of e-paths are defined as follows.

1) PN-paths (Proper and Normal Paths). The final configuration of a PN-path is a terminal configuration (must-terminated) and the values of data constraints are all TRUE.

2) PA-paths (Proper and Abnormal Paths). The final configuration of a PA-path is a terminal configuration (must-terminated) but there exists at least one data constraint whose value is FALSE.

*reason: There is no guard to prevent the transferring of abnormal data.

3) IN-paths (Improper and Normal Paths). The final configuration of an IN-path is a dead configuration (may-terminated) but the values of data constraints in it are all TRUE.

*reason: The value of one predicate in the final configuration does not satisfy a guard, but there is no disposition with it; or other control-flow errors.

4) IA-paths (Improper and Abnormal Path). The final configuration  of an IA-path is a dead configuration (may-terminated), there exists at least one data constraint \( \varphi \) whose value is FALSE.

*reason: The value of one data constraint in the final configuration does not satisfy a guard, but there is no disposition with it; or other control-flow errors.

PN-paths are correct e-paths which represent correct executions of the system. PA-paths, IN-paths, IA-paths are incorrect e-paths that represent incorrect executions of the system. In Fig. 2(c), there are totally 12 e-paths. For example, e1e2e3e5 is a PA-path because there is a constraint \( \psi e \) whose value is FALSE at the terminal configuration e5, and e1e2e4 is a PN-path because all data constraints’ values are TRUE at the terminal configuration e4.

Based on the four types of e-paths, we can calculate the unsound transitions. Thus, we focus on the transitions where deadlocks happen or the transitions that generate or transfer abnormal data.

By calculating these e-paths in a C-tree, a set of transitions transferring.

**Algorithm 1** Function BackTrack

**Input:** C-graph G; MCC-tree mc(CT); e, \( \bar{e} \);

**Output:** \( \bar{e}, \bar{c} \);

1: function BackTrack(mc(CT), \( \bar{e}, \bar{c} \))
2: if \(|e|^* = |\delta(e)^*| \) then
3: if \( *e \neq @ \) then
4: Mark \( \bar{e}, \bar{c} \) with "old";
5: Point \( \text{current} \) to \*e and \( *\bar{e} \);
6: \( Sq = \text{Sq.Substring}(0, \text{Sq.Length} - 1) \);
7: BackTrack(mc(CT), \( \bar{e}, \bar{c} \));
8: end if
9: end if
10: return \( \bar{e}, \bar{c} \);

**Algorithm 2** Constructing a MCC-tree

**Input:** C-graph G;

**Output:** MCC-tree mc(CT);

1: initialization: del(CT) = \( \emptyset \); String Sq = "";
2: Choose \( c_0 \) as a root configuration \( \bar{c}_0 \);
3: mc(CT) = \{\( (\emptyset, \bar{c}_0) \)\}; Point \( \bar{c}_0 \in C \) as a \( \text{current} \) \( \bar{e} \);
4: Point \( \bar{c}_0 = \emptyset \) as a \( \text{current} \) \( \bar{e} \);
5: if \( \exists \delta(e)^* \in C \land |e|^* < |\delta(e)^*| \) then
6: Find a configuration \( c_i \in \delta(e)^* : \delta(\bar{c}_i) \in mc(CT) \); \( \delta(\bar{c}_i) = c_i \land (\bar{c}_i, \bar{c}_i) \) marked with "old";
7: Record the \( \bar{c}_i \) as \( \bar{c}_{new} \); Generate a new \( \bar{c}_{new} = \bar{c}_i, \bar{c}_{new} \) such that \( \delta(\bar{c}_{new}) = \{\bar{c}_{new}, \bar{c}_{new}\} \);
8: \( mc(CT) = mc(CT) \cup \{\bar{c}_{new}, \bar{c}_{new}\} \);
9: \( Sq = \text{Sq}+"\delta(\bar{c}_{new})" ;\)
10: Mark \( \bar{e}, \bar{c} \) with "old"; Point \( \text{current} \) to \( \bar{c}_{new} \) and \( \bar{e}_{new} \);
11: if suffix of \( Sq \) is two continuous and repetitive strings then
12: let \( \bar{e}_{del} = \bar{c}_i; \bar{e}_{del} = \bar{c}_i; \)
13: Point \( \text{current} \) to \*e and \( *\bar{e} \);
14: \( Sq = \text{Sq.Substring}(0, \text{Sq.Length} - 1) \);
15: BackTrack(mc(CT), \( \bar{e}, \bar{c} \));
16: while \( \bar{e}_{del} \neq \bar{c} \land \bar{e}_{del} \) is not a terminal configuration do
17: \( del(CT) = del(CT) \cup \{(e_{del}, \bar{c}_{del})\} \);
18: \( \bar{e}_{del} = \bar{c}_{del}; \bar{e}_{del} = \bar{c}_{del} \);
19: end while
20: end if
21: else if \( \exists \delta(e)^* \in C \land |e|^* = |\delta(e)^*| \) then
22: if \( \bar{e} \) is a root configuration then
23: break;
24: else
25: BackTrack(mc(CT), \( \bar{e}, \bar{c} \));
26: end if
27: else
28: BackTrack(mc(CT), \( \bar{e}, \bar{c} \));
29: end if
30: end loop
31: \( mc(CT) = mc(CT) - del(CT) \);
Algorithm 3 Locating Unsound Component

Input: MCC-tree CT; Data constraint set \( \Phi = \{ \Phi_{\text{part},i} | i \leq n, n \in N^+ \} \);

Output: Unsound transition UT;

1: if there exist IN-paths then
2:  Mark the ending configuration \( \bar{c}_d \) of every IN-path with "uncount";
3: end if
4: while The "uncount" configuration \( \bar{c}_d \) exists do
5:  UT = UT \cup \{ \bar{c}(\text{input}(\bar{c}_d)) \};
6:  Remove the "uncount" mark;
7: end while
8: for \( \varphi_{\text{part},i} \in \Phi; \Phi \neq \emptyset; \Phi = \{ \varphi_{\text{part},i} \} \) do
9:  Initialize transition \( \bar{t} \); a string \( \lambda = \"\" \); transition sequence set \( \text{PreU}_{\varphi_{\text{part}}} = \emptyset; \) transition set \( \text{U}_{\varphi_{\text{part}}} = \emptyset; \)
10: Mark all e-paths as "new";
11: while "new" e-path exists do
12:  Arbitrarily choose one "new" path;
13:  for \( \bar{c} = \bar{c}_0, \bar{c} \neq \emptyset; \bar{c} = \bar{c}^* \) do
14:    Check constraint \( \varphi_{\text{part},i} \), in configuration \( \bar{c} \);
15:    record \( t \) that satisfies \( t = s(\delta(\bar{c})) \land \bar{c} = \text{input}(\bar{c}) \land \lambda = \lambda + t; \)
16:    end if
17:  PreU_{\varphi_{\text{part}}} = \text{PreU}_{\varphi_{\text{part}}} \cup \{ \lambda \}; \lambda = \"\" ;
18:  end for
19: Mark the path as "old";
20: while there exists two sequences \( \lambda_1, \lambda_2 \) in \( \text{PreU}_{\varphi_{\text{part}}} \) satisfying that \( \lambda_1 > \lambda_2 \) do
21: Delete \( \lambda_2 \);
22: end while
23: if \( n \neq 1 \) then
24:  For \( t \in \text{T}_{\text{part},i} \land t \) is contained in \( \lambda \} \) where \( \lambda \in \text{PreU}_{\varphi_{\text{part}}} \);
25:  if \( \text{Rd}(t) = \emptyset(\varphi_{\text{part},i}) = \emptyset \) then
26:    \( \text{U}_{\varphi_{\text{part}}} = \text{U}_{\varphi_{\text{part}}} \cup \{ t \}; \)
27:  end if
28: else
29:  \( \text{U}_{\varphi_{\text{part}}} = \{ t | t \) is the first transition of \( \lambda \} \) where \( \lambda \in \text{PreU}_{\varphi_{\text{part}}} \);
30: end if
31: UT = UT \cup \{ \text{U}_{\varphi_{\text{part}}} \};
32: end if
33: end for

The "uncount" configuration \( \bar{c}_d \) exists do
34: UT = UT \cup \{ \text{U}_{\varphi_{\text{part}}} \};
35: end for

V. CONTROLLING WFDC-NETS

The C-graph is used to verify the soundness of a WFDC-net, and the MCC-tree is used to look for unsound transitions. Next we still use MCC-tree as an intermediate to control the WFDC-net. In this section we introduce a controlling method of WFDC-net based on unsound transitions. The controlling purpose is to guarantee that there are no PA-paths, IN-paths or IA-paths in the controlled WFDC-net.

A. PSEUDO INTERMEDIATE SET

An MCC-tree presents complete dynamic behavior of a WFDC-net. Therefore, we can generate an intermediate as a modification reference based on four kinds of e-paths and thus change the flow trend of configurations and avoid the

Fig. 1. \( T = \text{T}_{\text{part},1} = \{ t_1, t_2, t_3, t_4 \}, \Phi = \Phi_{\text{part},1} = \{ \psi \} \) since it is a single system.

Based on the different roles of places and transitions, we can give the formal definition of unsound transitions.

Definition 4.8 (Unsound Transitions). Let \( N = (WD, Pa(\Phi)) \) be a WFDC-net and \( G = (C, E, s) \) be a C-graph of \( N \), \( CT = (C, E, \delta) \) is a MCC-tree of \( G \). The set of unsound transitions of \( N \) is a transition set in which every transition satisfies the following conditions:

1) For a transition \( t \in \text{T}_{\text{part}}, \) if a dead configuration is generated after firing \( t, \) in which \( \forall \varphi \in \Phi : \rho_\Phi(\varphi) = \text{TRUE}, \) then \( t \in UT; \)
2) For a transition \( t \in \text{T}_{\text{part}}, \) if it satisfies
   a) \( \exists \bar{c}, \bar{c}' \in C, \exists \varphi \in \Phi : \delta(\bar{c}) \land \delta(\bar{c}') \land \rho_\Phi(\varphi) = \text{FALSE}; \) and
   b) \( \forall \varphi \in \Phi_{\text{part},i}, \exists d \in \text{Rd}(t) \cap \Delta_\Phi(\varphi), \exists e \in E : s(\delta(\bar{c})) = t \Rightarrow \exists t' \in \text{T}_{\text{part}}, \exists d \in Wt(t'), \exists \Gamma_D(d) \in \Phi_{\text{part}}, : s(\delta(\bar{c}')) = t' \land \bar{c}' \leq \bar{c}, \)
3) If there exist \( t_1, t_2 \in \text{T}_{\text{part}}, \) satisfying condition 2) and 3), and \( \exists e_1, e_2 \in E : t_1 = s(e_1) \land t_2 = s(e_2) \land e_1 \leq e_2, \) then \( UT = UT - \{ t' \}. \)

In view of control-flows, the unsound transitions can generate dead configurations. In view of data-flows, the unsound transitions have three characteristics: (1) they appear behind the writing operation; (2) They cannot stop the transferring of data constraints whose value is FALSE; and (3) they approach the transitions with writing operation as closely as possible. According to the definition of unsound transitions, we can extract them from the MCC-tree. The detailed procedure of extracting unsound transitions is shown in Algorithm 3. Let \( N_D \) be the number of dead configurations in the MCC-tree, \( N_e \) be the number of data constraints and \( N_C \) be the number of configurations in the MCC-tree, then the time complexity of Algorithm 3 is \( O(N_D + N_e + N_C^2) + N_C \).

For example in Fig. 1, we can get the abnormal transition sequence of \( \psi \) is \( t_2 t_3 \). According to Algorithm 3, \( U_{\psi_{\text{part}}} = \{ t_2 \}. U_{\psi_{\text{part}}} \) means the unsound transitions of \( \psi_{\text{part}} \). Then \( UT = U_{\psi_{\text{part}}} = \{ t_2 \}. \)
abnormal configurations. As a result, we can control the WFDC-net based on these changes.

The modification reference is called pseudo intermediate set (PIS). It is a virtual intermediate for controlling WFDC-net. The PIS can offer a direct indication for changing the MCC-tree, controlling WFDC-net and ensure the controlled WFDC-net sound.

Due to the complexity of systems, the appearance of abnormal data is inevitable to some extent. Although an abnormal data appears sometimes, it can be stopped and deleted after several steps. This situation is sound but the reaction of the system still can be improved. Note that in IN-paths there may be a situation such that in the dead configuration \( \bar{c} \), marking \( M \) satisfies the enabled condition, all data constraints’ values are TRUE, but there exists a predicate \( \pi \in \Pi \) whose value is FALSE and when its value is changed to TRUE, the configuration becomes live. It is because that for the transition \( t \) at \( \bar{c} \), there is a predicate \( \pi \) satisfying \( \ell_{\varphi}(\pi) \cap \ell_{\varphi}(\varphi) = \emptyset \) and \( \rho_{\pi}(\pi) = \text{FALSE} \). As a result, the value of the guard is FALSE. Therefore, we should consider the effect of predicates’ value into the generating rule of PIC-graph and PIS.

In a single-participant system, if there is a problem with data, we only need to simply rewrite the data. In a multi-participant system, the interaction is tight and data have strong independency. Once a problem occurs, the whole procedure or the whole system must restart from beginning. Meanwhile, controls should be consistent with data constraints which are declared by different participants. For example in an e-commerce transaction system, the merchant needs to check the order and the amount of the shopper’s payment but does not need to consider the quality of goods. On the contrary, the shopper does not need to re-check the payment but does not need to consider the quality of goods. Therefore, we need to consider the beneficiaries of the data constraints in the generating rule. Based on e-paths, we get the PIC-graph and PIS according to the following rules:

**Definition 5.1.** Let \( CT = (C, E, \delta) \) be a MCC-tree and \( PIS = \emptyset \) be the initial PIS.

1) For an IN-path:
   - Record dead configuration \( \bar{c} \) and initial configuration \( \bar{c}_0 \) in \( CT \); \( PIS = PIS \cup \{ (\bar{c}, \bar{c}, \bar{c}_0, \text{null}, \text{null}) \} \).

2) For a PA-path or an IA-path:
   - Let \( \Phi = \{ \varphi_{\text{part}_1}, \ldots, \varphi_{\text{part}_n} \} \) where \( i \in \mathbb{N}^+ \) and \( n \in \mathbb{N}^+ \). For every constraint \( \varphi_{\text{part}_i} \in \Phi \), its value is FALSE in the final configuration:
     a) \( (n = 1) \) For every \( t \in U_{\varphi_{\text{part}_i}} \), we record every configuration \( \bar{c} \) generated by firing \( t \) and \( \bullet \bar{c} \) in \( CT \);
     - If \( \bar{c} \) is a terminal configuration, then \( PIS = PIS \cup \{ (\bar{c}, \bar{c}, \text{null}, \text{null}) \} \); otherwise, \( PIS = PIS \cup \{ (\bar{c}, \bar{c}, \text{null}, \text{null}) \} \).
     b) \( (n > 1) \) For every \( t \in U_{\varphi_{\text{part}_i}} \), we record every configuration \( \bar{c} \) that satisfies \( \delta(\bar{c})[t] \) and record initial configuration \( \bar{c}_0 \) in \( CT \);

   If \( \bar{c} \) is a terminal configuration, then \( PIS = PIS \cup \{ (\bar{c}, \bar{c}, \text{null}, \text{null}) \} \); otherwise, \( PIS = PIS \cup \{ (\bar{c}, \bar{c}, \text{null}, \text{null}) \} \).

**FIGURE 3.** An illustration of how PIS is generated from the C-tree. Fig. 2(c).

According to Definition 5.10, if there is no data constraint whose value is FALSE at the terminal configuration, then we regard this termination as normal since it is sound from the view of data. Therefore, PN-paths remain unchanged, and we only need to consider those paths which can not reach the terminal configurations (i.e., IN-paths and IA-paths) or have an abnormal data at terminal configurations (i.e., PA-paths). After the application of generating rule, PIS is obtained.

For example, the unsound transitions of the WFDC-net in Fig. 4 is UT \( \{ t_2 \} \). Use the generating rule of PIS, the PIS is obtained as PIS = \{ \( (c_3, e'_2, c_1, \psi, \text{null}) \), \( (c_3, e'_2, c_1, \psi, \text{null}) \), \( (c_3, e'_2, c_1, \psi, \text{null}) \), \( (c_3, e'_2, c_1, \psi, \text{null}) \) \} by using the generating rule. Fig. 3 illustrates how the PIS is generated from an MCC-tree.

**FIGURE 4.** The controlled example in Fig. 4(a) and its new C-graph(b).

**B. CONTROLLING STRATEGY**

According to the PIS, the original WFDC-net can be controlled and the control strategy is listed in TABLE 1: To
TABLE 1. Controlling strategy for WFDC-net.

<table>
<thead>
<tr>
<th>Step</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Let PIS = {(e, e', \psi)</td>
</tr>
<tr>
<td>2</td>
<td>Let ( \varpi ) be a compound proposition formula, ( \psi_{pis}, \psi_{pis}' \in PIS\rho ), if ( P_1 = P_1', P_2 = P_2', P_3 = P_3' ), then generate ( \psi_{pis}' = (P_1', e, e', P_2', M, \psi, \psi') ) .</td>
</tr>
<tr>
<td>3</td>
<td>Add new flow relation ((t_0', p_0') ) to ( F). ( \psi_{pis} = p_0' ) and ( \psi_{pis}' = t_0' ) where ( t_0' ) is a new transition and ( p_0' ) is a new place.</td>
</tr>
</tbody>
</table>

Elements in PIS\rho  

<table>
<thead>
<tr>
<th>Modifying WFDC-net based on elements on PIS\rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (P, e, P', \varpi, P)(P' = {i}) )</td>
</tr>
<tr>
<td>Add a new transition ( t' = t' ) and ( t'<em>{\psi</em>{pis}} = \psi_{pis}' );</td>
</tr>
<tr>
<td>Add ( t' ) with a delete function according to changes of data from ( \bar{e} ) to ( e ); and</td>
</tr>
<tr>
<td>Let ( grd(t) := \varpi ) and ( grd(t') := grd(t') \wedge \varpi ).</td>
</tr>
<tr>
<td>( (P, e, P', \varpi, \varnothing)(P' = {i}) )</td>
</tr>
<tr>
<td>Add ( t' ) with a delete function according to changes of data from ( \bar{e} ) to ( e ); and</td>
</tr>
<tr>
<td>Let ( grd(t) := \varpi ) and ( grd(t') := grd(t') \wedge \varpi ).</td>
</tr>
<tr>
<td>( (P, e, P', \varpi, \varnothing)(P' = {i}) )</td>
</tr>
<tr>
<td>Add ( t' ) with a delete function according to changes of data from ( \bar{e} ) to ( e ); and</td>
</tr>
<tr>
<td>Let ( grd(t) := \varpi ) and ( grd(t') := grd(t') \wedge \varpi ).</td>
</tr>
<tr>
<td>( (P, e, P', \varpi, \varnothing)(P' = {i}) )</td>
</tr>
<tr>
<td>Add ( t' ) with a delete function according to changes of data from ( \bar{e} ) to ( e ); and</td>
</tr>
<tr>
<td>Let ( grd(t) := \varpi ) and ( grd(t') := grd(t') \wedge \varpi ).</td>
</tr>
</tbody>
</table>

VOLUME 4, 2016

![Figure 5](image-url)

**FIGURE 5.** An illustration of step 3 in Table 1. **start** is the source place and **end** is the sink place.

match the formal definition of WFDC-net, we must change the configurations in PIS to places. Step 1 shows the transforming process of PIS. Arcs with different data constraints but the same pre-set and post-set may result in redundancy. Therefore, a step of merging these transitions is also necessary, which is illustrated as step 2. step 3 guarantees the characteristics of WFDC-net before controlling. The three kinds of tuples in step 4 are obtained through step 2 and the corresponding modification are illustrated in Fig. 5.

For example in Fig. 5, since PIS is obtained, we can easily get the PIS after executing steps 1 – 2 in Table 1 PIS\rho = \{(p_2, e_1, P_1, \psi_{\rho}, \varnothing)\}. Then, \( t_0' \), \( p_0' \), transition \( t_\psi \) and arcs are added, and a guard \( \neg \psi \) is added to \( t_\psi \). Meanwhile, \( \varnothing \) indicates that \( p_2 \) is not a terminal place, i.e., there is no need to extent a new transition before the terminal place. Therefore, we only need to label a \( \psi \) as a guard on \( t_3 \).

Based on these steps, the controlled WFDC-net is illustrated in Fig. 5(a), and its C-graph is shown in Fig. 5(b). By analyzing the C-graph we can know that the abnormal termination is avoided and the final system is sound.

VI. CASE STUDY

A workflow system of Paypal Express [1] includes three participants: shopper, merchant and TPP (the Third Payment
First, the shopper sends an order request \((\text{items})\) to merchant. Then the merchant generates the primary order \((\text{orderM})\) and total price \((\text{grossM})\) of the payment. The shopper pays the money \((\text{orderIDS}, \text{grossS})\) to TPP if the shopper agrees the total price. After receiving the payment, the TPP returns a transaction number \((\text{tranIDT})\) to the shopper. The merchant contacts the TPP with \(\text{tranIDT}\) that is received by the merchant from the shopper to verify \(\text{orderIDS}\) and \(\text{grossS}\) in its database. If the payment is correct, the merchant updates the order payment status and sends a notification to the shopper. At last the whole checkout procedure is done.

Note that some of unnecessary messages in [1] are ignored.

In this transaction process, the shopper probably becomes an attacker when there exist some vulnerabilities. To avoid confusion, we use different symbols of data to represent a data that may be falsified by different participants in different tasks. For example, \(\text{orderIDM}\) denotes the primary order that is generated by merchant, \(\text{orderIDS}\) denotes the orderID that the shopper/Attacker may falsify during payment. The business process of merchant does not check the \(\text{gross}\) when the TPP transfers the paid \(\text{tranIDT}\) and the merchant will mistakenly believe that \(\text{orderIDS}\) is paid.

Fig. [6] shows the WFDC-net of the system. We regard the data which is rewritten by different parties as different data. Transition set \(T = T_S \cup T_M \cup T_T \cup \{t_0, t_1, t_{11}, t_{12}\}\), where \(T_S = \{t_2, t_3, t_4, t_5\}\), \(T_M = \{t_6, t_7, t_8\}\) and \(T_T = \{t_9, t_{10}, t_{13}\}\).
For a better description, we use $\text{Checkout}_M(\text{orderIDS})_\varphi$ to present a data constraint and $\text{Checkout}_M(\text{orderIDS})$ to present a predicate in guards. Guards in blanket on transitions represents the control mechanism for validation. The solid line indicates the control-flows and the dotted line indicates the data-flows.

Fig. 7 shows its C-graph. From the C-graph we can see that there are several dead configurations (e.g., $c_{12}$ and $c_{16}$). Meanwhile, there exists a terminal configuration $t_{23}$ in which one of the data constraints evaluates FALSE. According to Definitions 3.9, 3.10 and 3.11, this system is neither control-sound nor data-sound.

Fig. 8 is the MCC-tree of Fig. 7. There are 8 different $e$-paths in it. 4 $e$-paths have dead configurations and 4 $e$-paths reach the terminal configurations. 2 $e$-paths have data constraints whose values are FALSE in the terminal configurations. Therefore, there are 2 PN-paths (e.g., $e$-paths 4, 8), 2 PA-paths (e.g., $e$-paths 3, 7) and 4 IA-paths (e.g., $e$-paths 1, 2, 5, 6).

Based on Algorithm 5, the transition leading to a deadlock is $t_{10}$. Next, we can get

$\text{PreU}_M(\text{orderIDS})_\varphi = \{t_3t_9t_4t_7t_{10}\}$

$\text{PreU}_M(\text{grossS})_\varphi = \{t_3t_9t_4t_7t_{10}t_5t_{11}\}$

Since $T_M = \{t_6, t_7, t_8\}$ and $Rd(t_8) = \{\text{orderIDS, grossS}\}$, we get

$U_M(\text{orderIDS})_\varphi = \{t_8\}$

$U_M(\text{grossS})_\varphi = \{t_8\}$
Therefore, the unsound transitions UT = \{t_8 \} \cup \{t_{10} \} = \{t_8, t_{10}\}. Note that t_8 in UT has a close relation with two corresponding data constraints. Therefore, t_8 should be controlled with a guard in which there is a conjunction of two predicates Checkout_M(orderIDS) and isGrossM(grossS).

After obtaining the unsound transitions of abnormal data elements, we can generate the PIS. For example, (c_{12}, e'_1, c_0, null, null) is added according to IA-path 1. Since the value of data constraints Checkout_M(orderIDS)_\varphi and isGrossM(grossS)_\varphi are FALSE in IA-path 1, (c_{12}, e'_2, c_0, Checkout_M(orderIDS)_\varphi, null) and (c_{12}, e'_3, c_0, Checkout_M(orderIDS)_\varphi, null) are also added in PIS.

Meanwhile, in terminal configuration of PA-path 3, the value of isGrossM(grossS)_\varphi is FALSE, then (c_{12}, e'_4, c_0, isGrossM(grossS), null) is added in PIS. Since the PIS of the system is large, elements in it are not listed one by one here.

Next, we transform PIS to PIS_\phi and integrate the redundancy (see step 1 and step 2 of Table 1). For example, there are (c_{12}, e'_1, c_0, null, null), (c_{12}, e'_2, c_0, Checkout_M(orderIDS)_\varphi, null) and (c_{12}, e'_3, c_0, Checkout_M(orderIDS)_\varphi, null) in PIS, and then we transform them to (\{p_4, p_7, endT, cp_7\}, e'_1, \{start\}, null, null), (\{p_4, p_7, endT, cp_7\}, e'_2, \{start\}, null, null).
Checkout/M(orderIDS) ∈ null and (\{p_4, p_7, endT, cp_7\}, e_5, \{start\}). Checkout/M(orderIDS) ∈ null. Then, they are integrated to (\{p_4, p_7, endT, cp_7\}, e_5, \{start\}, Checkout/M(orderIDS) ∨ isGrossS(grossS) ∈ null).

After step 1 and step 2, \(P_{IS} = (\{p_4, p_7, endT, cp_7\}, e_5, \{start\}, Checkout/M(orderIDS) ∨ isGrossS(grossS) ∈ null)\). Then, we modify WFDC-net according to three types (see step 2 and step 3 of Table 1). Therefore, a controlled WFDC-net is obtained as shown in Fig. 9. \(t_6\) that is mapped with functions are added to control the potential abnormal data grossS, and orderIDS. Predicate isGrossS(grossS) is added in to \(qrd(t_s)\).

Fig. 10 shows the C-graph of this new WFDC-net which indicates that the controlled WFDC-net satisfies both control-soundness and data-soundness.

VII. CONCLUSION

If we locate the unsound transitions where potential data problems may occur, we can effectively redesign systems at the design stage to avoid possible security issues. Meanwhile, if we can use a general method to automatically control the unsound transitions, then we can reduce the cost in the redesign process.

In this paper, we proposed an algorithm to locate the unsound transitions of a workflow system and a controlling strategy to avoid these risks.

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

1) To reduce the state expansion problem caused by concurrency in complicated systems, we plan to study the related methods based on unfolding techniques [36, 37].

2) Develop a tool based on our algorithms in order to automatically locate the unsound transitions and control them.

REFERENCES


CHUNGLANG YAN received the Ph.D. degree from Tongji University, Shanghai, China, in 2006. She is currently a Professor with the Department of Computer Science and Technology, Tongji University. Her current research interests include Petri net theory, model checking, Web service, workflow, discrete event systems, and information security.

YAOQIONG HE graduated from Tongji University, China, in 2012. She is currently working toward the Ph.D. degree in the Department of Computer Science and Technology, Tongji University. Her research interests include model checking, Petri net, business process management, and workflow systems.
CHANGJUN JIANG received the Ph.D. degree from the Institute of Automation, Chinese Academy of Science, Beijing, China, in 1995. He is currently the leader of the Key Laboratory of the Ministry of Education for Embedded System and Service Computing, Tongji University, Shanghai, China. He is an IET Fellow and an Honorary Professor with Brunel University London. He has published more than 300 papers in journals and conference proceedings, including Chinese Science, *IEEE Transactions on Robotics and Automation*, and *IEEE Transactions on Fuzzy Systems*. He has led over 30 projects supported by the National Natural Science Foundation of China, the National High Technology Research and Development Program of China, and the National Basic Research Developing Program of China. His research interests include concurrency theory, Petri nets, formal verification of software, cluster, grid technology, intelligent transportation systems, and service-oriented computing. Prof. Jiang has been the recipient of one international prize and seven prizes in the field of science and technology.

JIACUN WANG received the Ph.D. degree in computer engineering from the Nanjing University of Science and Technology (NJUST), China, in 1991. He was an Associate Professor with NJUST. He was a Research Associate with the School of Computer Science, Florida International University, Miami. From 2001 to 2004, he was a Scientific Staff Member with Nortel Networks, Richardson, TX, USA. He is currently a Professor with the Computer Science and Software Engineering Department, Monmouth University, West Long Branch, NJ, USA. His research interests include software engineering, discrete event systems, formal methods, wireless networking, and real-time distributed systems.