Detecting Infeasible Traces in Process Models

Lijie Wen
Tsinghua University
102 PUBLICATIONS **1,362** CITATIONS

DETECTING INFEASIBLE TRACES IN PROCESS MODELS

Zhaoxia Wang[1,2,3,4,5], Lijie Wen[1,3,4], Xiaochen Zhu[1,2,3,4], Yingbo Liu[1,3,4] and Jianmin Wang[1,3,4]
[1] *School of Software, Tsinghua University, Beijing 100084, China*
[2] *Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China*
[3] *Key Lab for Information System Security, Ministry of Education, Beijing 100084, China*
[4] *National Laboratory for Information Science and Technology, Beijing 100084, China*
[5] *Logistical Engineering University, Chongqing 400016, China*
{*wang-cx06, wenlj00, zhu-xc10, lyb01*}*@mails.thu.edu.cn, jimwang@thu.edu.cn*

Keywords: Infeasible Traces; Process Models; Workflow Analysis.

Abstract: Workflow testing is an important method of workflow analysis in design time. A challenging problem with trace-oriented test data generation in particular and trace-based workflow analysis in general is the existence of infeasible traces for which there is no input data for them to be executed. In this paper we build on the theory of workflow nets and introduce workflow nets where transitions have conditions associated with them. We then demonstrate that we can determine which execution traces, that are possible according to the control-flow dependencies, are actually possible taking the data perspective into account. This way we are able to more accurately determine in design time the infeasible traces caused by the correlation between transition conditions along this trace. Finally, we provide a solution to automatically detecting the shortest infeasible trace.

1 INTRODUCTION

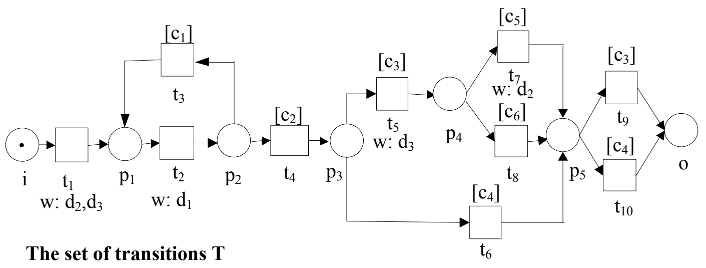
Workflow testing is an important method of workflow analysis in design time (van der Aalst and ter Hofstede, 2000). However, test data generation is an extremely complicated and time-consuming task as it requires a careful analysis and understanding of the source code and knowledge of the underlying concepts of the testing criterion. For this reason, automation of test data generation may provide a significant cost and time reduction for workflow testing. In trace-oriented test data generation, a set of traces is selected which satisfies one or more coverage criteria and test data are generated to exercise these traces. A challenging problem with trace-oriented test data generation in particular and trace-based workflow analysis in general is the existence of infeasible traces for which there is no input data for them to be executed. In this case, considerable effort might be wasted in trying to generate data for infeasible traces.

In software engineering domain, detection of branch condition and branch correlation is a kind of important methods in path feasibility analysis(Ngo and Tan, 2008). Here, inspired by the approaches of detecting the infeasible paths in software engineering domain we propose an approach to detect the infeasible traces in process model.

For instance, for the simplified example model for loan process (shown in Figure 1), the traces $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ and $t_1t_2(t_3t_2)*t_4t_6t_9$ become infeasible when transition conditions are taken into account. The reason of traces $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ non-existing is that the conjunctive transition conditions $c_3 \wedge c_4$ on transition t_{10} along $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ are not satisfied, they are then non-executable (that is, they are **infeasible traces**).

there are special features in workflow domain from in ordinary program domain. It is common that there are parallel structures in workflow models from control flow perspective while parallel structures don't be considered in program in general. In addition, most of workflow data is from external environment and isn't controlled by workflow, workflow model has limited data information. Therefore, new challenges should be solved to detect infeasible traces in workflow domain: how to formally describe the workflow model taking into data account? how to formally describe the behavioural semantic of workflow model with data? how to solve the state space explosion caused by the data dimension?

In this paper, we build on the theory of workflow nets and introduce workflow nets where transi-



The set of transitions T

t_1: receive_application	t_2: verify_application	t_3: modify_application
t_4: check_loan amount	t_5: check risk	t_7: modify-loan
t_9: approve by manager	t_{10}: approve by officer	t_6, t_8: silent task

The set of condition variables

d_1: info_complete d_2: loan amount d_3: risk

The set of conditions

c_1: $\neg d_1$ c_2: d_1 c_3: d_2>50,000 c_4: d_2<=50,000 c_5: d_3 c_6: $\neg d_3$

Figure 1: A workflow model for the simplified loan process.

tions have conditions associated with them. We then demonstrate that in principle it is possible to construct a reachability graph that takes conditions into account. Specifically, a historic constraint along a sequence is recorded dynamically, and this constraint as well as the transition condition on the current transition act together to determine whether the current transition is enabled or not. Finally we are able to more accurately determine in design time the infeasible traces caused by the correlation between transition conditions.

The rest of this paper is organized as follows. First a formal definition of *workflow nets with transition conditions* or *WTC-nets* for short is given (Section 2). Subsequently, an approach based on reachability analysis technique for automatically determining shortest infeasible traces, is described (Section 3). A tool that implements the proposed approach is presented (Section 4). This tool is used to apply the approach to a number of real-life models in order to provide some insight into its potential practical applicability (Section 5). Section 6 discusses the related work and Section 7 concludes the paper.

2 Preliminaries

In this section we briefly introduce workflow nets with transition conditions (for short, WTC-nets). WTC-nets are based on Petri nets and workflow nets, in order to make the paper self-contained some well-known definitions are introduced first.

2.1 Background

In this paper we build on workflow nets which are a subclass of Petri nets. In (van der Aalst, 1998b), van der Aalst argued for the suitability of these nets for workflow management. Below, we define Petri nets first. For an overview of Petri nets and an extensive bibliography the reader is referred to (Murata, 1989).

Definition 1 (Petri net)**.** *A Petri net N is a tuple (P,T,F) where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \varnothing$, $P \cup T \neq \varnothing$, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.*

For each node x, i.e. a place or a transition, we use $\bullet x$ to denote its pre-set - the set of nodes which are input of x - i.e. $\bullet x = \{y|(y,x) \in F\}$, and $x\bullet$ for its post-set - the set of nodes which are output of x - i.e. $x\bullet = \{y|(x,y) \in F\}$. Markings represent states of a net and formally correspond to an assignment of tokens to places.

Definition 2 (Marking)**.** *Let $N = (P,T,F)$ be a Petri net, a* marking *M of N is a function from its places to the set of natural numbers, i.e. $M : P \to \mathbb{N}$.*

A marking is multiset over P and can be written as a linear combination of places, e.g. $M = 7p_1 + 0p_2 + 2p_3$. It is convenient to only list places that contain tokens so marking M is written as $7p_1 + 2p_3$. For a Petri net $N = (P,T,F)$ we can compare markings. A marking M is greater than a marking M', $M > M'$, iff for all $p \in P : M(p) \geq M'(p)$ and there is at least a $q \in P$ such that $M(q) > M'(q)$. Transitions can change the marking of a Petri net if they are fired.

Definition 3 (Enabled transition)**.** *Let M be a marking of Petri net $N = (P,T,F)$, transition t is* enabled *in M, denoted $M[t>$, iff for every place $p \in \bullet t : M(p) >$ 0.*

Definition 4 (Firing a transition)**.** *Let M be a marking of Petri net $N = (P,T,F)$, and t a transition that is enabled in M, i.e. $M[t>$. Firing transition t yields a* *marking M' defined by: $M' = M - \bullet t + t\bullet$. This can be written as $M \xrightarrow{t} M'$.*

Let $\sigma = t_1 \dots t_n$ be a sequence of transitions (not all transitions need to be different) and M be a marking in which t_1 is enabled. For all $1 \leq i < n$, firing t_i yields a marking M_i in which t_{i+1} is enabled, and firing t_n yields marking M', i.e. $M \xrightarrow{t_1} M_1 \dots M_{n-1} \xrightarrow{t_n} M'$, then we write $M \xrightarrow{\sigma} M'$. We write $M \xrightarrow{*} M'$ to indicate that a transition sequence σ exists such that $M \xrightarrow{\sigma} M'$. For convenience, we allow σ to be the empty transition sequence, in which case $M = M'$.

Definition 5 (Petri net system)**.** *A Petri net system $\mathcal{S}$ is a Petri net (P,T,F) with a designated initial marking M_0, $\mathcal{S} = (P,T,F,M_0)$.*

Definition 6 (Reachable marking)**.** *Let $\mathcal{P} = (P,T,F,M_0)$ be a Petri net system. M is a reachable marking of $\mathcal{P}$ iff $M_0 \xrightarrow{*} M$.*

A workflow definition can be modelled as a Petri net, where tasks are captured by transitions, conditions by places. Based on the assumption that a typical workflow has a well-defined starting point and a well-defined ending point, van der Aalst defines workflow nets by imposing certain syntactical restrictions on Petri nets (see e.g. (van der Aalst, 1998a)).

Definition 7 (Workflow net (van der Aalst, 1998a))**.** *A Workflow net $\mathcal{P}$ is a Petri net (P,T,F) such that there is a unique source place i, $\bullet i = \varnothing$, a unique sink place o, $o\bullet = \varnothing$, and such that every node $x \in P \cup T$ is on a path from i to o.*

The initial marking for a workflow net, or *WF-net* for short, corresponds to one token in its initial place and thus formally corresponds to *i*. The desired final marking is the marking that has one token in the output place *o* and no tokens in any other places, i.e. it formally corresponds to *o*. As can be seen from Definition 7, *WF-nets* are solely concerned with control-flow dependencies between tasks and do not take conditions based on data into account.

A workflow net with data is a workflow net in which transitions (modeling tasks) can read from, write to, or delete data elements. A transition can have a (data- dependent) guard that blocks its execution when it is evaluated to false. In (Trčka et al., 2009), Workflow nets with data (that is, WFD-net) is defined formally.

Definition 8 (WFD-net (Trčka et al., 2009))**.** *A Workflow net with data $\mathcal{N}$= $(P,T,F,\mathcal{D},\mathcal{G}_{\mathcal{D}},rd,wt,del,grd)$ consists of*

- *$\mathcal{P} = (P,T,F)$ is a WF-net;*
- *$\mathcal{D}$ is a set of data elements;*
- *$\mathcal{G}_{\mathcal{D}}$ is a set of guards over $\mathcal{D}$;*
- *a reading data labeling function rd: $T \rightarrow 2^{\mathcal{D}}$;*
- *a writing data labeling function wt: $T \rightarrow 2^{\mathcal{D}}$;*
- *a deleting data labeling function del: $T \rightarrow 2^{\mathcal{D}}$;*
- *a guard function $grd : T \rightarrow \mathcal{G}_{\mathcal{D}}$ is the guarding function, assigning guards to transitions.*

It should be noted that no any concrete transition guard is defined in WFD-net.

2.2 Workflow nets with Transition Conditions

Central to *WTC-nets* is the notion of a *condition*. Here we only describe informally condition class *cond*. A condition *c* is an instance of *cond*.

The expression form of the class *cond* is a logical expression. The **operators** used in *cond* include comparison operators (eg. le, leq, ge, geq, eq), arithmetical operators (eg. $plus, minus, times, frac, mod, div$), boolean operators (eg. $neg, and, or, implies$). The data **type** in *cond* contains *int*, *real* and *bool*. For a **variable** *v* used in a *cond*, the **domain** of *v* is a set of values that *v* can hold. For instance, considering variables d_1 and d_2 in Figure 1: the type of variable d_1 is *bool* and the **domain** of d_1 is (true, false), the type of d_2 is *int* and the **domain** of d_2 is the range of integer. The notion *var* is used as the set of all variables as well as the notion *VAR* is used as the set of all possible assignments of types to variables.

In order to be able to focus on conditions that are well-formed (informally, well-formed condition requires that operations contained in the condition are applied to constants or variables of the correct type), function $V_{wfc}()$ determines whether a condition is well-formed. In addition, function $V_{var}()$ is applied to a condition to yield the variables that are used in this condition.

A *WF-net* with transition conditions, or *WTC-net* for short, is a *WF-net* where the transitions are assigned conditions. We give an assumption that the variables that a transition can write to, may take on any value as a result of its execution.

Definition 9 (Workflow net with transition conditions)**.** *A workflow net with transition conditions $\mathcal{W}$ is a tuple $(\mathcal{P},V,C,G,W,TV)$, such that:*

- *$\mathcal{P} = (P,T,F)$ is a WF-net;*
- *$V \subseteq var$ is a set of net variables;*
- *$C \subseteq cond$ is a set of transition conditions with $\cup_{c\in C} V_{var}(c) \subseteq V$ and for each $c \in C$, $V_{wfc}(c,TV)$ holds;*
- *$G : T \rightarrow C$ is a function assigning conditions to transitions;*

- $W : T \to 2^V$ *is a function that yields the variables that transitions can write to;*
- $TV \in VAR$ *is a function assigning a type to each net variable.*

Definition 10 (Marking of WTC-net)**.** *For a* WTC-net $\mathcal{W}= (\mathcal{P}, V, C, G, W, TV)$*, a marking* $\mathcal{M}$ *is a tuple* (M, σ)*, where* M *is a marking of* WF-net $\mathcal{P}$*, and* $\sigma \in T^*$ *is a sequence of transitions.*

Subsequently we formally define whether transition condition is satisfied in the context of a particular assignment of values to variables. Given an assignment as an environment e and the set of all possible environments as ENV. The set of all possible values is denoted as *Val* and it is defined as the union of all integers, reals, and booleans, i.e. $Val = \mathbb{Z} \cup \mathbb{R} \cup \mathbb{B}$.

Definition 11 (Satisfiability of conditions)**.** *Let* $TV : var \to Types$ *be an assignment of types to variables and let* c *be a* well-formed *condition in the context of type assignment* TV*, i.e.,* $c \in cond$ *such that* $V_{wfc}(c, TV)$*,* $e : var \to Val$ *an assignment of values to variables, and the function* $V_{holds} : cond \times ENV \to bool$ *can be applied to determine whether in the context of a given environment a condition holds.*

In order to evaluate if in the context of a transition sequence σ a given transition condition ϕ is hold, we need to take into account the variables that are used in the transition conditions of transitions in σ and in ϕ as well as the order in which variables may be written by transitions in σ. To solve this problem, we introduce *fresh variables* as replacements for variables used in transition conditions of transitions whenever it turns out that these variables could have received a new value through the execution of preceding transitions. Fresh variables in a transition sequence σ take the form v_t^n where t is a transition which could write v, i.e. $v \in W(t)$, and n is a natural number, which for a particular occurrence of transition t in σ represents the number of previous occurrences of t in σ (this is required in order to be able to deal with loops). We assume that variables of the form v_t^n do not occur as net variables in a WTC-net, so that indeed they will be fresh whenever introduced. We also assume that given a renaming function $\psi : var \rightarrowtail var$ and a condition $c \in cond$, $c[\psi]$ denotes condition c where for every variable $v \in V_{var}(c) \cap \mathsf{dom}(\psi)$ each of its occurrences has been replaced by $\psi(v)$.

Definition 12 (Conjunctive transition conditions)**.** *Let* $\mathcal{W}= (\mathcal{P}, V, C, G, W, TV)$ *be a* WTC-net*,* $\sigma \in T^*$ *a transition sequence (possibly empty, i.e.* $\sigma = \varepsilon$*),* $\psi : var \rightarrowtail var$ *a renaming of variables,* $G(t)[\psi]$ *changing transition condition according to this renaming,* $n : T \to \mathbb{N}$ *an assignment of natural numbers to transitions, and* b *is any boolean expression. The conjunctive transition conditions along sequence with variables renaming* $\rho^{\psi,\, n}(\sigma,\, b)$ *is defined by:*

$$\rho^{\psi,\, n}(\varepsilon,\, b) = b$$

$$\rho^{\psi,\, n}(t\sigma,\, b) = \rho^{\psi',\, n'}(\sigma,\, b \wedge G(t)[\psi]) \text{ where}$$

$$\psi' = \psi \oplus \{(v,\, v_t^{n(t)}) \mid v \in W(t)\} \text{ and } n' = n \oplus \{(t,\, n(t)+1)\}$$

We are now in a position to formally define when a transition in a *WTC-net* is enabled in the context of a given marking.

Definition 13 (Enabled transition in WTC-net)**.** *Let* $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ *be a* WTC-net*,* $\mathcal{M} = (M,\, \sigma)$ *be a marking of this net, and* $t \in T$ *a transition in this net. Let* b *be a conjunctive transition conditions with variables renaming defined by* $b = \rho^{\varnothing,\, \{(t,\, 0) \mid t \in T\}}(\sigma, true)$*. Transition* t *is enabled in marking* $\mathcal{M}$*,* $\mathcal{M}[t>$*, iff an environment* $e : V_{var}(b) \to Val$ *can be found such that* $V_{holds}(b,\, e,\, TV)$*.*

Definition 14 (Firing)**.** *Let* $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ *be a* WTC-net*,* $\mathcal{M} = (M, \sigma)$ *be a marking of this net, and* $t \in T$ *an enabled transition. Firing transition* t *in marking* $\mathcal{M}$ *results in marking* $\mathcal{M}' = (t \bullet \cup (M \setminus \bullet t), \sigma t)$*. As per usual, we write* $\mathcal{M} \xrightarrow{t} \mathcal{M}'$*.*

Example 1. Assuming for the WTC-net sample in the Fig. 1, given a current marking $(p_2, t_1t_2t_3t_2)$, let us consider transition t_4 can be enabled or not. Firstly, we observe how to derive the conjunctive form of transition conditions for transition t_4 along the sequence $t_1t_2t_3t_2t_4$. As shown in the first column of Table 1, variable d_1 is written twice by transition t_2. Once each writing operation happens the corresponding variable is mapped into a new fresh variable as shown in the second column of Table 1. For example, variable d_1 is mapped into $d_{1_{t_2}}^1$ and $d_{1_{t_2}}^2$ along the sequence $t_1t_2t_3t_2$. Then, the corresponding transition condition is changed based on the closest preceding variable renaming as shown in the third column in Table 1. For instance, the transition condition of transition t_3 in the sequence is changed as $\neg d_{1_{t_2}}^1$ following the closest preceding variable renaming $d_{1_{t_2}}^1$ as well as the transition condition of transition t_4 in the sequence is changed as $d_{1_{t_2}}^2$ following the closest preceding variable renaming $d_{1_{t_2}}^2$. The fourth column in Table 1 describes the dynamic evolution of the conjunctive form of transition conditions along the sequence. For the conjunctive form of transition conditions $\neg d_{1_{t_2}}^1 > 5 \wedge d_{1_{t_2}}^2$, there is at least an assignment of value *false* to $d_{1_{t_2}}^1$ and value *true* to $d_{1_{t_2}}^2$ which make the conjunctive form hold. Thus, transition t_4 is enabled in the current marking $(p_2, t_1t_2t_3t_2)$. Once transition t_4 is fired, the marking $(p_3, t_1t_2t_3t_2t_4)$ is reached.

Subsequently, assuming that in the marking $(p_3, t_1t_2t_3t_2t_4)$ transition t_6 is fired and marking

$(p_5, t_1t_2t_3t_2t_4t_6)$ is reached. let us consider transition t_9 can be enabled or not. Analogously, the conjunctive form of transition conditions for transition t_6 along $t_1t_2t_3t_2t_4t_6t_9$ is $\neg d1^1_{t_2} > 5 \wedge d1^2_{t_2} \wedge d2^1_{t_2} \leq 50,000 \wedge d2^1_{t_2} > 50,000$. It is obviously that there isn't an assignment of value for variable $d2^1_{t_2}$ which makes the $\neg d1^1_{t_2} > 5 \wedge d1^2_{t_2} \wedge d2^1_{t_2} \leq 50,000 \wedge d2^1_{t_2} > 50,000$ satisfiable. Thus, transition t_9 can not be enable.

Discussion. The state space of the WTC-net is explored in the term of symbolic state in the form (M, σ) where M is a control flow marking and a historic constraint system in the form of a conjunction of transition conditions is accumulated by traversing the firing sequence σ. For a transition t where $M[t>$, we further probe that the transition condition on t (if any) is satisfiable under the historic constraint system. In order to avoid the state explosion caused by naive exploration of the data dimension, we adopt the symbolic execution technique (King, 1976; Clarke, 1976) that converts the reachability problem to that of solving simple constraints systems (Kumar, 1992; Tsang, 1993; Ganzinger et al., 2004). That is, any vector of values assigning to variables satisfying the historic constraint system and the transition conditions will drive the WTC-net to be executed by firing the transition t. For example, for the transition t_6 in the state $(p_3, t_1t_2t_3t_2t_4t_6)$, there are many possible assignments of values to variables $d1^1_{t_2}$, $d1^2_{t_2}$, and $d2^1_{t_1}$ that make the constraint $\neg d1^1_{t_2} \wedge d1^2_{t_2} \wedge d2^1_{t_1} \leq 50,000$ satisfiable. Thus, one symbolic execution may correspond to many real executions.

Definition 15 (Initial marking). *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net*, the initial marking of this net $\mathcal{M}_0$ is defined as (i, ε), where i is the unique source place of $\mathcal{W}$.*

Definition 16 (Reachable marking). *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net, *a marking (M, σ) with $\sigma = t_1 \ldots t_n$ is reachable iff a sequence of reachable markings $M_0 M_1 \ldots M_n$ exists such that $(i, \varepsilon) \xrightarrow{t_1} (M_1, t_1) \xrightarrow{t_2} (M_2, t_1t_2) \xrightarrow{t_3} \ldots \xrightarrow{t_{n-1}} (M_{n-1}, t_1 \ldots t_{n-1}) \xrightarrow{t_n} (M_n, t_1 \ldots t_n)$ with $M_n = M$.*

2.3 Infeasible Trace of WTC-net

In this section, we will describe formally the semantics of infeasible trace of Workflow nets with transition conditions focusing on the correlation relationship between transition conditions along a transition sequence.

Definition 17 (Infeasible Trace). *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net, *(M, σ) be a marking of $\mathcal{W}$,* t *be a transition with guard function $G(t)$, $\sigma = t_1 ... t_n$ is the **executable prefix** of transition* t *as well as trace $\sigma \cdot t$ is **infeasible**, iif:*

i. $M[t >$, and

ii. there is an environment $e : var \cup \mathrm{ran}(\psi) \rightarrowtail Val$ such that for all $1 \leq i \leq |\sigma|$, $V_{holds}(\rho^{\varnothing, \{(t,\,0)\ |\ t \in T\}}(\sigma, true), e, TV)$, and $\neg(V_{holds}(\rho^{\varnothing, \{(t,\,0)\ |\ t \in T\}}(\sigma.t, true), e, TV))$.

Definition 18 (Correlation between transition conditions). *Given two transition conditions c_1 and c_2 assigned to guard function of t_1 and t_2 respectively. They are correlated, iif:*

(1) c_1 and c_2 reference common condition variables, i.e, they have common condition variable set $V_c = (v_1, v_2, ..., v_m)$,

(2) there is a trace from t_1 to t_2 such that at least one variable in V_c isn't written.

3 DETECTING INFEASIBLE TRACE OF WTC-NETS

This section illustrates the concrete algorithms to detect infeasible traces of WTC-nets. These algorithms are based on the reachability analysis of a WTC-net system. The concrete method is: firstly, an extended coverability graph algorithm is proposed; then a backward search algorithm is proposed to search the shortest infeasible trace.

3.1 Generating Coverability Graph

The coverability graph algorithm supports the creation of a coverability graph based on the reachability notion of a WTC-net defined in Section 2.

The whole process for generating a coverability graph of a WTC-net is similar with the algorithm that Finkel has proposed in (Finkel, 1991) to construct a Petri net coverability tree and to obtain a coverability graph from a coverability tree by merging the identical nodes. Here we briefly introduce the whole process of generating coverability tree: given a WTC-net $\mathcal{W}$ with $\mathcal{M}_0$, from the initial marking $\mathcal{M}_0$, many new markings as the number of *the enabled transitions* can be obtained. From each *new* marking, more markings can be reached. This process results in a coverability tree. Nodes represent markings generated from $\mathcal{M}_0$ (the root) and its successors, and each arc represents a transition firing, which transforms one marking to another. Specially, three different cases should be considered when a new node is explored to its successors:

- *Case 1 - Detection of an identical known marking.* For current node n, there is a node n_1 such that n_1

Table 1: Deriving the conjunctive form of transition conditions along sequence $t_1t_2t_3t_2t_4t_6t_9$ for WTC-net in Figure 1.

TS[1]	VR[2]	TCT[3]	CTC[4]
t_1	$d2_{t_1}^1, d3_{t_1}^1$		
t_2	$d1_{t_2}^1$		
t_3		$\neg d1_{t_2}^1$	$\neg d1_{t_2}^1$
t_2	$d1_{t_2}^2$		$\neg d1_{t_2}^1$
t_4		$d1_{t_2}^2$	$\neg d1_{t_2}^1 \wedge d1_{t_2}^2$
t_6		$d2_{t_1}^1 \leq 50,000$	$\neg d1_{t_2}^1 \wedge d1_{t_2}^2 \wedge d2_{t_1}^1 \leq 50,000$
t_9		$d2_{t_1}^1 > 50,000$	$\neg d1_{t_2}^1 \wedge d1_{t_2}^2 \wedge d2_{t_1}^1 \leq 50,000 \wedge d2_{t_1}^1 > 50,000$

[1] Transition Sequence.
[2] Variable Renaming.
[3] Transition Condition Transformation.
[4] Conjunctive Transition Conditions.

is an ancestor of n and n_1 has *the same anchor label* of n: identify n and n_1, then tag this node as *old* and shift to process other new node.

- *Case 2 - non boundedness detection.* For current node n, there is a node n_1 such that n_1 is an ancestor of n and $n.M > n_1.M$, i.e., n is coverable, then replace $n.M(p)$ by ω for each p such that $n.M(p) > n_1.M(p)$. Finally, node n is tagged *new* again and shift to process other *new* node.
- *Case 3 - new nodes generation.*

As described above, generating coverability graph with transition conditions is a standard coverability graph algorithm proposed by Finkel in (Finkel, 1991), with extra features: the ability to decide the enable of transition considering the satisfiability of transition conditions along a certain sequence of transitions and, the ability to decide the termination of the repeating cyclic executions.

On one hand, as defined in Definition 13, the enable of transition is determined not only by the underlying WF-net marking, but also by the satisfiability result of the condition attached to the transition (if any). Furthermore, the satisfiability result of the condition attached to the current transition is influenced by the preceding transition conditions as well as writing operations and writing orders in preceding transitions along a given transition sequence on the variables used by current transition condition.

Satisfiability (Algorithm 1) For certain transition which is enabled based on the WF-net marking of the current WTC-net marking, we should determine further whether the transition is enable considering the conjunctive transition conditions. Firstly, we generate the conjunctive transition conditions(line 4) following the Definition 12. Then,we decide the satisfiability of the conjunctive transition conditions following the Definition 11(line 5).

Termination of repeating cyclic execution In general, we can not terminate the repeating cyclic execution in a reachable set of a WTC-net system by judging directly the same WF-net marking merging twice along a sequence, because a marking of a WTC-net contains both the marking of the underlying WF-net and the firing sequence. However, there is the same local constraint of transition-condition pairs by traversing the cycle for the start state of the second and latter round of a cyclic execution while that is unsuitable for start state of the first round. For example, a local constraint $\{(t_3, c_1)\}$ is derived by traversing cycle t_2t_3 in Figure 1. And, for any start states of different round, they contain the common entry place of the cycle which can be decided from the transition invariant (that is, T-invariant) (Murata, 1989). For example, there is a transition invariant $t_2 + t_3$ corresponding to the cycle t_2t_3 in Figure 1, then the entry place for cycle t_2t_3 in the WTC-net is p_1. Thus, we can terminate the repeating cyclic executions by identify the start state of the second cyclic execution

Algorithm 1: $Enable_{sat}()$

```
    Input: $\mathcal{W}$: WTC-net, t: transition in WTC-net,
           $\mathfrak{n}_j$: a WTC-net marking
    Output: bool
1.1 if $\mathcal{W}$.G(t)=null then
1.2     return true;
1.3 if $\mathcal{W}$.G(t)$\neq$ null then
1.4     $\rho := true; \psi := \phi; n := \phi; b = \rho^{\varnothing,\ \{(t,\ 0)\ |\ t \in T\}}(\sigma, true)$;
1.5     if $(\exists e : V_{var}(b) \to Val, V_{holds}(b,\ e,\ TV))$
        then
1.6         return true;
1.7     else
            return false;
```

and that of the third cyclic execution by the same local constraint and the WF-net marking containing the same entry place $((p_1),\{(t_3,d_1)\})$. The concrete analysis and proof will be discussed in other paper. In view of the space, here we don't discuss in detail about the method.

Based on the extended coverability graph Algorithm, the constructed graph of Figure 1 is shown in Figure 2 [An aside: any node with gray cycle is a dead node, node 2 (p_1,t_1) is the start state of the first cyclic execution t_2t_3, node 4 and node 9 labeled with $(p_1,\{(t_3,c_1)\})$ are the start state of the second round and third round of cyclic execution t_2t_3 respectively, Thus, the repeating of the cyclic executions is terminated in the start state of the third round by merging the node 9 to node 4 with the directed dotted arc].

3.2 Detecting Shortest Infeasible Traces

Here, we illustrate how to detect the shortest infeasible traces based on the coverability graph.

The backward-search algorithm detects the shortest infeasible traces in workflow model. Each shortest infeasible trace begins from an infeasible transition extracted from an infeasible trace n_j. Then, an inverse transition trace σ' is searched backward step by step. Accordingly an inverse conjunction of transition conditions along the inverse trace is derived. This Algorithm iteratively processes the predecessor of the inverse trace until the inverse conjunction of conditions is unsatisfied.

Algorithm 2: Detecting Shortest Infeasible Traces

```
    Input: $\mathcal{W}$: WTC-net, $\mathbb{T}$: reachability tree, $S_{in}$:
           a set of firing sequences for infeasible
           traces
    Output: SIT: a set of infeasible traces with
            condition correlation
2.1 for $n_j$ in $S_{in}$ do
2.2     $\sigma' := \phi; b := null$;
2.3     for $i \leftarrow |n_j.\sigma|$ to 1 do
            $\sigma' := \sigma' \cup n_j(i)$;
            $b := \rho^{\varnothing,\ \{(t,\ 0)\ |\ t \in T\}}(\sigma', true)$;
2.4         if $e : V_{var}(b) \rightarrow Val, \neg V_{holds}(b,\ e,\ TV)$
            then
                $SIT := SIT \cup (n_j, n_j[|n_j|], n_j[i])$;
                skip;
```

Figure 3 shows the result of this detecting algorithm. The set of infeasible traces $t_1t_2(t_3 t_2)^*t_4t_5t_8t_{10}$ is caused by correlation relationship between transition condition c_3 on t_5 and transition condition c_4 on t_{10} as well $t_1t_2(t_3t_2)^*t_4t_6t_9$ caused by correlation relationship between transition condition c_4 on t_6 and transition condition c_3 on t_9.

4 TOOL SUPPORT

We implemented the proposed approach for detecting the infeasible traces for the system of workflow nets with transition conditions. We made use of a well-known, open-source process analysis framework (ProM) (van der Aalst et al., 2009) and developed our tool as a ProM 6 analysis plug-in[1]. The tool is made up of two main parts: the WTC-net constructor and the infeasible-trace detector.

4.1 WTC-net constructor

The WTC-net constructor provides the design environment for developing workflow nets with well-formed transition conditions. The two main components are the component of WTC-net editor and the component of WTC-net checker.

1. *WTC-net editor*: The editor supports the creation of a new WTC-net, and the importing and exporting of WTC-nets. On one hand, the editor supports the creation of a new WTC-net by importing WF-net model and editing the data information related with transition conditions in this editor based on the imported WF-net model. On the other hand, this editor supports importing and exporting WTC-net models.
2. *WTC-net Checker*: The function of this component is twofold: (1) to check whether the control flow structure of a WTC-net is a WF-net, and (2) to check the semantic correctness of transition conditions. For the purpose of the WF-net checking, we made use of an existing ProM plugin called Woflan (Verbeek et al., 2001).

The interface of the WTC-net constructor is shown in Figure 4. The left part of this screenshot illustrates the definition of data variables as well as the definition of transition conditions. The right part of this screenshot describes the underlying WF-net structure.

4.2 Infeasible-trace detector

The Infeasible-trace detector supports the creation of a coverability graph as well as the detection of infea-

[1]The ProM framework can be downloaded from `http://www.processmining.org/`

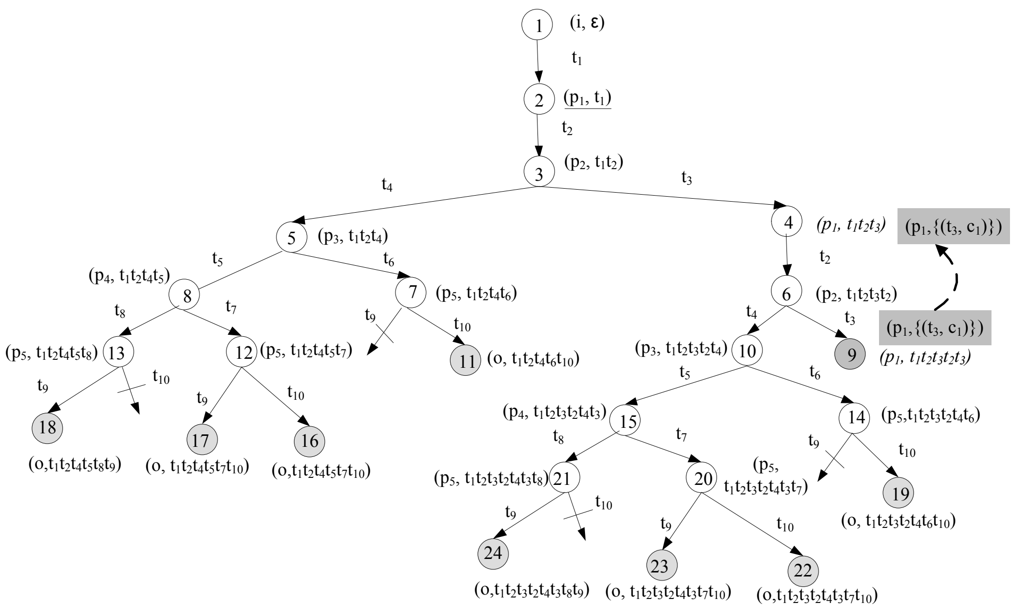
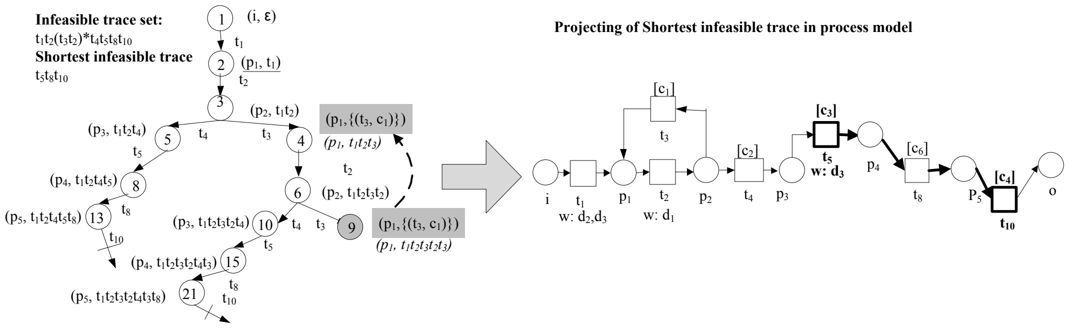
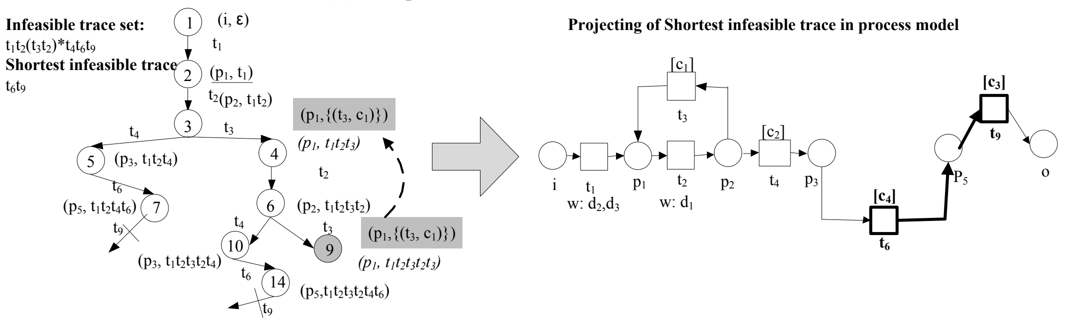


Figure 2: The coverability graph of the WTC-net in Figure 1.



Infeasible trace set:
$t_1t_2(t_3t_2)^*t_4t_5t_8t_{10}$
Shortest infeasible trace
$t_5t_8t_{10}$
Projecting of Shortest infeasible trace in process model

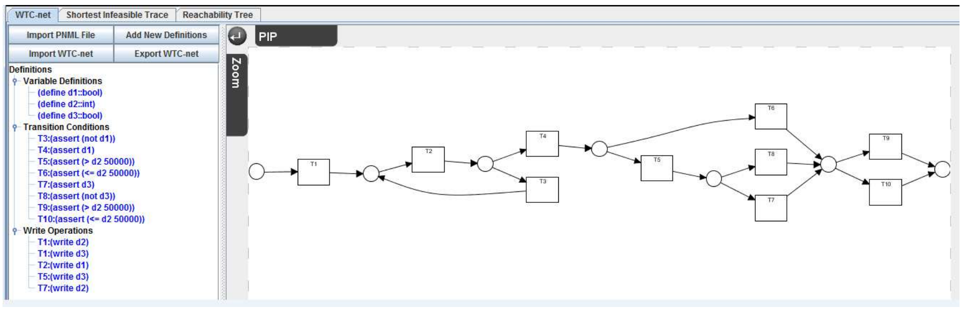
(a) Sample 1 for shortest infeasible traces



Infeasible trace set:
$t_1t_2(t_3t_2)^*t_4t_6t_9$
Shortest infeasible trace
t_6t_9
Projecting of Shortest infeasible trace in process model

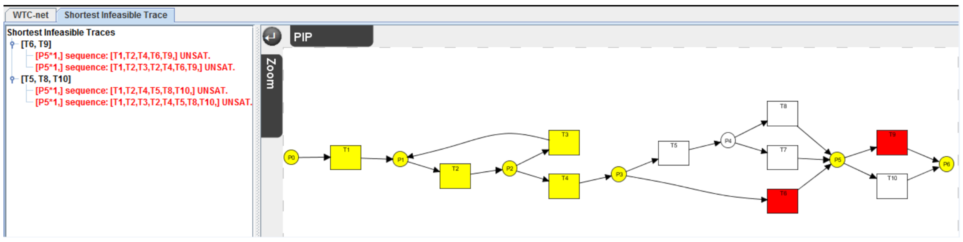
(b) Sample 2 for shortest infeasible traces

Figure 3: Backwards detecting shortest infeasible trace.



```
Definitions
  Variable Definitions
    (define d1::bool)
    (define d2::int)
    (define d3::bool)
  Transition Conditions
    T3:(assert (not d1))
    T4:(assert d1)
    T5:(assert (> d2 50000))
    T6:(assert (<= d2 50000))
    T7:(assert d3)
    T8:(assert (not d3))
    T9:(assert (> d2 50000))
    T10:(assert (<= d2 50000))
  Write Operations
    T1:(write d2)
    T1:(write d3)
    T2:(write d1)
    T5:(write d3)
    T7:(write d2)
```

Figure 4: The screenshot of the WTC-net constructor interface.



```
Shortest Infeasible Traces
  [T6, T9]
    [P5*1,] sequence: [T1,T2,T4,T6,T9,] UNSAT.
    [P5*1,] sequence: [T1,T2,T3,T2,T4,T6,T9,] UNSAT.
  [T5, T8, T10]
    [P5*1,] sequence: [T1,T2,T4,T5,T8,T10,] UNSAT.
    [P5*1,] sequence: [T1,T2,T3,T2,T4,T5,T8,T10,] UNSAT.
```

Figure 5: The screenshot of the detecting result of the shortest infeasible traces.

sible traces.

Here, we specially introduce the implementation of the satisfiability of conjunctive transition conditions during the generation of the coverability graph. We developed a Java interface to call the libraries in Yices (B. Dutertre and L. de Moura,) to justify the satisfiability of the condition. Yices (B. Dutertre and L. de Moura,) is a SMT (Satisfiability Modulo Theories) tool, which is developed in the C language at SRI International, integrating a theory solver (that checks the satisfiability of conjunctive formulae over a theory) with a SAT (Boolean Satisfiability) solver that enumerates over a Boolean abstraction of the entire formula.

Subsequently, the backward searcher is implemented to detect the infeasible trace and to return the detecting result to the WTC-net constructor. The left part of Figure 5 shows the returned result of detecting the infeasible traces in Figure 1: there are 2 sets of infeasible traces, one set is with the shortest infeasible trace t_6t_9 as well as the other set is with the shortest infeasible trace $t_5t_8t_{10}$. The right part of the screenshot illustrates the set of infeasible trace with the shortest infeasible trace t_6t_9 where all transitions of the infeasible traces set (except that transitions t_6 and t_9) are yellow. The assigned conditions for transition t_6 and t_9 respectively are correlated, then the red color filled on transitions t_6 and t_9 to make the correlation relationship more clear.

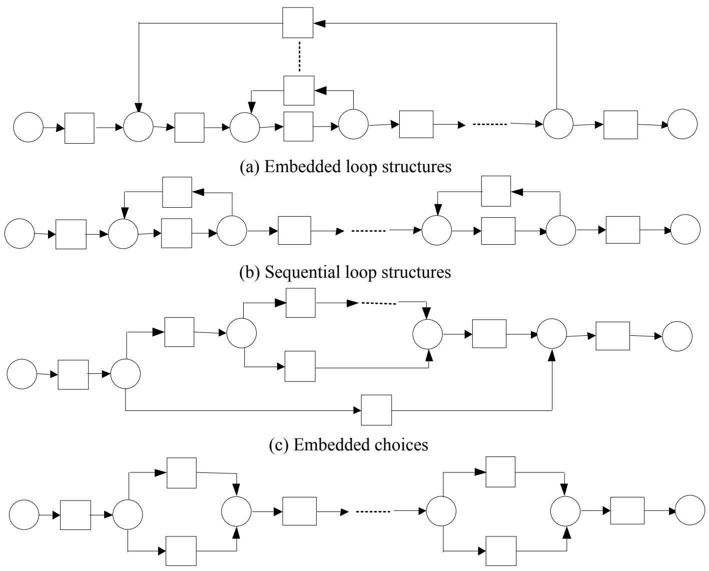
5 Experiments

5.1 Feasibility assessment

In this section, we evaluate the feasibility of our approach by assessing the running time of some typical artificial examples.

To the best of our knowledge, there isn't other verification tool specially developed for workflow net with data perspective. Thus we can not assess the feasibility of our approach by comparing the execution time among different tools. In addition, the complexity of the WTC-net is mainly determined by the satisfiability of the constraint. Then, we have specially designed artificial examples and tested the consuming time of generating coverability graph for these artificial examples. The running environment is an Acer laptop with the configuration of 2.67GHz Intel Core i5-480M CPU and 4.0GB memory.

We consider four representive situations related to conditional routing: the first is the embedded loop structures as shown in Fig. 6a, the second is the sequential loop structures as shown in Fig. 6b, the third is the embedded choices as shown in Fig. 6c, and the final is the sequential choices as shown in Fig. 6d. Moreover, we designed artificial WTC-net models with incremental basic conditional constructs from 2 to 6 for each situation. For example, for the situation of embedded loop structures, there are 5 artificial WTC-net models with embedded loop structures from 2 to 6 respectively. Thus, there were 20 artificial WTC-net models for the four situations. Ta-


(a) Embedded loop structures

(b) Sequential loop structures

(c) Embedded choices

(d) Sequential choices

Figure 6: Construct samples

Table 2: Time taken for the computation of the coverability graphs for artificial examples*

	$Time_{\#(2)}$	$Time_{\#(3)}$	$Time_{\#(4)}$	$Time_{\#(5)}$	$Time_{\#(6)}$
Embedded loops	241	292	701	5434	27624
Sequential loops	231	241	282	331	471
Embedded choices	171	171	191	181	171
Sequential choices	161	141	171	271	343

*$Time_{\#(n)}$ stands for how many milliseconds taken for the WTC-net with n constructs

ble 2 shows the consuming time for the different situations. The second row describes that the consuming time soon mounts up when the number of the embedded loop structures increases gradually. However, the third row, the fourth row and the fifth row represent that the consuming time stably increases when the number of the sequential loop structures, embedded choices and sequential choices increases gradually. The statistical information in Table 2 testifies again that the length of the consuming time is decided mainly by the number of the embedded loop structures. Especially when the number of the embedded loop structures is amount to 5 or above 5, the consuming time increases dramatically. However, there isn't important influence on the length of the consuming time for other complexity conditional routing situations (e.g., embedded choices, sequential choices and sequential loop structures).

In general, the testing results of the consuming time prove that the tool developed for our approach is feasible in practise to some extent.

5.2 Detecting the infeasible traces

Subsequently, we illustrate the applicability of our approach to 29 real process models by carrying out experiments that are based on process models and data information related with transition conditions found in the TiPLM system[2]. These 29 real process models are divided into 4 types: engineering design and review (DR), process engineering and changing (PC), release management (RM), and application management (AM). Firstly, we transformed the TiPLM workflow models and the extracted data information from TiPLM database into WTC-net models. Then, we used the implemented tool to detect the infeasible traces. Table 3 shows the result of the detection. Table 3 shows that there are 8 (27.6%) process models among the 29 transformed WTC0net models. We compared the detecting results with other measures including

[2]TiPLM is a product life-cycle management solution, which is developed by THsoft InfoTech company (http://www.thit.com.cn) and widely used in Mainland China (well over 100 companies in the manufacturing industry in China adopted the TiPLM system).

Table 3: The detecting result of infeasible traces for real process models*

PM	DR-1	DR-2	DR-3	DR-4	DR-5	DR-6	DR-7	DR-8	DR-9	DR-10
$SIT_{\#}$	0	2	26	0	0	0	1	0	0	0
PM	DR-11	PC-1	RM-1	RM-2	RM-3	RM-4	RM-5	RM-6	RM-7	RM-8
$SIT_{\#}$	11	0	0	2	1	0	0	0	0	0
PM	RM-9	RM-10	RM-11	RM-12	RM-13	AM-1	AM-2	AM-3	AM-4	
$SIT_{\#}$	0	0	0	1	0	0	1	0	0	

**PM* describes process model, $SIT_{\#}$ denotes the number of the set of infeasible traces possessing the same shortest infeasible traces

manual analysis. It turned out that our approach was always able to correctly detect the correlation relationship of the process definitions. All examples for experiments can be downloaded from `http://laudms.thss.tsinghua.edu.cn/trac/Test/attachment/wiki/chengguo/Test%20Data.zip`.

6 RELATED WORK

In recent years, there are formalisms for workflow modeling taking data perspective into account and the corresponding analysis approaches are then proposed based on the formalisms. Some researches (Trčka et al., 2009; Sun et al., 2006; Meda et al., 2007; Meda et al., 2010) focus on the data flow anomalies in workflow. Fan et al. in (Fan et al., 2007) extend the notion of classical soundness from (van der Aalst, 1998a) to support the case when data flow effect control flow. However, there is no explicit data perspective considered explicitly in the verification algorithm. Since the control/data flow interaction is considered in the transition conditions which are used to generate occurrence graph (OG), actually data parts can also impact the soundness property. Thus, this formal verification method is not efficient enough. In (Knuplesch et al., 2010), Knuplesch et al. consider data conditions within a process model serving as preprocessing step to the actual compliance checking rather than correctness verification of business process model. The influence of data condition on the flow of control was first mentioned in (Trčka, 2009). Here, two different situations with data perspective are identified, like false positives or false negatives, but no means for verifying soundness property is provided. Later, Sidorova et al. in (Sidorova et al., 2010; Sidorova et al., 2011) propose an approach to verify the soundness property of workflow net considering data perspective. The concrete approach in (Sidorova et al., 2010) is to construct a may-must reachibility graph based on standard may/must transition systems that guaranties the results to be valid considering with data perspective (e.g., data operations and abstract transition guards). Further, the approach proposed in (Sidorova et al., 2011) defines the semantics of WFD-nets using the concept of hyper transition systems rather than standard may/must transition systems to improve their previous characterizations of may- and must- soundness from (Sidorova et al., 2010) and to prove in theory the tough relationship between conceptual WFD-net with abstract predicate and the corresponding data refinement. It should be noted that the above researches consider about the influence of data perspective in verification of workflow correctness.

For us, WTC-net is proposed for studying workflow nets where routing may be determined by transition conditions. Further, we propose symbolic execution semantic of WTC-net. Finally, the infeasible traces are detected based on the correlation between transitions based on the analysis of symbolic reachability.

7 CONCLUSIONS

In this paper, we have defined workflow net with transition conditions *WTC-net* and its behavior semantics with the influence of transition condition in detail. In addition, we have presented an approach to detect infeasible trace and a tool has been implemented based on the approach.

In the future work, we will consider how to verify process model considering with transition condition in quantity. We will focus on how to deal with the volume of the solver space of trace condition and provide more precise information for model designer, who can further decide to modify the control flow structure or the tradition condition with the result of the detection.

ACKNOWLEDGEMENTS

This paper is supported by the 863 High-Tech Development Program of China (No. 2009AA043401), the National Science Foundation of China (No.

61003099, 61073005) and the National Science and Technology Major Project, China (No. 2010ZX01042-002-002-01).

REFERENCES

B. Dutertre and L. de Moura. The YICES SMT Solver. `http://yices.csl.sri.com/tool-paper.pdf`. Accessed Mar 2011.

Clarke, L. A. (1976). A system to generate test data and symbolically execute programs. *IEEE Trans. Software Eng.*, 2(3):215–222.

Fan, S., Dou, W., and Chen, J. (2007). Dual workflow nets: Mixed control/data-flow representation for workflow modeling and verification. In Chang, K. C.-C., Wang, W., 0002, L. C., Ellis, C. A., Hsu, C.-H., Tsoi, A. C., and Wang, H., editors, *Advances in Web and Network Technologies, and Information Management, APWeb/WAIM 2007 International Workshops: DBMAN 2007, WebETrends 2007, PAIS 2007 and ASWAN 2007, Huang Shan, China, June 16-18, 2007, Proceedings*, volume 4537 of *Lecture Notes in Computer Science*, pages 433–444. Springer.

Finkel, A. (1991). The minimal coverability graph for petri nets. In Rozenberg, G., editor, *Applications and Theory of Petri Nets*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer.

Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2004). DPLL(T): Fast decision procedures. In Alur, R. and Peled, D., editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer.

King, J. C. (1976). Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394.

Knuplesch, D., Ly, L. T., Rinderle-Ma, S., Pfeifer, H., and Dadam, P. (2010). On enabling data-aware compliance checking of business process models. In Parsons, J., Saeki, M., Shoval, P., Woo, C. C., and Wand, Y., editors, *Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, volume 6412 of *Lecture Notes in Computer Science*, pages 332–346. Springer.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44.

Meda, H. S., Sen, A. K., and Bagchi, A. (2007). Detecting data flow errors in workflows: A systematic graph traversal approach. In *Proceedings of the 17th Workshop on Information Technologies and Systems*, pages 133–138.

Meda, H. S., Sen, A. K., and Bagchi, A. (2010). On detecting data flow errors in workflows. *J. Data and Information Quality*, 2(1).

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Ngo, M. N. and Tan, H. B. K. (2008). Heuristics-based infeasible path detection for dynamic test data generation. *Inf. Softw. Technol.*, 50(7-8):641–655.

Sidorova, N., Stahl, C., and Trčka, N. (2010). Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In Pernici, B., editor, *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings*, volume 6051 of *Lecture Notes in Computer Science*, pages 530–544. Springer.

Sidorova, N., Stahl, C., and Trčka, N. (2011). Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.*, 36(7):1026–1043.

Sun, S. X., Zhao, J. L., and Nunamaker, J. F. (2006). Formulating the data-flow perspective for business process management. *Information Systems Research*, 17(4):374–391.

Trčka, N. (2009). Workflow soundness and data abstraction: Some negative results and some open issues. In *APNOC'09: International Workshop on Abstractions for Petri Nets and Other Models of Concurrency, Paris, France, June 22, 2009*, pages 19–25.

Trčka, N., van der Aalst, W., and Sidorova, N. (2009). Data-flow anti-patterns: Discovering data-flow errors in workflows. In van Eck, P., Gordijn, J., and Wieringa, R., editors, *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer-Verlag.

Tsang, E. P. K. (1993). *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press.

van der Aalst, W. (1998a). The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

van der Aalst, W. (1998b). *Information and Process Integration in Enterprises: Rethinking documents*, chapter 10: Three Good Reasons for Using a Petri-net-based Workflow Management System, pages 161–182. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Norwell.

van der Aalst, W. and ter Hofstede, A. H. M. (2000). Verification of workflow task structures: A Petri-net-baset approach. *Inf. Syst.*, 25(1):43–69.

van der Aalst, W., van Dongen, B., Günther, C., Rozinat, A., Verbeek, H., and Weijters, A. (2009). ProM: The process mining toolkit. In de Medeiros, A. and Weber, B., editors, *Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009), Ulm, Germany, September 8, 2009*, volume 489 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Verbeek, H., Basten, T., and van der Aalst, W. (2001). Diagnosing workflow processes using Woflan. *Comput. J.*, 44(4):246–279.