Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Verification of workflow nets with transition conditions*

Zhao-xia WANG[†1,2,3,4,5], Jian-min WANG[†‡1,3,4], Xiao-chen ZHU[1,2,3,4], Li-jie WEN[1,3,4]

([1]*School of Software, Tsinghua University, Beijing 100084, China*)

([2]*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*)

([3]*MOE Key Laboratory for Information System Security, Tsinghua University, Beijing 100084, China*)

([4]*National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China*)

([5]*Department of Logistical Information & Engineering, Logistical Engineering University, Chongqing 400016, China*)

[†]E-mail: wang-cx06@mails.thu.edu.cn; jimwang@thu.edu.cn

Received Dec. 8, 2011; Revision accepted Apr. 9, 2012; Crosschecked May 31, 2012

Abstract: Workflow management is concerned with automated support for business processes. Workflow management systems are driven by process models specifying the tasks that need to be executed, the order in which they can be executed, which resources are authorised to perform which tasks, and data that is required for, and produced by, these tasks. As workflow instances may run over a sustained period of time, it is important that workflow specifications be checked before they are deployed. Workflow verification is usually concerned with control-flow dependencies only; however, transition conditions based on data may further restrict possible choices between tasks. In this paper we extend workflow nets where transitions have concrete conditions associated with them, called WTC-nets. We then demonstrate that we can determine which execution paths of a WTC-net that are possible according to the control-flow dependencies, are actually possible when considering the conditions based on data. Thus, we are able to more accurately determine at design time whether a workflow net with transition conditions is sound.

Key words: Workflow nets, Transition condition, Verification, Process model
doi:10.1631/jzus.C1100364 **Document code:** A **CLC number:** TP311

1 Introduction

Workflow management systems provide support for the execution of business processes. On the basis of a process model, they route tasks to the right resources at the right time. They also maintain data produced by tasks. This data is possibly used as input to other tasks and can be used to evaluate conditions associated with routing decisions. Thus, data produced by tasks may influence the choice of subsequent tasks.

As workflows may take weeks, months, or even years to complete, it is important that errors be detected before they are deployed. Workflow verification has been a topic of research for well over a decade and is concerned with detecting errors in workflow specifications that may, for example, result in the workflow deadlock or improper termination.

Workflow nets (WF-net) (van der Aalst and van Hee, 2002) are a subclass of Petri nets (Murata, 1989) of which the benefits of workflow management have been argued in van der Aalst (1998b), who provided a well-accepted theory for workflow specification and analysis. The notion of soundness defines correctness criteria for workflow nets. These criteria are concerned with determining whether a workflow can always terminate, whether it terminates properly, and whether every task in the workflow can be performed in some scenario. Soundness has been studied in great detail and a tool, Woflan (Verbeek *et al.*, 2001), exists that can detect whether a workflow net is sound.

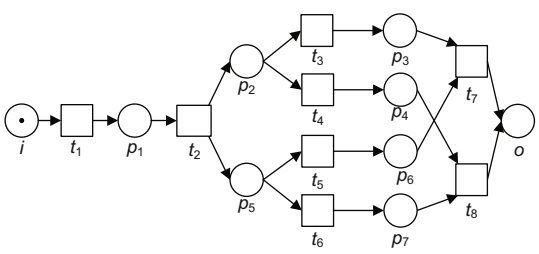
Workflow verification research has been

‡ Corresponding author

* Project supported by the National Science and Technology Major Project of China (No. 2010ZX01042-002-002-01), the National Basic Research Program (973) of China (No. 2009CB320700), and the National Natural Science Foundation of China (Nos. 61073005 and 61003099)

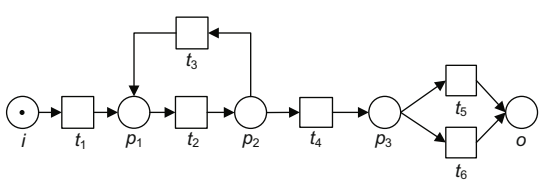
primarily concerned with control-flow dependencies between tasks. Data may, however, determine whether certain paths in the workflow graph are actually possible at runtime. As pointed out in Trčka (2009), workflow verification based on control-flow dependencies may produce only false positives or false negatives. A false positive corresponds to the case where a workflow is analyzed and determined to be sound while in actuality, due to the conditions specified, it is unsound. A false negative corresponds to the case where a workflow is determined to be unsound while it is actually sound. Trčka *et al.* (2009) presented the concept of workflow nets with data (WFD-nets), which extends the classical workflow nets with data operations (e.g., read, write, delete, and update) as well as abstract predicates as guard functions of transitions. Sidorova *et al.* (2010; 2011) generalized WFD-net as a conceptual workflow model, formalised the may- and must-soundness of the conceptual workflow model with data (WFD-net) and proved the tough relationship between the soundness result of a conceptual WFD-net and that of the corresponding data refinement in theory: if a WFD-net is proven must-sound, then it is sound in any possible data refinement; if it is not may-sound, then no data refinement can make it sound; if a WFD-net is may-sound, there is at least one sound refinement of it. Sidorova *et al.* (2010; 2011) answered with certainty that any data refinement is sound if the WFD-net is must-sound and that any data refinement is unsound if the WFD-net is not may-sound. However, given a may-sound WFD-net, there is not a certain soundness result for any data refinement. The answer is "I do not know. Sometimes the workflow is sound, sometimes it is not", because the soundness property depends on the concrete data refinement. In practice, the certain soundness result of any data refinement is very important to assist the modeler in modifying or improving the workflow model in the design time. Thus, further analysis based on concrete data refinement is needed.

To illustrate the necessities of analysing workflow models based on concrete data refinement, two examples in Fig. 1 illustrate that an accurate soundness result rather than just an abstract predicate will be derived when the concrete data refinement is considered. For variable x used in Fig. 1a, we assume that x is written by executing transition t_1 and cannot be modified in a parallel structure. This assumption is reasonable, as there will be data-flow anti-patterns 'inconsistent data' if variable x is modified by transition t_3, t_4, t_5, or t_6 (Trčka *et al.*, 2009). For variable y used in Fig. 1b, we consider three possible situations of writing operation on y:



Data refinement 1		Data refinement 2	
Task	Condition	Task	Condition
t_3	x>50 000	t_3	x>60 000
t_4	x≤50 000	t_4	x≤60 000
t_5	x>50 000	t_5	x>50 000
t_6	x≤50 000	t_6	x≤50 000

(a)



Task	Condition
t_3	y>5
t_4	y≤5
t_5	y>5
t_6	y≤5

(b)

Fig. 1 Samples of workflow net with different conditions. (a) Sample 1; (b) Sample 2. For variable x used in (a), we assume that x is written by executing transition t_1 and cannot be modified in a parallel structure; for variable y used in (b), we consider three possible situations of writing operation on y: (1) y is written on transition t_1; (2) y is written on transition t_2; (3) y is written on both transitions t_2 and t_4

(1) variable y is written on transition t_1; (2) variable y is written on transition t_2; (3) variable y is written on both transitions t_2 and t_4.

Example 1 In Fig. 1a, if we consider the WF-net without the transition conditions, a number of transition sequences are possible, for example: $\sigma_1 = t_1t_2t_3t_5t_7$, $\sigma_2 = t_1t_2t_4t_6t_8$, $\sigma_3 = t_1t_2t_3t_6$, and $\sigma_4 = t_1t_2t_4t_5$. The last two sequences cannot be extended and they result in the markings $p_3 + p_6$ and $p_4 + p_5$, respectively. The net therefore does not have the 'option to complete' and is thus not sound. If however, abstract condition assignments are taken into account (e.g., abstract condition pred1(x) to t_3, $\neg$pred1(x) to t_4, pred2(x) to t_5, and $\neg$pred2(x) to t_6, where pred1(x) and pred2(x) stand for different abstract conditions but share the same variable x), using the approach proposed in Sidorova *et al.* (2011), a conclusion is drawn: the workflow model is may-sound, and, some data refinements are sound while some data refinements are unsound. Then, for data refinement 1 and data refinement 2 in Fig. 1a, we can say "I don't know. Data refinement 1 and data refinement 2 are possibly sound or possibly unsound depending on the may-sound result of the workflow model with abstract predicates".

Now, depending on the concrete data information, we further analyse the soundness results for data refinement 1 and data refinement 2. For data refinement 1, trace σ_3 is no longer possible because transition t_6 cannot be fired along sequence $t_1t_2t_3$. The reason is that under the historic constraint $x > 50\,000$ accumulated from sequence $t_1t_2t_3$, there is no assignment of value to variable x that makes the transition condition $x \leq 50\,000$ of t_6 satisfiable. Analogously, σ_4 is no longer possible because transition t_5 cannot be fired along sequence $t_1t_2t_4$. The reason is that under the historic constraint $x \leq 50\,000$ accumulated along sequence $t_1t_2t_4$, there is no assignment of value to variable x that makes the transition condition $x > 50\,000$ of t_5 satisfiable. The net is then sound. For data refinement 2, trace σ_3 is no longer possible because transition t_6 cannot be fired along sequence $t_1t_2t_3$. The reason is that under the historic constraint $x > 60\,000$ accumulated along $t_1t_2t_3$, there is no assignment of value to variable x that makes the transition condition $x \leq 50\,000$ of t_6 satisfiable. However, σ_4 is possible because transition t_5 can be fired if assigning value to x around the domain $50\,000 < x \leq 60\,000$. The net therefore does not have the 'option to complete' and is thus not sound.

The sample of workflow net with data in Fig. 1a reveals that if concrete transition conditions are taken into account, a precise result of the soundness of a workflow model can be derived.

Example 2 The example in Fig. 1b further shows that the writing operations and writing order on variables used in the transition condition are also important in determining the routing decision. In general, changing variables by transition has a corresponding business scenario: a task (which corresponds to the notion of transition) may be executed by people or by external applications. In this case we do not know how they may change variables. Thus, we can make the design time assumption that the variables that a transition can write to may take on any value as a result of the transition's execution.

Subsequently, let us consider the WF-net of Fig. 1b without transition conditions. Some possible transition sequences are: $\sigma_1 = t_1t_2t_3t_2t_4t_5$, $\sigma_2 = t_1t_2t_3t_2t_4t_6$, $\sigma_3 = t_1t_2t_4t_5$, and $\sigma_4 = t_1t_2t_4t_6$. This net is sound. Consider the abstract predicate assignments (pred(y) to transitions t_3 and t_5 and $\neg$pred(y) to transitions t_4 and t_6) with three possible situations about writing operation on variable y. Using the approach proposed in Sidorova *et al.* (2011), a same conclusion for the above three situations is: these WFD-nets are may-sound. Thus, this soundness result cannot provide a more accurate soundness result if only the abstract predicates are taken into account. If however, we look further into the influence of the concrete transition conditions (shown in the right of Fig. 1b) and writing operation as well as writing order, the soundness results of these three workflow variants become precise. If concrete conditions with writing y on transition t_1 (situation a) are considered, the traces σ_1 and σ_2 are no longer possible because the transition condition $y \leq 5$ on t_4 is unsatisfiable under the historic constraint $y > 5$ accumulated from $t_1t_2t_3t_2$, and the trace σ_3 is no longer possible because the transition condition $y > 5$ on t_5 is unsatisfiable under the historic constraint $y \leq 5$ accumulated from $t_1t_2t_4$. Then the net has become unsound as it no longer has the 'option to complete' property and the 'no dead tasks' property. If transition t_2 may modify variable y (that is, situation b), trace σ_2 is possible because the writing operation of variable y on transition t_2 removes the influence of

the constraint $y > 5$ derived from transition t_3 on transition t_4. However, traces σ_1 and σ_3 are also no longer possible because the transition condition $y > 5$ on transition t_5 is unsatisfiable under the historic constraint $y \leq 5$ derived from transition t_4, and then the net has become unsound as it no longer has 'no dead tasks'. If however, both transitions t_2 and t_4 may modify variable y (that is, situation c), traces σ_1, σ_2, and σ_3 are possible again. The writing operations of variable y on transitions t_2 and t_4 remove the historic constraint $y > 5$ derived from transition t_3 on transition t_4 for traces σ_1 and σ_2, and the historic constraint $y \leq 5$ derived from transition t_4 on transition t_5 for trace σ_3, and then the net becomes sound.

This sample shows the influence of the preceding writing operation as well as writing order on variables used by both the transition condition of the current transition and the preceding transitions along a sequence on the behavioural semantic of the workflow net with a data perspective.

The two examples above show that a precise soundness result of a workflow net will be derived if data refinement is taken into account.

In this paper, based on the work in Sidorova *et al.* (2011), we give an accurate soundness result for any concrete data refinement in any given soundness result of the WFD-net (e.g., must-sound, may-sound, and not may-sound). Accordingly, we can give a precise soundness result of a data refinement if the WFD-net is may-sound. That is, under the situation of may-sound WFD-net, we can say "Given a set of data refinements, I know which data refinement yields a sound WF-net with data and which data refinement results in an unsound WF-net with data".

First, we formally introduce workflow nets with transition conditions (or WTC-nets for short) and formalise the concrete condition from both a syntactic and a semantic point of view. In doing this, we provide a fundamental for studying workflows where routing may be determined by conditions based on data. We then investigate their automated verification based on the formal definition of WTC-nets. We 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 and the transition condition on the current transition act together to determine whether the current transition is enabled or not. Finally, a more accurate result is produced based on the generated reachability graph.

Formal verification of a workflow net with data refinements has two challenges. The first challenge is that we need a way of interpreting the constraint on enabling of transitions caused by transition conditions as well as writing operations and writing orders on variables. That is, there can be too many or infinite values of variables viable for the satisfiability of the constraint. To address this challenge, we generate the historic constraint by accumulating the transition conditions along a sequence and symbolically interpret the satisfiability of the constraint. That is, each variable used in a constraint is described with a symbolic value, and then the satisfiability of each constraint with symbolic values for variables corresponds to many real values. The second challenge is that a state space explosion exists possibly because there can be at least one loop structure in a workflow net. For this challenge, we propose an approach to handle the state space explosion caused by the cyclic execution. Note that there is a different behavioural feature of the cyclic execution for WTC-net from that of WF-net under the influence of the transition conditions as well as writing operation and writing order in a cyclic execution.

2 Augmenting process models with conditions

In this section, the notion of conditional paths in process models is defined formally from both a syntactic and a semantic point of view. To make the paper self-contained, some well-known definitions are first introduced.

2.1 Background

In this paper, we build on workflow nets which are a subclass of Petri nets. van der Aalst (1998b) 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 a finite set of transitions such that $P \cap T = \varnothing$, $P \cup T \neq \varnothing$, and $F \subseteq (P \times T) \cup (T \times P)$ 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 that are input of x, i.e., $\bullet x = \{y|(y,x) \in F\}$, and $x\bullet$ for its post-set—the set of nodes that 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 a 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 list only 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 as $M[t>$, 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 $\sigma = t_1 t_2 \cdots t_n$ 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., $M \xrightarrow{t_1} M_1 \cdots M_{n-1} \xrightarrow{t_n} M'$, and 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 this 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 end point, van der Aalst (1998a) defined workflow nets by imposing certain syntactic restrictions on Petri nets.

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, $o\bullet = \varnothing$, such that every node $x \in P \cup T$ is on a path from i to o.

The initial marking for a WF-net 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 elsewhere; i.e., it formally corresponds to o. As can be seen from Definition 7, WF-nets are concerned solely 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 (modelling 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 (WFD-net) is defined formally.

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

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

Note that no concrete transition guard is defined in WFD-net.

2.2 Workflow nets with transition conditions

In this subsection the notion of a 'workflow net with transition conditions', or WTC-net for short, is introduced and formally defined.

Central to WTC-nets is the notion of a 'condition'. As the definitions of the notion condition from both a syntactic and a semantic point of view are rather lengthy, we have moved them to the Appendix. Here we describe only 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 (e.g., le, leq, ge, geq, eq), arithmetic operators (e.g., plus, minus, times, frac, mod, div), and Boolean operators (e.g., 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. The notion var is used as the set of all variables and the notion VAR as the set of all possible assignments of types to variables. To be able to focus on conditions that are well-formed (e.g., operations contained in conditions are applied to constants or variables of the correct type), herein, 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. For all formal definitions please refer to the Appendix.

A WF-net with transition conditions is a WF-net where the transitions are assigned conditions. The intended meaning is that a transition cannot fire if its condition does not evaluate to true. Transitions may change the values of variables, so we need to know what the variables are that a transition may write to. In workflow management, tasks (which correspond to the notion of transition) may be executed by people or by external applications and thus we do not know how they may change variables. Consequently, we have to work on the assumption that the variables that a transition can write to, may take on any value as a result of its execution. Therefore, in the definition of a WTC-net we record only which transitions can write to which variables, but there is no restriction on how transitions can modify these variables.

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

1. $\mathcal{P} = (P, T, F)$ is a WF-net;

2. $V \subseteq$ var is a set of variables used in transition conditions;

3. $C \subseteq$ cond is a set of transition conditions with $\cup_{c \in C} V_{\text{var}}(c) \subseteq V$ and for each $c \in C$, $V_{\text{wfc}}(c, \text{VT})$ holds;

4. G: $T \to C$ is a function assigning conditions to transitions;

5. W: $T \to 2^V$ is a function that yields the variables that transitions can write to;

6. VT $\in$ VAR is a function assigning a type to each variable.

If the type assignment VT is clear from the context, we will simply omit it.

2.3 Behaviour of WTC-nets

In this subsection, we introduce a semantics for WTC-nets considering transition conditions and the preceding writing operation as well as writing order on variables used in transition conditions.

In a marking of a WTC-net we should not only record the tokens in each of the places, but also maintain the transition sequence that leads to the current distribution of tokens. The idea is that on the basis of this transition history, we can determine whether a conflict exists between the condition of the current transition and the preceding transition conditions along the sequence.

Definition 10 (Marking of WTC-net) For a WTC-net $\mathcal{W}$= $(\mathcal{P}, V, C, G, W, \text{VT})$, 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.

Note that the initial marking of this net $\mathcal{M}_0$ is defined as (i, ε), where i is the unique source place of $\mathcal{W}$. And there are a set of final markings where $M = o$ is the unique sink place of $\mathcal{W}$.

To evaluate whether in the context of a transition sequence σ a given transition condition ϕ is held, we need to consider the variables that are used in the conditions of transitions in both σ and ϕ. In addition, we need to consider the order of the writing operations on these variables by transitions in σ. If, for example, $\sigma = t_1 t_2$ and the transition condition associated with t_1 is $p = 1$ and with ϕ is $p = 2$, then we have to assume that ϕ cannot hold if p is not in $W(t_2)$, i.e., if it is not changed by t_2. If, however, p can be changed by transition t_2, it is to be treated as if it were a different variable when evaluating condition ϕ, and it is conceivable that ϕ holds.

To solve this problem, we introduce an operation on transition sequences that introduces fresh variables as replacements for variables used in transition conditions whenever these variables could have received a new value through the execution of preceding transitions. The concept of the 'fresh variable' is inspired from the concept of static single assignment (SSA) form (Cytron *et al.*, 1991). That is, by introducing a new special variable, every variable is assigned a value at most once. The difference is that the fresh variable is used as a dynamic replacement for variable whenever the variable has received a new

value during the running time. However, the SSA is used as a static replacement for variable when the variable is defined each time in the control flow graph (CFG) of a program. A fresh variable in a transition sequence σ takes the form v_t^n. Herein, the parameter t is a transition which could write v, i.e., $v \in W(t)$, and the parameter n is a natural number, which represents the number of previous occurrences of t in σ (this is required to be able to deal with loops). We assume that a variable of the form v_t^n does not occur as a net variable in a WTC-net, so that it will be fresh indeed whenever introduced. We also assume that given a renaming function $\psi : \text{var} \rightarrowtail \text{var}$ and a condition $c \in \text{cond}$, we can transform this condition by replacing occurrences of variables with their counterparts as captured in ψ. We will write $c[\psi]$ to denote condition c where for every variable $v \in V_{\text{var}}(c) \cap \text{dom}(\psi)$ each of its occurrences has been replaced by $\psi(v)$.

Definition 11 (Conjunctive form of transition conditions along the sequence) Let $\mathcal{W}=(\mathcal{P}, V, C, G, W, \text{VT})$ be a WTC-net, $\sigma \in T^*$ a transition sequence (possibly empty, i.e., $\sigma = \varepsilon$), $\psi : \text{var} \rightarrowtail \text{var}$ a renaming of variables, $G(t)[\psi]$ a changing transition condition according to this renaming, $n : T \rightarrow \mathbb{N}$ an assignment of natural numbers to transitions, and b any Boolean expression. The conjunctive form of transition conditions along the sequence with variables renaming $\rho^{\psi,\ n}(\sigma,\ b)$ is defined by

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

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

Before we can formally define whether a transition is enabled, we need to be able to determine whether its transition condition is satisfied in the context of a particular assignment of values to variables. We will refer to such an assignment as an environment 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., $\text{Val} = \mathbb{Z} \cup \mathbb{R} \cup \mathbb{B}$.

Definition 12 (Satisfiability of conditions) Let $\text{VT} \in \text{VAR}$ and let c be a well-formed condition in the context of type assignment VT, i.e., $c \in \text{cond}$ such that $V_{\text{wfc}}(c, \text{VT})$, $w : \text{var} \rightarrow \text{Val}$ an assignment of values to variables, and the function $V_{\text{holds}} : \text{cond} \times \text{ENV} \times \text{VAR} \rightarrow \text{bool}$ is applied to determine whether in the context of a given environment a condition holds.

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

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

Example 3 For the WTC-net sample in Fig. 1b, assume the variable y is modified by transitions t_2 and t_4 respectively. Given a current marking $(p_3, t_1t_2t_3t_2t_4)$, let us consider that transition t_5 can be enabled or not. Firstly, we observe how to derive the conjunctive form of transition conditions for transition t_5 along the sequence $t_1t_2t_3t_2t_4t_5$. As shown in the first column of Table 1, variable y is written twice by transition t_2 and once by transition t_4. Once each writing operation occurs the corresponding variable is mapped into a unique fresh variable, as shown in the second column of Table 1. For example, variable y is mapped into $y_{t_2}^1$, $y_{t_2}^2$, $y_{t_4}^1$ along the sequence $t_1t_2t_3t_2t_4t_5$. The corresponding transition condition is then 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 $y_{t_2}^1$ following the closest preceding variable renaming $y_{t_2}^1$. And the transition condition of transition t_4 in the sequence is changed as $\neg y_{t_2}^2$ following the closest preceding variable renaming $y_{t_2}^2$. 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 $(y_{t_2}^1 > 5) \wedge (y_{t_2}^2 \leq 5) \wedge (y_{t_4}^1 > 5)$, there is at least an assignment of value 6 to $y_{t_2}^1$, value 4 to $y_{t_2}^2$, and value 8 to $y_{t_4}^1$, which makes the conjunctive form

Table 1 Deriving the conjunctive form of transition conditions along sequence $t_1t_2t_3t_2t_4t_5$ for WTC-net with variable y written by transitions t_2 and t_4 in Fig. 1b

TS	VR	TCT	CTC
t_1			
t_2	$y^1_{t_2}$		
t_3		$y^1_{t_2}>5$	$y^1_{t_2}>5$
t_2	$y^2_{t_2}$		$y^1_{t_2}>5$
t_4	$y^1_{t_4}$	$y^2_{t_2}\leq 5$	$(y^1_{t_2}>5)\wedge(y^2_{t_2}\leq 5)$
t_5		$y^1_{t_4}>5$	$(y^1_{t_2}>5)\wedge(y^2_{t_2}\leq 5)\wedge(y^1_{t_4})>5$

TS: transition sequence; VR: variable renaming; TCT: transition condition transformation; CTC: conjunctive transition conditions

hold. Thus, transition t_5 is enabled in the current marking $(p_3, t_1t_2t_3t_2t_4)$. Once transition t_5 is fired, the marking $(o, t_1t_2t_3t_2t_4t_5)$ is reached.

Discussion The state space of the WTC-net is explored in terms of symbolic state in the form of (M,σ). Herein, 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. To avoid the state explosion caused by naive exploration of the data dimension, we adopt the symbolic execution technique (Clarke, 1976; King, 1976) that converts the reachability problem to that of solving simple constraint systems (Kumar, 1992; Tsang, 1993; Ganzinger *et al.*, 2004). That is, any vector of values assigned to variables, which makes both the historic constraint system and the transition conditions satisfied, will drive the WTC-net's execution by firing the transition t. For example, for the transition t_5 in the state $(p_3, t_1t_2t_3t_2t_4)$, there are many possible assignments of values to variables $y^1_{t_2}$, $y^2_{t_2}$, and $y^1_{t_4}$ that make the constraints $(y^1_{t_2}>5)\wedge(y^2_{t_2}\leq 5)\wedge(y^1_{t_4}>5)$ satisfiable. Thus, one symbolic execution may correspond to many real executions.

Definition 15 (Reachable marking) Let $\mathcal{W}=(\mathcal{P},V,C,G,W,\mathrm{VT})$ be a WTC-net. A marking (M,σ) with $\sigma{=}t_1t_2\cdots t_n$ is reachable iff a sequence of reachable markings $M_0M_1\cdots M_n$ exists such that $(i,\varepsilon)\xrightarrow{t_1}(M_1,t_1)\xrightarrow{t_2}(M_2,t_1t_2)\xrightarrow{t_3}\cdots\xrightarrow{t_{n-1}}(M_{n-1},t_1t_2\cdots t_{n-1})\xrightarrow{t_n}(M_n,t_1t_2\cdots t_n)$ with $M_n=M$.

Based on Definition 15, we now define the soundness property of a WTC-net. van der Aalst and van Hee (2002) introduced a notion of soundness for WF-nets. This notion aims to capture desirable properties of a net in a workflow context. Introducing soundness for WTC-nets requires only minimal changes. We limit ourselves to the definition presented in Verbeek (2004) where the 'option to complete' requirement and the 'proper completion' requirement are truly orthogonal and this definition follows the observation in van Hee *et al.* (2004).

Definition 16 (Soundness, adapted from Verbeek (2004)) Let $\mathcal{W}=(\mathcal{P},V,C,G,W,\mathrm{VT})$ be a WTC-net. WTC-net $\mathcal{W}$ is sound iff:

1. Option to complete: for every reachable marking (M,σ), there is a marking (M',σ') such that $(M,\sigma)\xrightarrow{*}(M',\sigma')$ and $M'\geq o$.

2. Proper completion: for every reachable marking (M,σ) with $M\geq o$, we have $M=o$.

3. No dead tasks: for every task $t\in T$ there is a reachable marking (M,σ) such that t is enabled in (M,σ), i.e., $(M,\sigma)[t\rangle$.

3 Reduction for cyclic execution

The soundness verification, which is based on the reachability set of the WTC-net, is a reachability analysis method. The major problem in applying reachability analysis is the potential explosion of the state space. To attack this problem, we propose different approaches to compact the state space. First, we extend reduction rules proposed in Wang *et al.* (2009) to generate a canonical WTC-net from the original WTC-net. The basic idea is to reduce the complexity of the initial WTC-net by removing the redundant tasks from the WTC-net. Here, redundant tasks are defined as tasks having neither transition conditions nor writing operations. A canonical net is then generated. This net contains the reduced underlying WF-net with pure transition conditions as well as writing operations on the variables used by transition conditions. These reduction rules were described in detail in Wang *et al.* (2009). Second, we propose a solution to terminate repeated cyclic executions and avoid exhaustive state space explosion caused by repeated cyclic executions. In this section, the solution of the state space explosion caused by repeated cyclic executions in a WTC-net system is considered.

In general, cycles possibly exist in a reachability graph of a Petri net system. Every cycle in a

reachability graph can be executed from a start state and reproduce that start state. Thus, the termination of repeated executions of a cycle in the reachability graph can be determined by observing that the same marking occurs twice along a firing sequence σ. For example, as shown in Fig. 2a, a cycle t_2t_3 is executed from marking p_1 to marking p_1 in the reachability graph of the underlying WF-net of Fig. 1b. Then, the repeated executions of the cycle t_2t_3 are terminated when the start state p_1 of the cyclic execution is reproduced along the sequence $t_1t_2t_3$.

However, we cannot directly terminate the repeated cyclic executions of a WTC-net system by observing that the same underlying WF-net marking occurs twice along a firing sequence in the reachable set of a WTC-net system. For a WF-net system, conditional routings, i.e., conditions of branchings or loops, are often neglected and are to be considered only as non-deterministic and fair. However, for a WTC-net system, conditional routings can be determined by looking at transition conditions based on data (That is, the manipulation of data variables through write operations, the order of these operations, and use of the data variables in transition conditions). Thus, WTC-net variants with cyclic execution, all having the same underlying WF-net, may exhibit different behaviours. For example, Figs. 2b and 2c illustrate the different behaviours of the execution of cycle t_2t_3 for different WTC-net variants with the same underlying WF-net in Fig. 1b. Accordingly, the termination of repeated cyclic executions cannot be determined by observing that the same underlying WF-net marking occurs twice along a firing sequence in the reachable set of a WTC-net system.

For a WTC-net system, the local constraint inside a cycle governs the latter rounds of repeated cyclic executions excluding the first round. It then makes a possible behavioural difference for the second and the latter rounds of repeated cyclic executions from the first round. Conversely, the local constraint also makes a behavioural duplication of the former cyclic execution for the latter cyclic execution from the third round. For the behavioural difference aspect, as shown in Fig. 2b, the partial reachability tree of a WTC-net variant illustrates that there are two possible choices at the marking (p_2, t_1t_2) during the first round of the execution of the cycle t_2t_3: firing the transition t_3 to continue the rest of the first round of cyclic execution, or shifting to fire the transition t_4 thereby quitting the cyclic execution. There is only one possibility at the marking $(p_2, t_1t_2t_3t_2)$ during the second round of cyclic execution t_2t_3: firing the transition t_3 to continue the rest of the second round of cyclic execution. For the behavioural duplication aspect, as shown in Figs. 2b

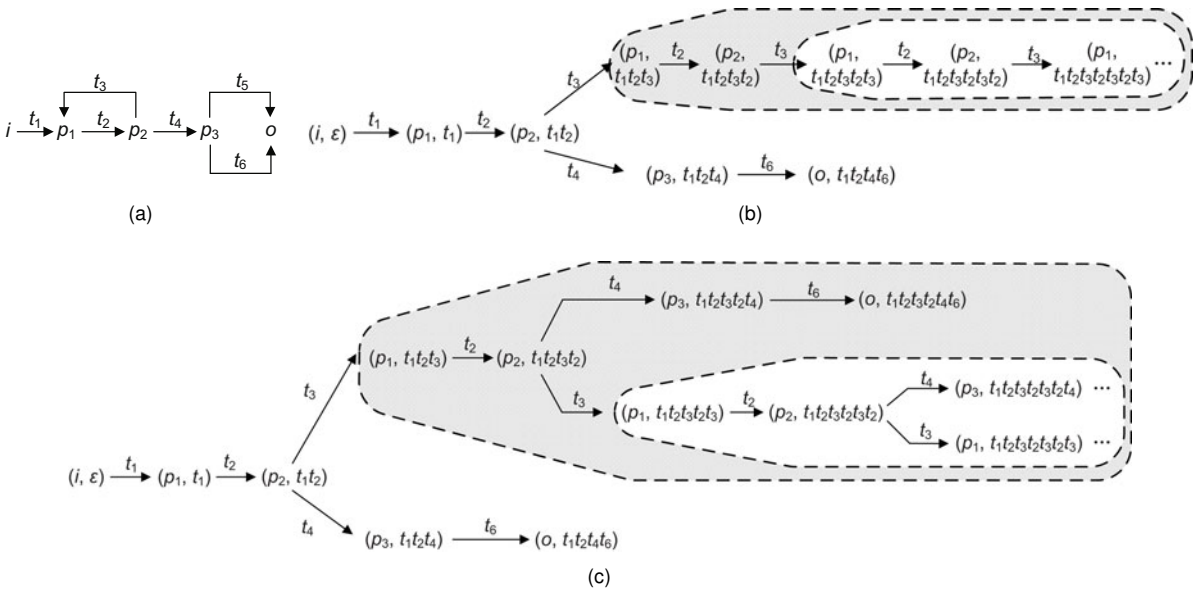


Fig. 2 Partial reachability sets for the WF-net model and the WTC-net variants in Fig. 1b. (a) is the reachability graph of the WF-net in Fig. 1b; (b) and (c) are the partial reachability trees of the WTC-net with variable y being written by transitions t_1 and t_4 in Fig. 1b, respectively

and 2c respectively, the inner marked area closed with a dotted line (the third round of the execution of cycle t_2t_3) is a behavioural duplication of the outer marked area closed with a dotted line (the second round of the cyclic execution). Upon further analysis, as shown in Fig. 2c, for the WTC-net variant of Fig. 1b with y written by transition t_2, a new constraint $y^1_{t_2} > 5$ is accumulated in the start state $(p_1, t_1t_2t_3)$ of the second round of the cyclic execution after the first cyclic execution. In addition, a new constraint $y^2_{t_2} > 5$ is accumulated in the start state $(p_1, t_1t_2t_3t_2t_3)$ of the third round cyclic execution after the second cyclic execution. Specifically, both $y^1_{t_2}$ and $y^2_{t_2}$ are fresh variables of the original variable y. Accordingly, the dynamic constraints $y^1_{t_2} > 5$ and $y^2_{t_2} > 5$ have the corresponding common cycle constraint set $\{(t_3, y > 5)\}$ (that is, $y > 5$ on transition t_3) by traversing the cycle t_2t_3. Thus, looking at the start state of the underlying WF-net of a cyclic execution with a specific cycle constraint set, we can identify the start state of the third round and that of the second round of repeated cyclic executions. Then repeated cyclic executions can be terminated in the start state of the third round.

Subsequently, we consider how to identify the start state of the underlying WF-net of a cyclic execution in a WTC-net system. As the first step, we need to define cycles.

Definition 17 (Cycle) Given a WTC-net system $\mathcal{S}$= ($\mathcal{W}$,$\mathcal{M}_0$) where $\mathcal{M}_0$= (i, ε) is the initial marking, let s be a firing sequence of WTC-net system $\mathcal{S}$. Sequence s from marking (M, σ) to marking $(M', \sigma.s)$ is a cycle of the WTC-net system $\mathcal{S}$ iff $M = M'$.

It is well known that transition invariants (that is, T-invariants) give hints to cycles in a Petri net system (Schmidt, 2003). Every transition sequence, which can occur in some state and which returns to the same state, corresponds to a T-invariant. That is, for a cycle σ, there is always a corresponding unique T-invariant x where $\sharp_t(\sigma) = x(t)$ (Here, $\sharp_t(\sigma)$ is the number of occurrences of t in σ). Specifically, a cycle is an elementary cycle if and only if there exists a corresponding minimal T-invariant. T-invariants and minimal T-invariants have been described in more detail in Murata (1989).

In Fig. 3, the underlying WF-net has a cycle t_3t_4 from marking $p_2 + p_5$ to $p_2 + p_5$. The corresponding minimal T-invariant of this cycle is $t_3 + t_4$. The cycle $t_2t_3t_5t_6$ from marking $p_1 + p_5$ to $p_1 + p_5$ has a corresponding minimal T-invariant $t_2 + t_3 + t_5 + t_6$. The cycle $t_2t_3t_4t_3t_5t_6$ from marking $p_1 + p_6$ to $p_1 + p_6$ has a corresponding T-invariant $t_2 + 2t_3 + t_4 + t_5 + t_6$. And as a last example the cycle $t_3t_5t_6t_2t_3t_4$ from marking $p_2 + p_6$ to $p_2 + p_6$ has the corresponding T-invariant $t_2 + 2t_3 + t_4 + t_5 + t_6$. Therein, the cycles t_3t_4 and $t_2t_3t_5t_6$ are elementary cycles because there exist the corresponding minimal T-invariants $t_3 + t_4$ and $t_2 + t_3 + t_5 + t_6$, respectively. The cycles $t_2t_3t_4t_3t_5t_6$ and $t_3t_5t_6t_2t_3t_4$ have the same T-invariant $t_2+2t_3+t_4+t_5+t_6$, which is a combination of minimal T-invariants $t_3 + t_4$ and $t_2 + t_3 + t_5 + t_6$. The difference between the cycles $t_2t_3t_4t_3t_5t_6$ and $t_3t_5t_6t_2t_3t_4$ is that the start state of a possible execution of the cycle $t_2t_3t_4t_3t_5t_6$ contains place p_1 while the start state of a possible execution of the cycle $t_3t_5t_6t_2t_3t_4$ contains place p_2.

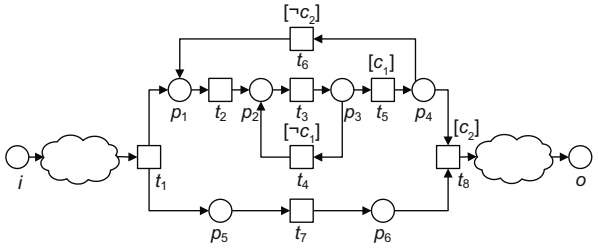


Fig. 3 A sample of WTC-net with loop structures

Next, we shall use the statical structure of a WTC-net to identify potential candidates of the start states of the underlying WF-net for repeated cyclic executions.

Definition 18 (Elementary loop and entry point) Let x be a minimal T-invariant of the underlying WF-net for a WTC-net $\mathcal{W}$. A subset $\mathcal{L}$ of $\mathcal{W}$ (T', P', F') is an elementary loop of the underlying WF-net if and only if $T' = \{t|x(t) > 0\}$, $P' = (\cup_{t\in T'} \bullet t) \bigcup (\cup_{t\in T'} t\bullet)$, and $F' = (P' \times T') \bigcup (T' \times P')$. A place $p_i \in P'$ is an entry point of $\mathcal{L}$ if and only if there exists at least a transition $t_j \in (T - T')$ (outside of the loop) and an arc $(t_j, p_i) \in (F - F')$. Thus, the entry point set $\mathcal{E}$ of a WTC-net $\mathcal{W}$ is a set of entry point sets of all elementary loops.

For example, as shown in Fig. 3, there is a corresponding minimal T-invariant $t_3 + t_4$ for the cycle t_3t_4. Accordingly, there is a static loop $p_2 \to t_3 \to p_3 \to t_4 \to p_2$ with an entry point set $\{p_2\}$ in the underlying WF-net. Analogously, there is a corresponding minimal T-invariant $t_2 + t_3 + t_5 + t_6$ for the cycle $t_2t_3t_5t_6$. Accordingly, there is a static loop

$p_1 \to t_2 \to p_2 \to t_3 \to p_3 \to t_5 \to p_4 \to t_6 \to p_1$ with entry point set $\{p_1, p_2\}$ in the underlying WF-net. The entry point set $\mathcal{E}$ of the underlying WF-net in Fig. 3 is $\{p_1, p_2\}$.

Now note that according to Definitions 17 and 18, a dynamic cyclic execution must contain a firing sequence which has the corresponding elementary loop or a combination of different elementary loops. As an elementary loop always contains at least one entry point, we can formulate the following proposition:

Proposition 1 Given a WTC-net $\mathcal{W}$ and its set of entry points $\mathcal{E}$, and the fact that every cycle in $\mathcal{W}$ has a corresponding elementary loop or a combination of different elementary loops, a cyclic execution of a WTC-net system contains at least one state (M, σ) where $M(p) > 0$, $p : p \in \mathcal{E}$ is an entry point. Then, (M, σ) is one possible start state for cyclic execution.

The start state of the first round of repeated cyclic executions is denoted as the notion 'start point of repeated cyclic executions'. Now we define how to compute it.

Definition 19 (Start point of repeated cyclic executions) Given a WTC-net $\mathcal{W}$ and its set of entry points $\mathcal{E}$, and letting $\sigma = \sigma_1 \cdot \sigma_2$ be a sequence of transitions such that $(i, \varepsilon)[\sigma_1 > (M, \sigma_1)$ and $(M, \sigma_1)[\sigma_2 > (M, \sigma)$, (M, σ_1) is a start point of repeated cyclic executions along the sequence σ iff: there are a set of reachable markings $\mathcal{S}_M$ along the sequence σ containing WF-net marking M where $\exists p : M(p) > 0 \wedge p \in \mathcal{E}$, and marking (M, σ_1) first appears for all possible markings of $\mathcal{S}_M$ along sequence σ.

Inductively, there are two common features to identify the start state of each round of repeated cyclic execution from the second round in a WTC-net system: one is that the same underlying WF-net marking contains an entry point, and the other is that the start of each cycle has the same cycle constraint set (that is, a set of the transition-condition pairs accumulated by the traversing cycle). Here, we use the term 'anchor label' for the combination of the two features above. Thus, the termination of repeated cyclic executions can be determined by the same anchor label appearing twice along a firing sequence in the reachable set of a WTC-net system.

Next, we formally define the cycle constraint set and anchor label.

Definition 20 (Cycle constraint set and anchor label) Given a WTC-net $\mathcal{W}$ and its set of entry points $\mathcal{E}$, let $\sigma = t_1 t_2 \cdots t_n$ be a sequence of transitions (not all transitions need to be different) from a possible start state of a certain cycle (M', σ') in which t_1 is enabled. For all $1 \le i < n$, firing t_i yields a marking $(M_i, \sigma' t_1 \cdots t_i)$ in which t_{i+1} is enabled, and firing t_n yields marking (M'', σ''), i.e., $(M', \sigma') \xrightarrow{t_1} (M_1, \sigma' t_1) \cdots (M_{n-1}, \sigma' t_1 \cdots t_{n-1}) \xrightarrow{t_n} (M'', \sigma'')$. There is an execution of cycle σ iff $M' = M''$. Then, the corresponding cycle constraint set $\mathcal{C}_l(\sigma)$ is defined by

$$\mathcal{C}_l(\varepsilon) = \psi,$$
$$\mathcal{C}_l{}'(t\sigma) = \mathcal{C}_l(\sigma) \oplus (t, G(t)).$$

Correspondingly, the anchor label $\mathcal{M}_{\updownarrow}$ for the marking (M'', σ'') is $(M'', \mathcal{C}_l(\sigma))$.

In the WTC-net in Fig. 3 there is a possible execution of the cycle $t_2 t_3 t_5 t_6$ from the WF-net marking $p_1 + p_5$ to $p_1 + p_5$; therefore, the corresponding anchor label for the start states of both the second and the third rounds is $(p_1 + p_5, \{(t_5, c_1), (t_6, \neg c_2)\})$. Analogously, since there is a possible execution of the cycle $t_2 t_3 t_4 t_3 t_5 t_6$ from the WF-net marking $p_1 + p_5$ to $p_1 + p_5$, the corresponding anchor label for the start states of both the second and the third rounds is $(p_1 + p_5, \{(t_4, \neg c_1), (t_5, c_1), (t_6, \neg c_2)\})$.

Now, we consider the computation of the termination point (i.e., the start state of the third round cyclic execution) for repeated cyclic executions.

Definition 21 (Termination point) Given a WTC-net $\mathcal{W}$, a transition sequence $\sigma = t_1 t_2 \cdots t_n$ exists such that $(M', \sigma') \xrightarrow{\sigma} (M'', \sigma'')$ and $M' = M''$. (M'', σ'') is a termination point iff $\mathcal{M}_{\updownarrow}(M', \sigma') = \mathcal{M}_{\updownarrow}(M'', \sigma'')$.

For example, the anchor label of the marking $(p_1, t_1 t_2 t_3)$ is $(p_1, \{(t_3, y > 5)\})$ and the anchor label of the marking $(p_1, t_1 t_2 t_3 t_2 t_3)$ is $(p_1, \{(t_3, y > 5)\})$ in Figs. 2b and 2c, respectively. Thus, the marking $(p_1, t_1 t_2 t_3 t_2 t_3)$, which has the same anchor label $(p_1, \{(t_3, y > 5)\})$ compared to the marking $(p_1, t_1 t_2 t_3)$, is the termination point. The repeated cyclic executions can then be terminated. By finding the same anchor label along a reachable path, the final coverability graph can be generated (Figs. 4a and 4b).

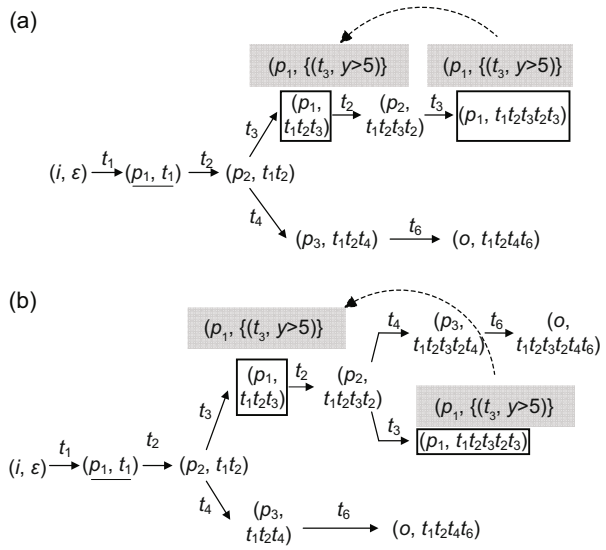


Fig. 4 Coverability graph of the WTC-net in Fig. 1b with variable y written by transition t_1 (a) or t_2 (b). The underlined marking is the start point of repeated cyclic executions, the markings marked by solid rectangles are the start states of the second and third cyclic executions, the extra-labels of the marking annotated with gray rectangles are the corresponding anchor labels of the start states of the second and third cyclic executions. Also, a directed dotted line connects the marking $(p_1, t_1t_2t_3t_2t_3)$ to the marking $(p_1, t_1t_2t_3)$

4 Tool support

We have implemented the proposed approach for determining the correctness of workflow nets with transition conditions. The tool we developed is called the WTC-net soundness checker. The overview of the tool architecture is shown in Fig. 5. We used 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 (The ProM framework can be downloaded from http://www.processmining.org). The tool is composed of two main parts: the WTC-net constructor and the soundness analyser. Amongst these, the soundness analyser part contains two components: (1) a coverability graph generator, and (2) a soundness checker.

4.1 WTC-net constructor

The WTC-net constructor provides the design environment for developing workflow nets with well-formed transition conditions. It includes two main components: the WTC-net editor and the WTC-net checker. The WTC-net checker contains a well-structured WF-net checker and a well-formedness checker for transition conditions.

1. WTC-net editor: The editor supports the creation of a new WTC-net, and importing and exporting of WTC-nets. A WTC-net includes two parts: the underlying WF-net and the data related with transition conditions. The editor supports the creation of a new WTC-net by importing a WF-net model and editing the data related to transition conditions based on the imported WF-net model. The editor also supports importing and exporting WTC-net models.

2. WTC-net checker: The function of this component is twofold: (1) to check whether the transition condition is well-formed (The well-formed condition is defined in the Appendix)—the related implementation follows the functions TypeOf and V_{wfc} defined in the Appendix; (2) to check whether the control flow structure of a WTC-net is well structured. A WF-net is well structured if its short-circuited net is well handled (Verbeek, 2004). The main goal in

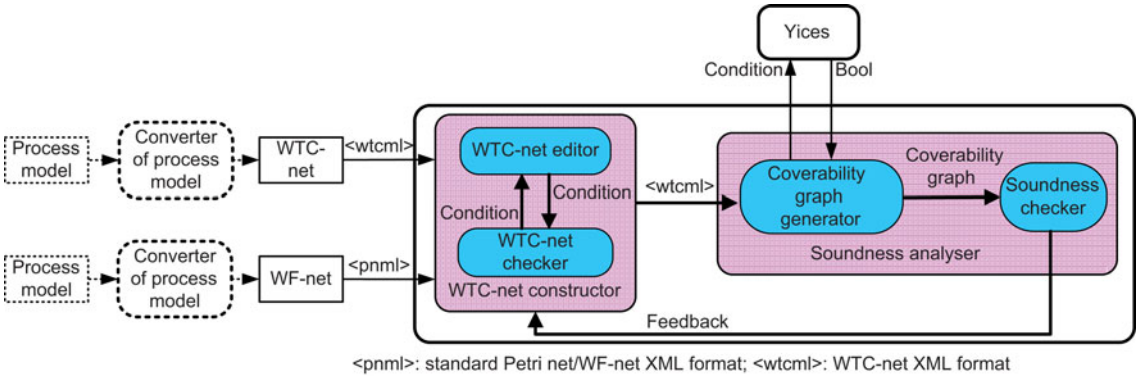


Fig. 5 Architecture of the WTC-net soundness checker

checking the well-structuredness is to provide a preprocessing before soundness checking. Some syntax anomalies from the WF-net structure aspect are excluded at first. As discussed in Verbeek (2004), the TP- and PT-handle will prevent certain transitions from being enabled. These situations are commonly regarded as anomalies from the structural point of view of the WF-net. The well-structuredness checking is implemented by calling Woflan (Verbeek *et al.*, 2001) and the TP/PT handle component integrated in ProM 6.0.

Fig. 6 shows the interface of the WTC-net constructor which describes the WTC-net variant of Fig. 1b with variable y being written on transition t_2.

4.2 Coverability graph generator

The coverability graph generator supports the creation of a coverability graph based on the reachability notion of a WTC-net defined in Section 2. The whole process of generating a coverability graph from a WTC-net is similar to the algorithm proposed in Karp and Miller (1969): first, a Petri net coverability tree is constructed, and then a coverability graph is obtained by merging the identical nodes in the coverability tree.

Since reachability analysis is the core of our approach, we briefly introduce the process of generating the coverability tree of a WTC-net: given a WTC-net $\mathcal{W}$ with $\mathcal{M}_0$, from this initial marking $\mathcal{M}_0$, many new markings can be obtained with the increase in the number of the enabled transitions. From each new marking, more markings can be reached. This process results in a coverability tree. For 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. Specifically, three different cases should be considered when a new node is explored with its successors.

Case 1: Detection of an identical known marking. For current node n, there is a node n_1 that is an ancestor of n and has the same anchor label as n: identify n and n_1, and then tag node n as old and shift to process another new node.

Case 2: Non-boundedness detection. For current node n, there is a node n_1 that is an ancestor of n and $n.M > n_1.M$ (i.e., n is coverable), and then replace $n.M(p)$ by ω for each p such that $n.M(p) > n_1.M(p)$. Finally, tag node n as new again and shift to process another new node.

Case 3: New nodes generation.

As described above, generating a coverability graph with transition conditions is supported by an algorithm proposed by Karp and Miller (1969). This algorithm has extra features: the ability to decide on the enabling of transitions when the satisfiability of transition conditions along a certain sequence of transitions is considered, and the ability to decide on the termination of the repeating cyclic executions.

As determined in Definition 13, the enabling of transition is determined not only by the underlying

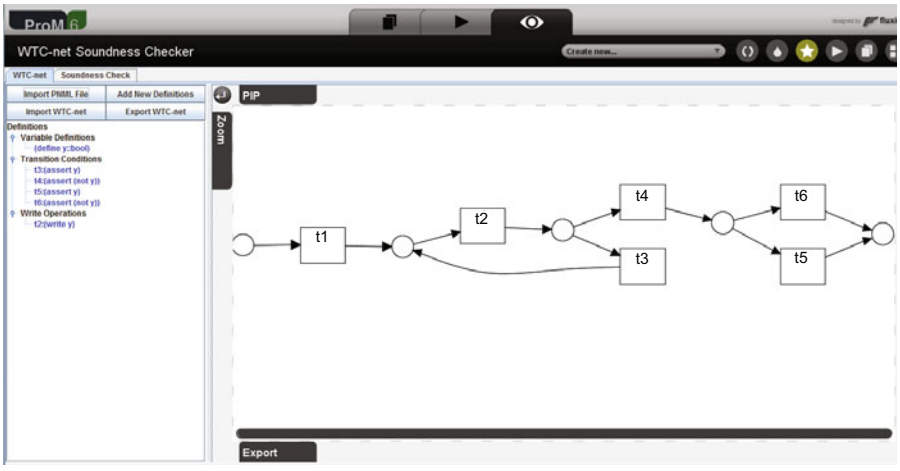


Fig. 6 WTC-net constructor interface. The right-hand side illustrates the underlying WF-net and the left-hand side describes the editor interface for the data (that is, data declaration, transition condition assignment, and writing operations)

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 on the variables used under the current transition condition. The procedures for determining the satisfiability of a transition condition are as follows:

1. Variable extraction. This procedure extracts a (possibly empty) set of data variables from a condition. The implementation follows the function V_{var} defined in the Appendix.

2. Derivation of the conjunctive form of transition conditions with variable renaming along the sequence. This procedure accumulates the transition conditions along a sequence and dynamically renames variables in a certain transition condition along a sequence if there is a writing operation on these variables in a preceding transition. The implementation follows Definition 11.

3. Condition satisfiability check. This procedure follows Definition 12. We developed a Java interface to call the libraries in Yices (Dutertre and de Moura, 2006b). Yices is an satisfiability module theories (SMT) tool, 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 Boolean abstraction of the entire formula. The conjunctive form of transition conditions is regarded as the input parameter to the Yices interface, and then the Yices interface returns the satisfiability result of the conjunctive form (true or false). Thus, if the returned value is true then the constraint is satisfiable and the current transition is enabled, whereas if the returned value is false then the constraint is unsatisfiable and the current transition is not enabled.

The procedures for determining the termination of repeated cyclic executions (if there are loop structures in the WTC-net) are as follows:

1. Entry point set. This procedure calculates the entry point set of all elementary loops in the WTC-net by calling the T-invariant component in ProM 6.0.

2. Termination of repeated cyclic execution. This procedure is to determine the termination of repeated cyclic executions following Definitions 20 and 21 in Section 3.

Fig. 7 shows the coverability graph of the WTC-net variant of Fig. 1b with variable y being written on transition t_2.

4.3 Soundness property checker

The soundness property checker implements the soundness checking algorithm defined in Definition 16. The bottom of Fig. 8 shows the soundness result of the WTC-net variant of Fig. 1b with variable y being written on transition t_2. As expected, the results reveal that the WTC-net model is unsound. The reason is: the historic constraint $y > 5$ accumulated from transition t_4 makes $y \leq 5$ on

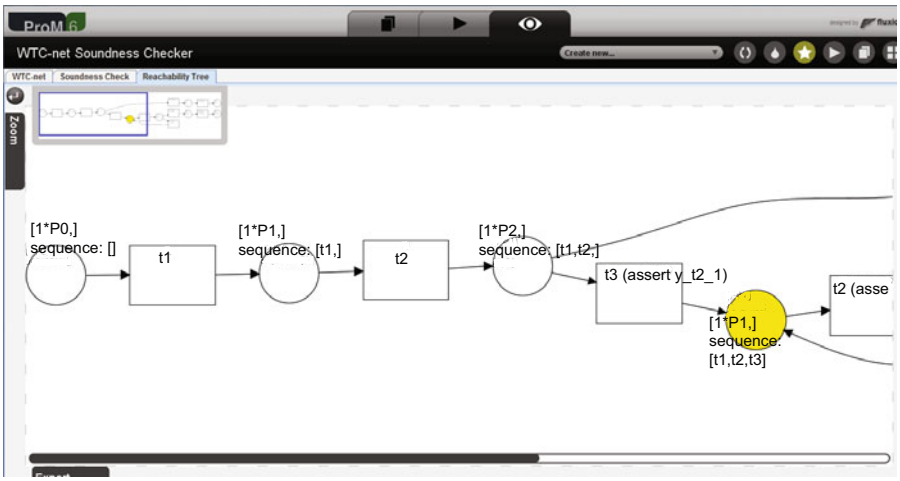


Fig. 7 The coverability graph of the WTC-net variant in Fig. 1b with variable y being written on transition t_2

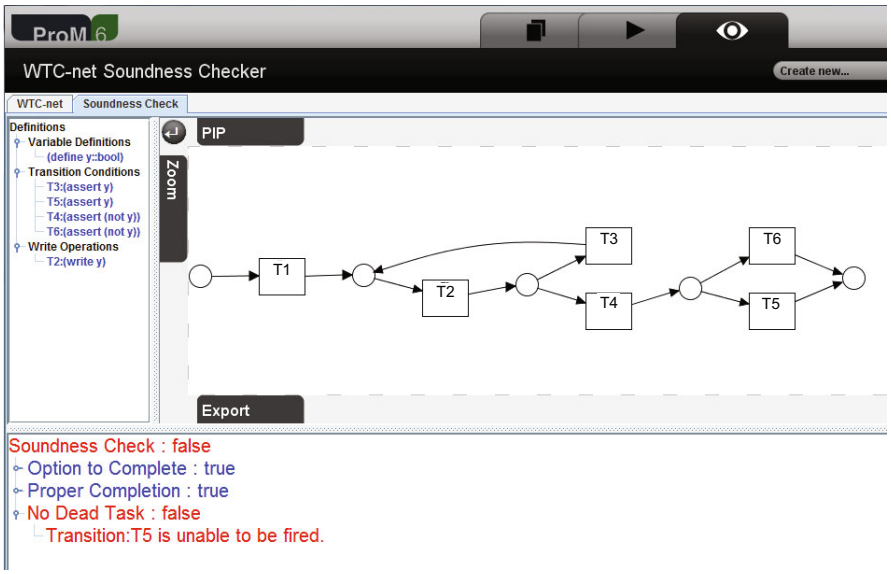


Fig. 8 The soundness result of the WTC-net variant in Fig. 1b with variable y being written on transition t_2

transition t_5 unsatisfiable, resulting in transition t_5 not being enabled. Thus, transition t_5 is a dead task.

4.4 Complexity of the coverability graph of the WTC-net system

In this subsection, we discuss the complexity of the coverability graph of a WTC-net since the graph is the core of the soundness analysis approach.

The complexity of the coverability graph of a WTC-net system is determined by two aspects: one is the complexity of the underlying WF-net coverability graph which is the main-bone for the coverability graph of the WTC-net system; the other is the complexity of the satisfiability of the symbolic state which is deduced from the sequence part σ of a WTC-net marking (M, σ).

For the complexity of the coverability graph of the first WF-net aspect, assuming that both the number of places P and the number of transitions T of the WF-net are much smaller than the number of nodes N of the generated finite coverability tree, a constant α allows to integrate the computer influence, and the notation $O(\cdot)$ is used to express the algorithm complexity.

Firstly, we consider independently the complexity of each case of the algorithm before evaluating the complexity of the whole coverability tree. Assume that the coverability tree is finite having size N, and suppose the algorithm suppresses K nodes during its execution.

Case 1: Detection of an identical known marking. In the worst case less than $N+K$ nodes can be considered to be detected if the marking of the current node is equal to that of another one. The complexity can be expressed as

$$\begin{aligned}\alpha \sum_{n\in X} a(n) &= \alpha \cdot \sum_{n=1}^{N+K} (N+K-1)\\ &= \alpha \cdot (N+K) \cdot (N+K-1),\end{aligned}$$

where n represents the index of the currently treated node, X the set of all the tree nodes (of size less than $N+K$), $a(n)$ the set of the previously generated nodes (of size less than $N+K-1$), and α the computer-dependent coefficient. From the previous expression it is easy to conclude that the complexity of case 1 is $O((N+K)^2)$.

Case 2: Non-boundedness detection. This detects if the marking of the node under treatment is greater than the ancestor marking of it. Like in case 1 this complexity can be expressed by $O((N+K)^2)$.

Case 3: New nodes generation. For this last case, the fireable transitions must be detected for each tree marking. The corresponding complexity can be evaluated as

$$\alpha \sum_{n\in X} (t + C(n)),$$

where $C(n)$ corresponds to the nodes generating

complexity from node n, the sum $\sum C(n)$ is the total number of the nodes generated and is therefore less than $N+K$, and t is the complexity associated with all enabled transitions (less than $|T|$). Finally, the complexity can be estimated as $O((N+K)\cdot|T|)$.

Cases 1 and 2 have the same exponential complexity, higher than the complexity of case 3 which is linear. So, it is easy to conclude that the complexity of finite coverability tree construction is quadratic with the number of tree nodes, i.e., $O((N+K)^2)$.

The coverability graph is constructed by identifying and merging identical nodes from the finite coverability tree. This kind of transformation is linear. Thus, the complexity of graph transformation can be neglected when compared to the complexity of tree construction. Finally, the complexity of the coverability graph of the WF-net is determined as $O((N+K)^2)$.

For the complexity of the satisfiability of the symbolic state, it is well known that the satisfiability problem of the linear constraint is NP-complete (The transition conditions in this paper are restricted to linear formulas). However, the relationship between NP-completeness and problem difficulty has become ever more fuzzy over the last decade, particularly, but not exclusively, for the satisfiability problem (SAT) (Franco, 2005). An important element in these advances has been an improving understanding of structural properties of instances of SAT and their relationship to complexity. To some extent it has come from surprising sources such as statistical mechanics and probabilistic analysis, e.g., the DPLL algorithm (Dutertre and de Moura, 2006a). Thus, although SAT solving is a quintessential NP-complete problem, now it is amazingly fast in such practice as bounded model checking, planning and software verification. In the bounded model checking area, for example, SAT solving is used widely and can handle 10^{300} states (Rushby, 2006).

Finally, we can conclude that the complexity of the coverability graph of a WTC-net system is NP-complete. It is determined mainly by the complexity of the satisfiability of the symbolic state. In view of the wide application of SMT technology in many areas, the feasibility of the approach decided mainly by the complexity of the satisfiability problem will be evaluated by assessing the running time of some real process models and artificial examples in the following section.

5 Experiments

In this section, firstly we illustrate the applicability of our approach to a set of 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. TiPLM is a product life-cycle management solution, developed by the THsoft InfoTech company (http://www.thit.com.cn) and widely used in mainland China (Well over 100 companies in the manufacturing industry in China are using the TiPLM system). To sustain development and research, THsoft InfoTech maintains domain-specific process model templates and also possesses a collection of deployed business process models. Finally, we evaluate the feasibility of our approach by assessing the running time of some real examples and artificial examples. All examples for experiments can be downloaded from http://laudms.thss.tsinghua.edu.cn/trac/Test/attachment/wiki/chengguo/Test%20Data.zip.

5.1 Process model in TiPLM workflow

Here, we provide a brief introduction of TiPLM workflow which is the workflow management software component of TiPLM. In TiPLM workflow, a process model is described by a directed graph that has four different kinds of nodes. Fig. 9 illustrates the different components in the TiPLM workflow model using a real process engineering and changing model (abbreviated as the PC-1 model):

1. Activity nodes represent tasks that are undertaken by some actors;

2. Route nodes represent decision points that determine the execution flow;

3. The start node denotes the start of a workflow;

4. The end node denotes the end of a workflow.

Nodes are connected by directed links to define different kinds of control flows. There are two kinds of links in TiPLM workflow: normal link and false link. Normal links represent normal execution flows, while false links represent those execution flows when routing conditions are evaluated to be false.

The data information setting is performed in building time. There are two stages: the definition stage of process variables and the expression setting stage. Firstly, the definition stage of process vari-

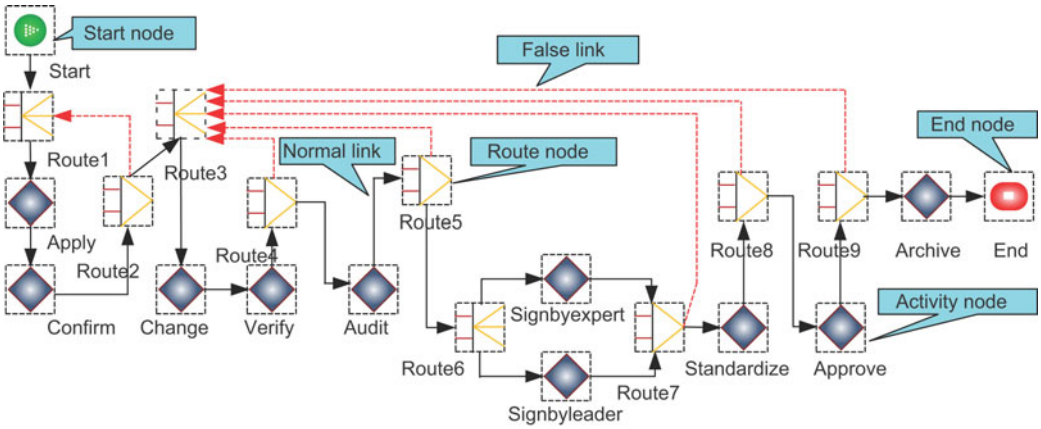


Fig. 9 The original TiPLM workflow model of the WTC-net model PC-1 with six loop structures. To make it discernable, we repaint the false links using dotted lines; however, in the real workflow model false links are represented by solid lines

ables is performed when the process properties are defined. At this stage, the workflow designer specifies the process variables that will be used in this model (including variable declaration and variable initialization). As shown in Fig. 10, the top screenshot illustrates all defined variables in this model, and the bottom screenshot illustrates an interface for a concrete variable definition. Next, the expression setting stage is performed when node properties are defined. The workflow designer specifies routing conditions (that is, relation or Boolean expressions) on the corresponding route node. As shown in Fig. 11, the top screenshot illustrates that the type selecting of a route node is determined by specifying the relationship among input-nodes and that among output-nodes. If the relationship is set as conditional routing (e.g., XOR split construct), the concrete expression is defined as shown in the bottom screenshot of Fig. 11.

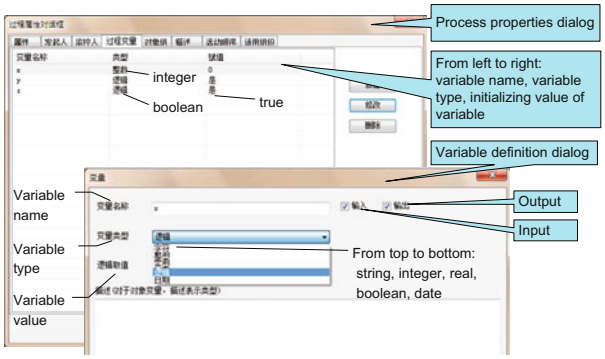


Fig. 10 Screenshots of process variables definition

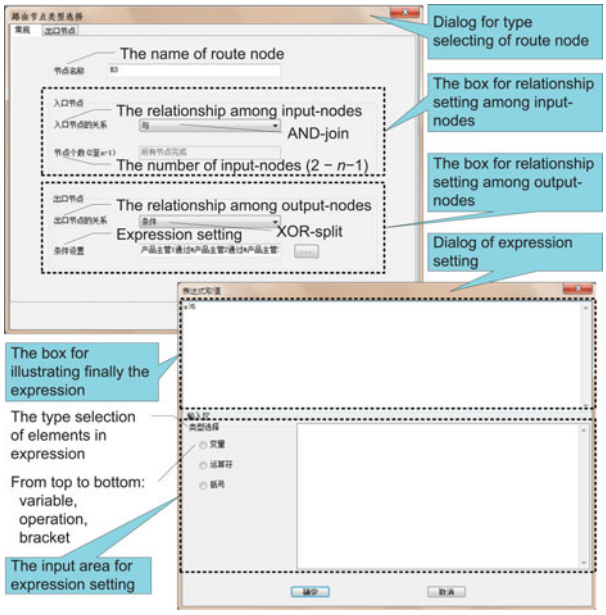


Fig. 11 Screenshots of expression setting

5.2 Data collection

We collected 29 business process models and data-based routing information from the THsoft InfoTech company. These models were created using TiPLM workflow for the following four process modules: engineering design and review, process engineering and changing, release management, and application management. Table 2 shows the distribution of these process models over these modules.

In terms of the number of process models, the largest module is release management, which contains 13 TiPLM workflow models, while the smallest

Table 2 Business process models from the THsoft InfoTech company

Module	#P	#A	#XORP	#XOR	#LP	#L
Engineering design and review (DR)	11	71	11	42	11	35
Process engineering and changing (PC)	1	10	1	6	1	6
Release management (RM)	13	140	11	14	11	14
Application management (AM)	4	20	4	12	4	10

#P: number of process models; #A: number of activities; #XORP: number of process models with XOR splits; #XOR: number of XOR splits; #LP: number of process models with loops; #L: number of loops

module is process engineering and changing, which contains only one TiPLM workflow model. In these TiPLM workflow models, the total number of activities is 241, and there are 27 Tiworkflow models containing XOR split constructs and loop structures, 93% of the total number of these models. Furthermore, the total number of XOR split constructs is 74 and the total number of loop structures is 65. The above statistics show that there is a relatively high use of conditional routing and that the executions of many activities are significantly determined by conditional routings.

As the first step, these 29 TiPLM process models were mapped to WF-net models using the Tiworkflow converter. We have developed this converter based on the algorithm proposed in Zha *et al.* (2011). This converter has also been integrated into the TiPLM system. In addition, we developed an information extractor to extract the data-based routing information from the TiPLM database (Note that the variable in string type used in expression was handled as a variable in enumeration type, which can be regarded as a special integer type with a finite domain, and the variable in date type used in expression was handled as a variable in integer type). This information was then incorporated with the translated WF-net and produced corresponding WTC-nets. We then anonymised these models. Thus, the total number of WTC-nets from THsoft InfoTech used in experiments is 29.

5.3 Tool application

We used the WTC-net soundness checker that was implemented and integrated in the ProM framework to check the soundness property of the collected 29 WTC-net models. The well-structured WF-net checking and well-formed condition checking were executed when the WTC-net model was imported into the WTC-net editor. Fortunately, there were no models that did not satisfy the well-structured WF-net requirements. Hence, there were 29 such models used as input models for further experiments.

5.4 Soundness validation

We have conducted experiments of soundness analysis on 29 real-life process models. The analysis results of these 29 models are shown in Table 3. We compared verification results with other measures including manual analysis. It turned out that our approach was always able to correctly assess the correctness of process definition with a concrete data refinement.

Table 3 Soundness checking result for transformed WTC-net models

Property	#Passed	#Failed	Error ratio
Soundness property	23	6	20.7%
Option to complete	26	3	10.3%
Proper completion	29	0	0%
No dead tasks	23	6	20.7%

These soundness checking results were as follows: there were 6 (20.7%) unsound WF-nets among 29 translated WTC-nets (Table 3). Specifically, 3 (10.3%) WTC-nets did not satisfy the 'option to complete' rule, 0 (0%) WTC-nets did not satisfy the 'proper completion' rule, and 6 (20.7%) WTC-nets did not satisfy the 'no dead tasks' rule.

For WTC-nets that violate the 'option to complete' rule, it is possible to identify the markings that cannot arrive at a kind of marking in which the underlying WF-net marking contains the sink place. Based on this information, we can identify tasks which can cause problems at certain markings. After careful examination, we found a common cause for the violation of the 'option to complete' rule. That is, there is a correlation between two transition conditions assigned to different transitions. Correlation means: (1) two transition conditions, which are assigned to different transitions along a sequence, contain common variables; (2) there are not

writing operations on the common variables between the two transitions along the sequence. Once the latter transition along the sequence is in a cyclic execution, there is possible live-lock and WTC-nets cannot exit from the live-lock. This is a very valuable finding that could not be detected by analysing a process specification from control flow structure alone. One possible remedy for such an error is to ensure that there is a writing operation on common variables between two transitions along a sequence; thus, there will be a possible exit point during cyclic execution.

For WTC-nets that violate the 'no dead task' rule, it is possible to identify the dead task. Based on this information, we can identify that there is a situation causing this error: it is possible that the historic constraint is not satisfiable and then the transition cannot fire. Further, we can probe that if there are two sequential choice structures and a correlation relationship exists among transition conditions assigned separately to these two choice structures, then the satisfiability result of the former transition condition may make the latter transition condition unsatisfiable. Thus, the error of violating 'no dead task' possibly occurs.

For a WTC-net of its underlying WF-net with PT or TP handle, it is possibly easy to violate the 'option to complete', the 'proper completion', and the 'no dead tasks' rules. Thus, currently the implemented tool focuses mainly on the soundness checking of WTC-net with well-structured WF-net. The above consideration is part of the reason why there is no WTC-net model violating the 'proper completion' rule.

The results from soundness checking aim at providing a better insight into the compatibility between process models and condition routings. We report the soundness result of the WTC-net and that of the underlying WF-net to the users. Based on the above decision-supporting information, users can then determine further whether the designed process model is reasonable or not. To ensure that our suggested remedy is in compliance with the business semantics of a process, we communicated our findings to domain experts. Based on subsequent discussions with these domain experts, we corrected these errors and performed soundness checking in an iterative manner until all WTC-nets were found to be sound.

During our interactions, the domain experts expressed interest in learning more about our experiments. It is our expectation that when the tool can provide automated support for stronger expression of conditions, the tool can be used by domain experts to validate their process models before the TiPLM system is deployed in an organisation.

5.5 Feasibility assessment

To the best of our knowledge, there is no other verification tool developed specifically for a workflow net with a data perspective. Thus, we cannot assess the feasibility of our approach by comparing the execution time among different tools. Hence, to evaluate the feasibility of our approach, we first tested the time of generating a coverability graph for the transformed 29 WTC-net models from the THsoft InfoTech company. We then investigated and further tested from two aspects—the volume of process models and the number of routing conditions in the process model—to probe the main factor for the time of generating the coverability graph of a WTC-net. The running environment is an Acer laptop with the configuration of 2.67 GHz Intel Core i5-480M CPU and 4.0 GB memory.

5.5.1 Time results of the coverability graph for 29 WTC-net from THsoft InfoTech

For 29 transformed WTC-net models, Table 4 illustrates structure features for each WTC-net model: the column #Tr represents the number of transitions of a WTC-net model, and the column #Pl represents the number of places of a WTC-net. Compared carefully to Table 2, the total number of transitions in Table 4 is not in conformity with the total number of activities in Table 2. The reason is that transitions in a WTC-net are transformed not only from activities but also from route nodes, the start node, and the end node in the original TiPLM workflow model.

Subsequently, we continue to illustrate the experimental result of the execution time in Table 4. For multiple loop structures in these collected WTC-net models there is a common feature that multiple loop structures are nested together. Thus, the column #NL not only represents the number of loop structures but also explains the depth of nested loop structures. Finally, the column #XOR describes the number of all possible conditional routings in a WTC-net. Thus, the number of choice constructs

Table 4 Time taken for the computation of the coverability graphs

Model	#Tr	#Pl	#XOR	#NL	Time (ms)
DR-1	19	21	3	3	297
DR-2	15	13	3	2	266
DR-3	25	19	7	4	4280
DR-4	16	15	3	3	313
DR-5	18	19	3	3	328
DR-6	22	19	5	5	8120
DR-7	23	19	5	4	3795
DR-8	12	11	2	2	235
DR-9	12	11	2	2	219
DR-10	19	16	4	3	663
DR-11	23	19	5	4	766
PC-1	28	24	6	6	27 624
RM-1	21	23	3	3	313
RM-2	24	32	2	2	258
RM-3	18	26	1	1	157
RM-4	16	22	1	1	231
RM-5	19	28	1	1	235
RM-6	17	24	1	1	227
RM-7	14	18	1	1	235
RM-8	16	22	1	1	231
RM-9	15	25	0	0	157
RM-10	21	32	1	1	235
RM-11	3	4	0	0	141
RM-12	25	40	1	1	231
RM-13	21	32	1	1	220
AM-1	14	12	3	3	313
AM-2	11	10	2	2	235
AM-3	16	13	3	2	266
AM-4	21	17	4	3	235

#Tr: number of transitions; #Pl: number of places; #XOR: number of XOR splits; #NL: number of nested loops

can be deduced from the difference between the number of all possible conditional routings and the number of loop structures. For example, for the DR-3 model in Table 4, there are four nested loop structures and three choice constructs.

Table 4 shows that the time improves significantly when the number of nested loop structures increases gradually. For instance, the time for the DR-1 model with three nested loop structures is 297 ms, that for the DR-3 model with four nested loop structures is 4280 ms, that for the DR-6 model with five nested loop structures is 8120 ms, and that for the PC-1 model with six nested loop structures significantly amounts to 27 624 ms.

From the results of times for the transformed 29 WTC-net models, we can conclude that the number of nested loop structures is possibly a critical factor in determining the length of the execution time of our approach.

To further probe the main factor of execution time for generating the coverability graph, we further investigate the influence of the following two aspects—the volume of a process model and the number of routing conditions—on the time.

5.5.2 Investigating result from SAP reference models

Herein, to study the influence of the volume of a process model on the time of the coverability graph for a WTC-net, we have investigated the well-known reference models of SAP, which contain 604 event-driven process chains (EPCs) modelling different business processes supported by the R/3 system (Curran and Keller, 1998). As the first step, we have transformed these EPC models into Petri nets using ProM. This resulted in 591 Petri nets (13 EPC models could not be mapped to Petri nets using ProM). Table 5 shows the statistical features of transformed Petri nets from both SAP reference models and the THsoft InfoTech company.

As shown from columns 2 to 7 in Table 5, the basic features of 591 Petri nets from SAP reference models and those of 29 Petri nets (WF-nets is a subset of Petri nets) from the THsoft InfoTech company are as follows: on one hand, amongst these 591 Petri nets from SAP reference models, the minimal ones contain only one transition and two places while the maximal ones contain 53 transitions and 83 places; the average number of transitions of a process model is 6.79 while the average number of places of a process model is 10.6. On the other hand, amongst these 29 Petri nets from the THsoft InfoTech company, the minimal ones contain only three transitions and four

Table 5 Features of transformed Petri nets from both SAP reference models and the THsoft InfoTech company

Model	$\#\mathrm{Min_T}$	$\#\mathrm{Max_T}$	$\#\mathrm{Ave_T}$	$\#\mathrm{Min_P}$	$\#\mathrm{Max_P}$	$\#\mathrm{Ave_P}$	$\#T_{0-19}$	$\#T_{20-29}$	$\#T_{30-}$
SAP	1	53	6.79	2	65	10.6	570	17	4
THsoft InfoTech	3	28	18.00	4	40	20.0	18	11	0

$\#\mathrm{Min_T}$: minimal number of transitions for a Petri net; $\#\mathrm{Max_T}$: maximal number of transitions for a Petri net; $\#\mathrm{Ave_T}$: average number of transitions for a Petri net; $\#\mathrm{Min_P}$: minimal number of places for a Petri net; $\#\mathrm{Max_P}$: maximal number of places for a Petri net; $\#\mathrm{Ave_P}$: average number of places for a Petri net. $\#T_{0-19}$: number of Petri nets in which the number of transitions is around 0–19; $\#T_{20-29}$: number of Petri nets in which the number of transitions is around 20–29; $\#T_{30-}$: number of Petri nets in which the number of transitions is above 30

places while the maximal ones contain 28 transitions and 40 places; the average number of transitions of a process model is 18 while the average number of places of a process model is 20.

Subsequently, we count the distribution of these models with different scales. As shown from columns 8 to 10 in Table 5, on one hand, amongst these 591 Petri nets from SAP reference models, there are 570 models (96%) for which the number of transitions is around 0–19, 17 models (3%) for which the number of transitions is around 20–29, and 4 models (1%) for which the number of transitions is above 30. On the other hand, amongst these 29 Petri nets from the THsoft InfoTech company, there are 18 models (62%) for which the number of transitions is around 0–19, 11 models (38%) for which the number of transitions is around 20–29, and 0 model (0%) for which the number of transitions is above 30.

The statistical information of model features from SAP reference models and this from the THsoft InfoTech company show that it is common for an industry business model that the number of transitions and the number of places are around two digits.

Furthermore, we have transformed four Petri nets from SAP reference models, in which the number of transitions is above 30, to WF-nets. Subsequently, we annotated conditions on these WF-nets using our WTC-net soundness checker. The time of the coverability graph for the four derived WTC-nets is about 200 ms (Table 6). Thus, we can draw a conclusion temporarily that the volume of a model is not the critical factor affecting the time of generating a coverability graph.

5.5.3 Experimental results of artificial samples

To further probe the relationship between the number of different situations of conditional routing and the length of execution time, we consider four representative situations related to conditional routings: nested loop structures (Fig. 12a), sequential loop structures (Fig. 12b), nested choices (Fig. 12c), and sequential choices (Fig. 12d). 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 nested loop structures, there are 5 artificial WTC-net models with nested loop structures from 2 to 6 respectively. Thus, there are 20 artificial WTC-net models for these four situations.

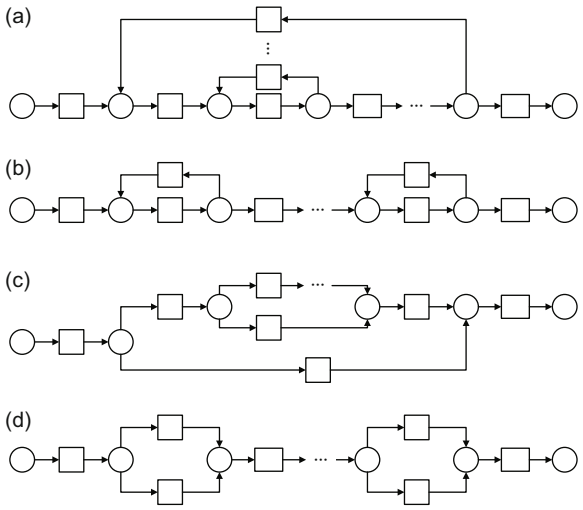


Fig. 12 Construct samples: (a) nested loop structures; (b) sequential loop structures; (c) nested choices structures; (d) sequential choices structures

Table 7 shows the time for different situations. The second row shows that the time soon mounts up when the number of nested loop structures increases gradually. However, the third, fourth, and fifth rows show that the time increases in a stable manner when the number of sequential loop structures, nested choices, or sequential choices increases gradually. The statistical information in Table 7 testifies again that the length of execution time is decided mainly by the number of nested loop structures. Especially when the number of nested loop structures amounts to five or above five, the time increases dramatically. However, there is no important influence on the length of time for other complexity conditional routing situations (e.g., nested choices, sequential choices, and sequential loop structures).

Table 6 Time taken for the computation of coverability graphs for four examples from SAP reference models

$\text{Time}_{\#(1\text{Er\_hhi9})}$ (ms)	$\text{Time}_{\#(1\text{Er\_ixso})}$ (ms)	$\text{Time}_{\#(1\text{Er\_hsto})}$ (ms)	$\text{Time}_{\#(1\text{Un\_jh6h})}$ (ms)
200	221	219	211

$\text{Time}_{\#(1\text{Er\_hhi9})}$ stands for the time taken for the WTC-net named 1Er_hhi9, analogously for $\text{Time}_{\#(1\text{Er\_ixso})}$, etc.

Table 7 Time taken for the computation of coverability graphs for artificial examples

Situation	$\text{Time}_{\#(2)}$ (ms)	$\text{Time}_{\#(3)}$ (ms)	$\text{Time}_{\#(4)}$ (ms)	$\text{Time}_{\#(5)}$ (ms)	$\text{Time}_{\#(6)}$ (ms)
Nested loops	241	292	701	5434	27 624
Sequential loops	231	241	282	331	471
Nested choices	171	171	191	181	171
Sequential choices	161	141	171	271	343

$\text{Time}_{\#(n)}$ stands for the time taken for the WTC-net with n constructs

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

6 Related work

6.1 Soundness verification of control flow perspective

The soundness property, originally defined for a WF-net (van der Aalst, 1997; 1998a), ensures that from any reachable marking the final marking can be reached, and every task can potentially be executed. It is the minimum requirement that every workflow net must meet. This property guarantees the absence of livelocks, deadlocks, and other anomalies that can be detected without domain knowledge.

Based on original WF-nets, variants of WF-nets are proposed to support more control patterns, e.g., models that allow for cancellation, complex joins such as the inclusive OR-join (Wynn *et al.*, 2006). In van Hee *et al.* (2003), two alternative notions of soundness were introduced: k-soundness and generalized soundness. These notions allow for dead parts in the workflow but address problems related to multiple instantiation. The notion of weak soundness was proposed in Martens (2003; 2005), which allows for dead transitions. The notion of relaxed soundness was introduced in Dehnert and van der Aalst (2004), which supports the analysis for potential deadlocks and livelocks. Lazy soundness (Puhlmann and Weske, 2006) is another variant that focuses only on the end place and allows for excess tokens in the rest of the net. All the above research in soundness verification is restricted to the control flow perspective only.

6.2 Soundness verification considering data perspective

In recent years, increasing research has focused on the influence of data perspective in verification of workflow correctness. Some researchers focused on data flow anomalies in workflow (Sun *et al.*, 2006; Meda *et al.*, 2007; 2010; Trčka *et al.*, 2009). Fan *et al.* (2007) extended the notion of classical soundness from van der Aalst (1998a) to support the case in which data flow affects control flow. However, no explicit data perspective is considered in the verification algorithm. Since the control/data flow interaction is considered in transition conditions which are used to generate an occurrence graph (OG), actually data parts can also impact the soundness property. Thus, this formal verification method is inefficient. Knuplesch *et al.* (2010) considered data conditions within a process model serving as a preprocessing step to actual compliance checking rather than correctness verification of the business process model. The influence of the data condition on the flow of control was first mentioned in Trčka (2009), where two different situations with data perspective were identified, i.e., false positives and false negatives, but no means of verifying the soundness property was provided. Later, Sidorova *et al.* (2010; 2011) proposed an approach to verify the soundness property of the workflow net taking into account data perspective. The concrete approach in Sidorova *et al.* (2010) is to construct a may-must reachability graph based on standard may/must transition systems that guarantees valid results based on data perspective (e.g., data operations and abstract transition guards). Furthermore, 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.

At first sight, work in Sidorova *et al.* (2010; 2011) is similar to our work. In the following, we analyse further the similarities and differences

between their work and ours.

Sidorova *et al.* (2010; 2011) and our paper both focus on the soundness verification for workflow nets taking into account data perspective. To deal with soundness verification, two challenges should be considered: one is to formally express workflow models with data perspective, and the other is to verify workflow models with any possible data refinement.

For the formalisation of workflow nets with data perspective, the formalisation of the WFD-net in Trčka (2009) and Sidorova *et al.* (2011) is proposed by supporting the representation of data operations and abstract guards. We propose workflow net with transition conditions (that is, WTC-net) and give a concrete definition for the transition condition from both syntactic and semantic points of view. Compared to WTC-net, WFD-net is a more general approach leading to more precise formulation especially from data operations. However, WFD-net does not illustrate further how to support arbitrary concrete guards. In contrast, WTC-net gives more concrete formalisation of the transition condition which is fundamental for further behavioural analysis of WTC-net.

For the analysis approach of workflow nets with data perspective, Sidorova *et al.* (2011) described the semantics of WFD-nets using the concept of hyper transition systems, which allows to connect a single state to a set of possible successor states. Using this analysis method a certain soundness result of any data refinement can be deduced if the WFD-net is must-sound (or not may-sound). However, an accurate soundness result of any data refinement cannot be deduced if the WFD-net is may-sound. Because this method supports only soundness analysis for WFD-net with abstract predicate, it does not provide a formalisation of the concrete condition expression, and hence cannot analyse the soundness topic on the basis of concrete data refinement. In our approach, first, we formalise the term 'condition' from both syntactic and semantic points of view. Subsequently, we analyse the behavioural semantics of a WTC-net based on the influence of concrete transition conditions as well as writing operations and propose a reachability analysis method to verify the soundness property considering data perspective. Specifically, we adopt the symbolic execution technique (Clarke, 1976; King, 1976; Khurshid *et al.*, 2003) to translate the reachability problem into a symbolic constraint system, and then handle the state space explosion caused by infinite or finite data dimension. Subsequently, we propose an approach to deal with the termination of repeated cyclic executions in the reachability space of a WTC-net. Thus, the ambiguous soundness result of any data refinement deduced from the may-soundness result of the corresponding WFD-net caused by the abstract guard function in Sidorova *et al.* (2011) can be solved to some extent by the approach proposed in our paper. Finally, we implement the proposed approach as a plug-in in ProM 6.0. A practical experiment further illustrates the potential applicability of the approach we propose in the real industry domain.

6.3 Program analysis

The topic we examine in this paper is also similar to the topic of program analysis to some extent. Here we discuss the relationship between our work and program analysis. Program analysis includes two main facts: data flow analysis and program verification.

To optimise the representation of data flow, data flow analysis can be implemented using the single static assignment (SSA) form (Cytron *et al.*, 1991). The concepts of SSA explained so far hold for sequential programs only. Therefore, an extension of SSA for programs containing concurrency has been proposed by Lee *et al.* (1998). The concurrent single static assignment (CSSA) form has already been applied to some research works on the analysis topic in business process management (Moser *et al.*, 2007; Heinze *et al.*, 2009; 2011). Moser *et al.* (2007) presented a method to extract data flow information by constructing a CSSA representation and detecting data dependencies that affect communication behaviour. Those discovered dependencies were used to construct a more precise formal model of the given BPEL process and hence to improve the quality of analysis results. Heinze *et al.* (2009) used a combination of workflow graphs and CSSA to restructure WS-BPEL processes. Based on the hybrid format, it is able to identify loops with a static quasi-constant loop condition and transform them in such a way that conditional control flow is replaced by unconditional control flow. Later, Heinze *et al.* (2011) introduced a restructuring technique to resolve data-based choices with conditions over variables, whose

values are determined by the contents of incoming messages.

Note that the notion 'fresh variable' introduced in our paper has a similar idea of SSA to some extent. The difference is that the 'fresh variable' is used as replacement for variable whenever the variable has received a new value during the running time. However, SSA is used as replacement for variable when the variable is defined in a control flow graph (CFG). We introduce the notion 'fresh variable' to represent the snapshot of the writing operation that occurred each time in a loop structure in running time. However, SSA cannot describe clearly this situation. For example, in SSA (or CSSA) form the single assignment property is static, which means that a variable assignment inside a loop is regarded as a single definition. However, the control flow may reach that assignment more than once when the program is executed. That is, a variable inside a loop gets a new value assigned each time the loop is executed, but for each iteration it does not get a new subscript of SSA form for the variable.

On the other hand, both the symbolic execution technique (Clarke, 1976; King, 1976; Khurshid *et al.*, 2003) and the predication abstraction technique (Graf and Saïdi, 1997) have been widely used in the program verification area. Symbolic execution represents values of program variables with symbolic values instead of concrete (initialized) data and manipulates expressions involving symbolic values. Symbolic execution traditionally arose in the context of checking sequential programs. Here, we extend it with the underlying WF-net to a parallel model. The predication abstraction technique is devoted to automatically construct abstract state graphs paying a reasonable price. Here, we directly address the satisfiability of the history constraint along a sequence by an SMT solver. There can be an improvement of algorithm efficiency if we combine the symbolic execution technique and the predication abstraction technique in the future.

7 Conclusions

In this paper, we have defined workflows net with transition conditions as WTC-nets and have given it behavior semantics taking into account the influence of transition conditions in detail. Using this definition, we have presented an approach to verify workflow correctness based on WTC-nets and considered the technique to solve state space problems. Finally we described our implementation of the proposed approach in a form of a plug-in in the ProM 6.0 framework. This plug-in was used to test the soundness of a number of real-life models. The application of the plug-in to these models shows that our approach is able to correctly assess the correctness of process definition with a concrete data refinement.

Our current work provides a qualitative method of verifying process models with transition conditions, that is, determining whether a sequence can be executed based on the satisfiability of the constraint derived from the transitions along the sequence. Building on this, in the future, we will consider a quantitative technique of verifying process models with transition conditions, for example, given a concrete solution space for an executable path, or an unsatisfiable solution space for a non-executable path.

Acknowledgements

We are grateful to THsoft InfoTech for providing process models with data information related to transition conditions, all contained in their TiPLM system. Furthermore, we would like to thank Arthur ter HOFSTEDE and Moe WYNN of Queensland University of Technology for their constructive suggestions, Eric VERBEEK of Technische Universiteit Eindhoven for his help in interfacing our soundness analyser with Woflan, and Christian STAHL of Technische Universiteit Eindhoven for his useful comments. Part of the work was conducted during a visit of the first author to Queensland University of Technology, Brisbane, Australia.

References

Clarke, L.A., 1976. A system to generate test data and symbolically execute programs. *IEEE Trans. Software Eng.*, **2**(3):215-222. [doi:10.1109/TSE.1976.233817]

Curran, T., Keller, G., 1998. SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K., 1991. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, **13**(4):451-490. [doi:10.1145/115372.115320]

Dehnert, J., van der Aalst, W.M.P., 2004. Bridging the gap between business models and workflow specifications. *Int. J. Cooper. Inf. Syst.*, **13**(3):289-332. [doi:10.1142/S0218843004000973]

Dutertre, B., de Moura, L.M., 2006a. A Fast Linear-Arithmetic Solver for DPLL(T). 18th Int. Conf. on Computer Aided Verification, p.81-94. [doi:10.1007/11817963_11]

Dutertre, B., de Moura, L., 2006b. The Yices SMT Solver. Available from http://yices.csl.sri.com/tool-paper.pdf

Fan, S., Dou, W., Chen, J., 2007. Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. Advances in Web and Network Technologies and Information Management, p.433-444. [doi:10.1007/978-3-540-72909-9_46]

Franco, J.V., 2005. Typical case complexity of satisfiability algorithms and the threshold phenomenon. *Discr. Appl. Math.*, **153**(1-3):89-123. [doi:10.1016/j.dam.2005.05.008]

Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C., 2004. DPLL(T): Fast Decision Procedures. 16th Int. Conf. on Computer Aided Verification, p.175-188. [doi:10.1007/978-3-540-27813-9_14]

Graf, S., Saïdi, H., 1997. Construction of Abstract State Graphs with PVS. 9th Int. Conf. on Computer Aided Verification, p.72-83. [doi:10.1007/3-540-63166-6_10]

Heinze, T.S., Amme, W., Moser, S., 2009. A Restructuring Method for WS-BPEL Business Processes Based on Extended Workflow Graphs. 7th Int. Conf. on Business Process Management, p.211-228.

Heinze, T.S., Amme, W., Moser, S., 2011. Process Restructuring in the Presence of Message-Dependent Variables. Service-Oriented Computing - ICSOC 2010 Int. Workshops, PAASC, WESOA, SEE, and SOC-LOG, p.121-132.

Karp, R.M., Miller, R.E., 1969. Parallel program schemata. *J. Comput. Syst. Sci.*, **3**(2):147-195. [doi:10.1016/S0022-0000(69)80011-5]

Khurshid, S., Pasareanu, C.S., Visser, W., 2003. Generalized Symbolic Execution for Model Checking and Testing. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, p.553-568. [doi:10.1007/3-540-36577-X_40]

King, J.C., 1976. Symbolic execution and program testing. *Commun. ACM*, **19**(7):385-394. [doi:10.1145/360248.360252]

Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P., 2010. On Enabling Data-Aware Compliance Checking of Business Process Models. 29th Int. Conf. on Conceptual Modeling, p.332-346.

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

Lee, J., Midkiff, S.P., Padua, D.A., 1998. Concurrent Static Single Assignment Form and Constant Propagation for Explicitly Parallel Programs. 10th Int. Workshop on Languages and Compilers for Parallel Computing, p.114-130. [doi:10.1007/BFb0032687]

Martens, A., 2003. On compatibility of Web services. *Petri Net Newslett.*, **65**:12-20.

Martens, A., 2005. Analyzing Web Service Based Business Processes. 8th Int. Conf. on Fundamental Approaches to Software Engineering, p.19-33. [doi:10.1007/978-3-540-31984-9_3]

Meda, H.S., Sen, A.K., Bagchi, A., 2007. Detecting Data Flow Errors in Workflows: a Systematic Graph Traversal Approach. Proc. 17th Workshop on Information Technologies and Systems, p.133-138.

Meda, H.S., Sen, A.K., Bagchi, A., 2010. On detecting data flow errors in workflows. *J. Data Inf. Qual.*, **2**(1):1-31. [doi:10.1145/1805286.1805290]

Meyer, B., 1990. Introduction to the Theory of Programming Languages. Prentice-Hall, Upper Saddle River, NJ, USA.

Moser, S., Martens, A., Görlach, K., Amme, W., Godlinski, A., 2007. Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-Based Data Flow Analysis. IEEE Int. Conf. on Services Computing, p.98-105. [doi:10.1109/SCC.2007.22]

Murata, T., 1989. Petri nets: properties, analysis and applications. *Proc. IEEE*, **77**(4):541-580. [doi:10.1109/5.24143]

Puhlmann, F., Weske, M., 2006. Interaction Soundness for Service Orchestrations. 4th Int. Conf. on Service-Oriented Computing, p.302-313.

Rushby, J., 2006. Threatened by a Great Opportunity: Disruptive Innovation in Formal Verification. Federated Logic Conf.

Schmidt, K., 2003. Using Petri Net Invariants in State Space Construction. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, p.473-488. [doi:10.1007/3-540-36577-X_35]

Sidorova, N., Stahl, C., Trčka, N., 2010. Workflow Soundness Revisited: Checking Correctness in the Presence of Data while Staying Conceptual. 22nd Int. Conf. on Advanced Information Systems Engineering, p.530-544.

Sidorova, N., Stahl, C., 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. [doi:10.1016/j.is.2011.04.004]

Sun, S.X., Zhao, J.L., Nunamaker, J.F., 2006. Formulating the data-flow perspective for business process management. *Inf. Syst. Res.*, **17**(4):374-391. [doi:10.1287/isre.1060.0105]

Trčka, N., 2009. Workflow Soundness and Data Abstraction: Some Negative Results and Some Open Issues. Int. Workshop on Abstractions for Petri Nets and Other Models of Concurrency, p.19-25.

Trčka, N., van der Aalst, W.M.P., Sidorova, N., 2009. Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows. 21st Int. Conf. on Advanced Information Systems Engineering, p.425-439.

Tsang, E.P.K., 1993. Foundations of Constraint Satisfaction. Academic Press, London, UK.

van der Aalst, W.M.P., 1997. Verification of Workflow Nets. 18th Int. Conf. on Application and Theory of Petri Nets, p.407-426.

van der Aalst, W.M.P., 1998a. The application of Petri nets to workflow management. *J. Circ. Syst. Comput.*, **8**(1):21-66. [doi:10.1142/S0218126698000043]

van der Aalst, W.M.P., 1998b. Information and Process Integration in Enterprises: Rethinking Documents. Kluwer Academic Publishers, Norwell, p.161-182. [doi:10.1007/978-1-4615-5499-8_10]

van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, H.M.W., Weijters, A.J.M.M., 2009. ProM: the Process Mining Toolkit. Proc. Business Process Management Demonstration Track, p.9-12.

van der Aalst, W.M.P., van Hee, K.M., 2002. Workflow Management: Models, Methods, and Systems. MIT Press, London, UK.

van Hee, K.M., Sidorova, N., Voorhoeve, M., 2003. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. 24th Int. Conf. on Applications and Theory of Petri Nets, p.337-356.

van Hee, K.M., Sidorova, N., Voorhoeve, M., 2004. Generalised Soundness of Workflow Nets is Decidable. 2nd Int. Conf. on Business Process Management, p.197-215.

Verbeek, H.M.W., 2004. Verification of WF-Nets. PhD Thesis, Eindhoven University of Technology, Eindhoven, the Netherlands.

Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P., 2001. Diagnosing workflow processes using Woflan. *Comput. J.*, **44**(4):246-279. [doi:10.1093/comjnl/44.4.246]

Wang, Z., Wang, J., Wen, L., Liu, Y., 2009. Deriving Canonical Business Object Operation Nets from Process Models. Proc. 11th Int. Conf. on Enterprise Information Systems, p.182-187.

Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D., 2006. Verifying Workflows with Cancellation Regions and OR-Joins: an Approach Based on Reset Nets and Reachability Analysis. 4th Int. Conf. on Business Process Management, p.389-394.

Zha, H., van der Aalst, W.M.P., Wang, J., Wen, L., Sun, J., 2011. Verifying workflow processes: a transformation-based approach. *Software Syst. Model.*, **10**(2):253-264. [doi:10.1007/s10270-010-0149-9]

Appendix: Condition

This appendix contains the definitions capturing the syntax of the notion of a condition as used in this paper. In our definitions of syntax throughout this part we base ourselves as much as possible on the notations used in Meyer (1990). In addition, in line with the argumentation in this book with respect to the matter of concrete syntax versus abstract syntax, we prefer to use abstract syntax throughout to be able to focus on the essence of concepts and not the way by which they may be represented.

Definition A1 (Condition) A condition is an instance of the syntactic class cond. This class is governed by the following abstract syntax:

$$\begin{aligned}
\text{cond} &\triangleq \text{ele_cond} \mid \text{bi_cond} \mid \text{un_cond}\\
\text{ele_cond} &\triangleq \text{const} \mid \text{var} \mid \text{comparison}\\
\text{comparison} &\triangleq t_1,\ t_2 : \text{term};\ o : \text{com_oper}\\
\text{term} &\triangleq \text{bi_ari_expr} \mid \text{const} \mid \text{var}\\
\text{bi_ari_expr} &\triangleq b_1,\ b_2 : \text{term};\ o : \text{bi_ari_oper}\\
\text{bi_cond} &\triangleq b_1,\ b_2 : \text{cond};\ o : \text{bi_bool_oper}\\
\text{un_cond} &\triangleq u : \text{cond};\ o : \text{un_bool_oper}\\
\text{const} &\triangleq y : \text{type}; d : \text{value}\\
\text{com_oper} &\triangleq \text{le} \mid \text{leq} \mid \text{ge} \mid \text{geq} \mid \text{eq}\\
\text{bi_bool_oper} &\triangleq \text{and} \mid \text{or} \mid \text{implies}\\
\text{bi_ari_oper} &\triangleq \text{plus} \mid \text{minus} \mid \text{times} \mid \text{frac} \mid \text{mod} \mid \text{div}\\
\text{un_bool_oper} &\triangleq \text{neg}\\
\text{type} &\triangleq \text{int} \mid \text{real} \mid \text{bool}\\
\text{value} &\triangleq \mathbb{Z} \mid \mathbb{R} \mid \mathbb{B}
\end{aligned}$$

Since we use examples of conditions, it is convenient to also have a concrete syntax for conditions to go with the abstract syntax of Definition A1. It should be straightforward to match our use of concrete syntax to the abstract syntax. To this end, we deploy the usual symbols for the various operations (e.g., + for plus). Also, when we represent a constant we assume that it is clear enough about what the intended type of the constant is.

Conditions that are in accordance with the syntax presented in Definition A1 may be erroneous as the syntax is quite liberal. In particular, conditions may contain operations that are applied to constants or variables of the wrong type. To be able to focus on conditions that are well-formed, we introduce a number of static semantic functions. First, we need a function that given a term and a typing of variables, determines the type of this term. As far as types are concerned, we focus only on a number of basic types, and introduce the collective name Types for this set. This set is defined as {`int`, `real`, `bool`, `undefined`}, where `undefined` is a special type to represent the situation where a term cannot be assigned a meaningful type. In the context of a particular assignment of types to variables (VAR will be used as the set of all such assignments), we can determine the type of a term through application of the function TypeOf : term × VAR → Types.

TypeOf (t : term, VT : VAR) ≜
  case t **of**
    const ⇒
      case $t.y$ **of**
        int ⇒ `int` | real ⇒ `real` | bool ⇒ `bool`
      end |
    var ⇒ VT(t) |
    bi_ari_expr ⇒
      if TypeOf($t.b_1$, VT) = `int` ∧
        TypeOf($t.b_2$, VT) = `int`
      then
        if $t.o \neq \div$ **then** `int` **else** `real` **end**
      elif (TypeOf($t.b_1$, VT) = `int` ∧
            TypeOf($t.b_2$, VT) = `real`) ∨
          (TypeOf($t.b_1$, VT) = `real` ∧

```
            TypeOf(t.b2, VT) = int) ∨
          (TypeOf(t.b1, VT) = real ∧
            TypeOf(t.b2, VT) = real)
        then if t.o ∉ {mod, div}
          then real else undefined end
        else undefined
        end
    end
  end
```

Using the TypeOf function we can determine whether a condition is well-formed through application of the function $V_{\text{wfc}} : \text{cond} \times \text{VAR} \to \text{bool}$. This function is defined as follows:

```
V_wfc(c : cond, VT : VAR) ≜
  case c of
    ele_cond ⇒
      case c of
        const ⇒ c.y = bool |
        var ⇒ VT(c) = bool |
        comparison ⇒
          (TypeOf(c.t1, VT) ∈ { int, real} ∧
            TypeOf(c.t2, VT) ∈ { int, real}) ∨
          (TypeOf(c.t1, VT) = bool ∧
            TypeOf(c.t2, VT) = bool ∧ c.o = eq)
      end |
    bi_cond ⇒ V_wfc(c.b1, VT) ∧ V_wfc(c.b2, VT) |
    un_cond ⇒ V_wfc(c.u, VT)
  end
```

An important auxiliary function is the function $V_{\text{var}} : \text{cond} \to \text{setofvar}$. The application of this function to a condition yields the variables that are used in this condition.

```
V_var(c : cond) ≜
  case c of
    ele_cond ⇒
      case c of
        const ⇒ ∅ | var ⇒ {c} |
        comparison ⇒ V_var(c.t1) ∪ V_var(c.t2)
      end |
    bi_cond ⇒ V_var(c.b1) ∪ V_var(c.b2) |
    un_cond ⇒ V_var(c.u)
  end
```

The function $V_{\text{var}} : \text{term} \to \text{setofvar}$ for conditions uses a similarly named function on terms defined as follows:

```
V_var(t : term) ≜
  case t of
    const ⇒ ∅ | var ⇒ {t} |
    bi_ari_expr ⇒ V_var(t.b1) ∪ V_var(t.b2)
  end
```