Property-Transition-Net-Based Workflow Process Modeling and Verification

Yu Huang, Hanpin Wang, Peng Yu, Yunni Xia

School of Electronics Engineering and Computer Science
Peking University, China

Abstract

The development of workflow management system requires satisfactory models and concepts. As mentioned in [14], classical Petri nets are not suitable to describe some advanced workflow patterns, thus this paper presents a Petri-net-based model characterized by places with new properties which are suitable to represent some advance workflow patterns. Many workflow models confuse the behaviors of workflow engine and external environment(tasks), and fail to describe the real semantics of workflow engine. In our model, through separating transitions from routings, the place will describe the behaviors of engine and the transition will describe the behavior of task. Because workflow engine is a reactive system, the token-game semantic behavior of a Petri net-based workflow model will differ from its behavior at run time. However, the semantics of property-transition-net-based model with ST firing rules is suitable enough to capture the behavior of workflow process. In this paper the formal definitions and semantics of this model are discussed in detail. This paper concludes with an introduction of a verification technique for structure-soundness detection based on some new reduction rules.

Keywords: Business process modeling, verification of process model, workflow pattern, property transition nets

1 Supported by the National Natural Science Foundation of China under Grant No. 60173002 and the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312004.
2 Email: hy@pku.edu.cn
3 Email: whpxhy@pku.edu.cn
4 Email: peng.yu@oracle.com
5 Email: xiayunni@pku.edu.cn

1571-0661 © 2006 Elsevier B.V. Open access under CC BY-NC-ND license.
1 Introduction

Workflow has become an essential technology in e-business which provides a flexible and appropriate environment for development and maintenance of next-generation component-oriented information systems for enterprise applications [11]. Nowadays, workflow management systems have widely been deployed in the domains of administration, production, and scientific research in order to ensure high efficiency of business process.

Workflow process modeling can be understood from a number of different perspectives [15]. The control flow perspective describes activities and their execution orders through different constructors. Presently, many models of workflow process have been introduced. e.g., van der Aalst and his collaborators provided workflow net (WF-net) based on Petri nets to model the workflow process [12][13]; Alonso and Agrawal in [1][8] gave a workflow model (ADEPT) depending on a conceptual graph which aimed at solutions of transaction problems and the flexibility in workflow process; Wasim Sadiq used a directed acyclic graph (DAG) to construct the process model; and [3] introduced a new model of workflow process based on reference nets.

A workflow model will be deployed into workflow engine to be interpreted. Therefore it must describe the behavior of the engine. At run time, the engine based on the workflow model dispatchs some new tasks to work list and some completed tasks from its environment return signals to engine. Tasks must have two state at least, i.e., ready state and completed state. Tasks have no ability to decide the routings. Nevertheless in some workflow models such as WF-net, the tasks can represent the routings and these models are not elegant solutions for modeling workflow process. Actually, the benefit of a workflow management is the separation of the procedure logic and the business execution. Consequently the applications represented by tasks only concern the business executions. A workflow model describes the behavior of engine, however in many models such as WF-net, DAG, there do not explicitly exist the formal description for the actions of engine. In these models, the actions of workflow engine and tasks are confused so that the behavior of workflow engine cannot be analyzed.

An action of engine mainly represent decision-making. As mentioned in [13], the classical Petri net is not suitable to represent many different decision-makings, i.e., the workflow patterns or routings. To overcome the difficulty of describing some advance workflow patterns, in [14], a new model YAWL (yet another workflow language) is proposed which has Petri net semantics. Since YAWL adds many element to prescribe the patterns, it is not a suitable solution. Also, the formal semantics of YAWL is too complicated to understand.
Furthermore, workflow engines are reactive systems. A reactive system runs concurrently with its environment and enforces certain desirable effects in the environment [9]. When the actions of engine are activated, they must occur immediately. The executions of tasks are not the behaviors of engine, therefore tasks must have two states, namely ready and completed. Unfortunately in many models, their semantics fails to express the behaviors of state and interaction between workflow engine and its environment.

While building a large workflow model, it is possible to introduce error situations. Such modeling inconsistencies and errors can lead to undesirable execution of some or all possible instances of a workflow[15]. The familiar errors are structural conflicts in process model such as deadlock and lack of synchronization [11] [12]. It is necessary to rectify such problems at design phase rather than after deploying the process into the workflow management. Therefore the correctness of the business process supported by the workflow system is important to its organization. Many researchers use the reduction technique to detect the errors within workflow models. However their reduction rules are not complete, such as [11], etc.

Focusing on the above issues, this paper aims at defining a well-suited workflow model based on Petri nets from the control flow perspective. In the model, the actions of engine and tasks are separated and respectively represented by places and transitions. Different from the classical Petri nets, the places can actively occur; transitions, which have two states, namely firing(ready) and fired(completed), can have only one input place and one output place; tokens also have two states: finished and starting. In our model, some new occurrence rules are given to express the semantics of the model. Accordingly some new reduction rules are proposed to verify the model.

The remainder of this paper is structured as follow. Section 2 states the formal definition and the semantics based on some new firing rules. This section also detailedly presents the description of advanced patterns. Section 3 gives some discussions of new reduction rules for verifying the soundness of workflow process model. The paper concludes in Section 4.

2 Property Transition Nets

In workflow reference model [10], there are five workflow process routings to be introduced: sequence, iteration, OR-Join, OR-Split, Parallel routing. van der Aalst and ter Hofstede in [13] advanced a number of different workflow process routings called workflow patterns. Actually, those twenty patterns can be separated into static patterns and dynamic patterns. Static patterns, including basic control flow patterns, advanced branching and synchronization
patterns, and arbitrary cycles pattern[13], are unmodified after deploye d. Others are dynamic patterns which will be influenced in instances or at run time. Because from the control flow perspective we cannot hold the changes of data which determine the dynamic property of control flow, we only consider the static patterns in this paper.

Classical Petri nets can express some patterns naturally, such as XOR and AND routings. Nevertheless, in [14], van der Aalst and his coauthors thought, when it came to patterns involving multiple instances, advanced synchronization, and cancelation, classical Petri nets offered little support. Therefore, they addressed another more suitable Petri-net-based language YAWL to describe the workflow process and to support more patterns. In [1][8][11], graph-based workflow models were introduced, and those models only realized basic control flow patterns. [3] figured out some difficulties of describing advanced patterns using reference nets.

**Problems and Formal Model**

In viewpoints of Mary Shaw and David Garlan, software architecture can be split into components and connectors. Components deal with computing and business logics, while connectors describe the interaction between the components. The model of workflow process can also be separated into components which represent the activities of the workflow process and connectors which represent routings between control flows. However, according to classical Petri nets, the parallel-routing and the choice-routing are represented by transitions and places respectively. Therefore, places and transitions both express the routings.

Fig.1 models a WF-net of travel arrangement process from [7]. As shown above, the tasks are also responsible of routings. After a workflow process is deployed into a workflow management software system, the workflow engine will execute process logic according to the routings among tasks, and dispatch tasks to actors or other computer systems. Obviously, if the workflow process model definitely separate the tasks and the routings, the structure and function of workflow can be clearly laid out in a process model.

Unfortunately it is both complex and inconvenient to model advanced patterns through classical Petri nets, such as the representation of the M-out-of-N pattern(Fig.2). Therefore, extension is needed so that advanced patterns can be easily expressed.

Our model is based on coloured Petri nets(CPN). The expressions of arcs represent the temporal constraints of routing and the guards of transitions specify tasks. We present the CPN definition as in [6]. Function Type and var apply in this context: Type returns the type(s) of variable(s) or functions(s);
Var returns the set of variables present in an attribute (or parameter) list, and expression, or a function. The subscript \(M_S\) denotes multisets (bags), and \(\mathbb{B} = \{true, false\}\).

**Definition 2.1** [Coloured Petri Nets, CPN [2][5][6]] A Coloured Petri nets is a tuple \(\mathcal{N} = (\Sigma, P, T, A, N, C, G, E, I)\) satisfying the following requirements:

(i) \(\Sigma\) is a finite set of non-empty types called colour sets (or simply colours).

(ii) \(P\) is a finite set of places.

(iii) \(T\) is a finite set of transitions.

(iv) \(A\) is a set of arcs such that: \(P \cap T = A \cap P = A \cap T = \emptyset\).

(v) \(N\) is a node function defined from \(A\) into \((P \times T \cup T \times P)\).

(vi) \(C\) is a colour function mapping each place \(p \in P\) into a colour from \(\Sigma\).

(vii) \(G\) is a guard function. It is defined from \(T\) into expressions such that:
\[
\forall t \in T, Type(G(t)) = \mathbb{B} \land Type(Var(G(t))) \subseteq \Sigma
\]

(viii) \(E\) is an arc expression function. It is defined from \(A\) into expressions such that:
\[
\forall a \in A, Type(E(a)) = C(p(a))_{MS} \land Type(Var(E(a))) \subseteq \Sigma \text{ where } p(a) \text{ is the place connected to } a.
\]

(ix) \(I\) is an initialization function mapping each place \(p \in P\) into a multiset over \(C(p)\).

Since coloured Petri nets are equivalent to Turing machine, it can describe
the workflow process and express various patterns. However, modeling the workflow process by coloured Petri nets is both complex and inapplicable for generic modelers who are unfamiliar with Coloured Petri nets. The definition of our model is presented as below.

**Definition 2.2** [Property Transition Net] A property transition net is a tuple $\text{PTNet}=\left(\Sigma \cup \{f, s\}, P, T, A, N, C \cup \mathcal{C}, G, E \cup \mathcal{E}, I \cup \mathcal{I}, \Theta, \Lambda\right)$ satisfying the following requirements:

(i) $\mathcal{N}=\left(\Sigma, P, T, A, N, C, G, E, I\right)$ is a CPN which describes the details of business process except the relations of control flow i.e. the routings.

(ii) $\Lambda$ is a finite set of variables, $\Lambda \subseteq \Pi^* (\Pi$ represents an alphabet). $\Lambda$ contains interface variables which affect the routings and can be changed by tasks and workflow engine. This parameter is used in patterns.

(iii) $f$ and $s$ represent the signals of control flow. The colour $f$ represents the finished signal, and $s$ represents the starting signal.

(iv) $\Theta$ is a place property function. It is defined from $P$ into $(\mathbb{N} \cup \Lambda) \times (\mathbb{N} \cup \Lambda)$, is a set of natural number. One place attached with property of $(a, b) \in (\mathbb{N} \cup \Lambda) \times (\mathbb{N} \cup \Lambda)$, means when the finished signals in the place is equal ‘$a$’, the place produces ‘$b$’ starting signals at the same time ‘$a$’ finished signals is consumed. The property function changes the occurrence rules and semantics of the model which is discussed in next subsection.

(v) $\forall e \in P \cup T, \text{PRE}(e) = \{N(\alpha)(e) | \exists \alpha \in A\}, \text{POST}(e) = \{\beta | \exists \beta \in P \cup T \wedge \exists \alpha \in A \wedge N(\alpha)(\beta) = e\}$. The property transition net constrains the number of all transitions pre-set and post-set in order for modeling workflow process.

\[ \forall t \in T, |\text{PRE}(t)| = |\text{POST}(t)| = 1. \]

(vi) $|\{p | p \in P \wedge |\text{PRE}(p)| = 0\}| = 1 \wedge |\{p | p \in P \wedge |\text{POST}(p)| = 0\}| = 1$. This means the PTNet has only one starting node and one termination node, which is the same as the WF-net. The place $p$ of $|\text{PRE}(p)| = 0$ is identified as ‘$b$’, called **beginning place**, and the place $p$ of $|\text{POST}(p) = 0|$ is identified as ‘$e$’, called **ending place**.

(vii) $\mathcal{C}, \mathcal{E}, \mathcal{I}$ are functions only related to colours $f$ and $s$. $\mathcal{C}$ is a colour function mapping each $p \in P$ into $\{f, s\}$; in this paper, $\mathcal{C}$ is a constant function which means all places can contain token with colour $f$, $s$, or both. $\mathcal{E} \subseteq (A \cap P \times T) \times \{1.s\} \cup (A \cap (A \cap T \times P) \times \{1.f\})$, is the weight function of arc. This means output arc which arrow points to a transition consumes one starting signal and input arc produces one termination signal. $\mathcal{I}$ is an initialization function of colour $f$ and $s$, where $\forall p \in P, p \neq b \Rightarrow \mathcal{I}(p) = 0$ and $\mathcal{I}(b) = 1.s$,
The **PTNet** enhances the CPN and represents the workflow process in an elegant way. \( f \) and \( s \) are signals of control flow. In this paper, we ignore how to modify the variables in \( \Lambda \) and other colours in \( \Sigma \). Places attached with properties of 2-dimensions vectors actually represent the constraints of routings, and transitions represent the task activated by workflow engine. Since the model rigorously separates the task and the routing of control flow, it directly represents different functional units of workflow process.

![PTNet model of travel arrangement](image)

**Fig. 3. A PTNet: Travel Arrangement**

Fig.3 shows a **PTNet** model of travel arrangement. The nodes of places are attached with a vector as \([m,n]\), and other places unattached have the default property \([1,1]\). The property of \([m,n]\) can be simply interpreted as when there are \( m \) tasks in the pre-set of the place finished, the engine will generate \( n \) tasks into starting pool and notify respective actors. The description of the routings is very intuitive. The dashed tasks identified as \( \text{skip} \) directly transfer the signals of control flow and other colour tokens without modification. Intuitively, according to the semantics of the property, the **PTNet** of travel arrangement describes the same process as the Fig.1. The following subsection will give the property a formal semantics. The model illustrated by Fig.3 is obviously superior to that of Fig.1.

**Occurrence Rules**

In the literature of Petri nets, the occurrence rules present the semantics of behavior. Based on those rules, some dynamic properties of a workflow process model, such as process termination, can be analyzed through the reachability graph. Given the occurrence rules, we can also simulate the workflow process, and evaluate some runtime performances.

The simplest occurrence rule is the *interleaving rule* [7]. It assumes the firing of transition step by step. Another occurrence rule that discriminates between concurrent and sequential behavior is the *ST firing rule* [7]. In this paper, we modify the *ST firing* rules so that the semantics of **PTNet** suits the practical meaning of workflow by supporting semi-true concurrency.
In our model, there are two kinds of colour sets $\Sigma$ and $\{f, s\}$. The following occurrence rules mainly focus on the tokens which are directly related to the control flow, i.e. $\{f, s\}$ and the tokens with colours in $\Sigma$ take an effect in the expression and guard of tasks.

In a system of workflow management, it is clear and necessary to represent executable tasks and completed tasks definitely. Consequently it is necessary to split the state of the execution of a transition into firing and fired. Unfortunately, the marking of CPN cannot perfectly depict the state of a PTNet system.

Definition 2.3 A binding\cite{5} of a transition $t$ is a function $b$ defined on $\text{Var}(t)$, such that: $\forall v \in \text{Var}(t) : b(v) \in \text{Type}(v)$ and $G(t) < b >$. By $B(t)$ we denote the set of all bindings for $t$. Respectively, $\mathcal{B}$ denotes the superset of all bindings for $T$. In our model, there are no variables of $\{f, s\}$. Therefore no bindings are related to colours $\{f, s\}$.

Definition 2.4 Give a PTNet system $\text{PTNS} = (\text{PTNet}, [M, T])$, $\text{PTNet} = (\Sigma \cup \{f, s\}, P, T, A, N, C, G, E, I, T, \Theta, \Lambda)$, where $[M, T]$ describes the marking or state of a PTNet system. We define $M : P \rightarrow (\Sigma \cup \{f, s\})_{MS}$, and $T \subseteq \mathbb{N} \times (T \times \mathcal{B})$ that is multiset over $T \times \mathcal{B}$, that represents the table of firing transitions with a binding. Because no transitions is during firing step in the initial state, $M_0$ is equal to $I \cup \mathcal{T}$ and $T_0$ is equal to $\varnothing$. Initial state $i$ is represented by $M_0$. Ending state $o$ means that only the place $e$ have a control token with colour $s$ and $T$ is equal to $\varnothing$. $\mathcal{T}$ indicates the transition $t$ is in firing, and $t$ indicates the transition $t$ is fired. $\overline{T}$ and $\mathcal{T}$ represent the respective sets of transitions . In $\mathcal{T}$, it is not necessary to distinguish the firing and fired transitions. In the following we replace $[M, T]$ to $M$.

Definition 2.5 [One Step Transition Occurrence Rule] One transition step $Y$ can occur in a marking $M = [M, T]$ iff the following request is satisfied:

(i) $\forall (\alpha, b) \in Y : \alpha \in \overline{T} \cup T$, $b$ is a binding. The step $Y$ represents a set of firing and fired transition.

(ii) $t \in T \land (\overline{t}, b) \in Y \implies |Y \cap (\overline{T} \times \mathcal{B})| = \Theta(PRE(t))[2], \Gamma = POST(PRE(t))$. \_\_\_\_\_\_\_\_[n] represents the $n$-th component. In general, workflow engine dispatch concurrent tasks which have the same pre-condition, i.e. connecting the same place. To suit the practice, this formula gives the constraint.

(iii) $\forall p \in P : \sum_{(\alpha, b) \in Y \cap T \times \mathcal{B}} E(p, \alpha) < b > + \mathcal{E}(p, a) \leq M(p)$

(iv) $\forall (\alpha, b) \in (Y \cap (T \times \mathcal{B})) \implies (a, b) \in T$

The occurrence of a step $Y$ in marking $M_1 = (M_1, T_1)$, modifies $M_1$ to
\[ M_2 = (M_2, T_2), \text{ defined by: } \]
\[ \forall p \in P : M_2(p) = M_1(p) - \sum_{(\alpha, b) \in Y \cap T \times \mathcal{B}} (E(p, \alpha)<b> + \mathcal{E}(p, \alpha)) \]
\[ + \sum_{(\delta, b) \in Y \cap T \times \mathcal{B}} (E(\delta, p)<b> + \mathcal{E}(\delta, p)) \]

and \[ T_2 = T_1 + Y \cap \overline{T} \times \mathcal{B} - Y \cap T \times \mathcal{B} \]

\[ M_2 \] is directly reachable from \[ M_1 \]. This is written: \[ M_1[Y > M_2] \]

**Definition 2.6** [One Step Place Occurrence Rule]In our model, the property of place represents the executive logic of workflow engine. Therefore, place has active semantics.

A place step \( X \subseteq P \) can occur in a marking \( M = [M, T] \) iff the following property is satisfied: \( \forall p \in X : M(p) \geq \{\Theta(p)[1].f\} \).

The occurrence of place step \( X \) in a marking \( M_1 \), modifies \( M_1 \) to \( M_2 \), defined by: \( \forall p \in X : M_2(p) = M_1(p) - \{\Theta(p)[1].f\} + \{\Theta(p)[2].s\} \).

Nevertheless, the set of \( T \) remain intact.

**Definition 2.7** [A Finite Occurrence Sequence]A finite occurrence sequence is a sequence of markings and steps:

\[ M_1[Y > M_2[X > M_3[Y > M_4[X > M_5[... Y > M_{2n}[X > M_{2n+1} \]

such that \( n \in \mathbb{N} \), and \( M_i[X > M_{i+1} \) for all \( i \in \{1, 2, \ldots, n\} \); \( M_1 \) is the start marking, \( M_{2n+1} \) is the end marking and \( n \) is the length. if \( M_1 \cap \mathcal{I} = \mathcal{I} \) and \( M_{2n+1} \cap \{(\varepsilon, 1.f)\} = \{(\varepsilon, 1.f)\} \), then, \( Y_1Y_2 \ldots Y_n \) is called a case executing steps.

Given a **PTNet** model system, we can construct the reachability graph. Through analyzing the reachability graph, many properties about cases can be achieved. When the model illustrated by Fig.3 is initialized, according to the occurrence rules, a case executing steps can be obtained:

\{enter\} \rightarrow \{enter\} \rightarrow \{getinfo \| getinfo \| getbudget\} \rightarrow \{getinfo \| getinfo \| getbudget\} \rightarrow \{approve\} \rightarrow \{approve\} \rightarrow \{skip1 \| skip1\} \rightarrow \{skip1 \| skip2\} \rightarrow \{book\} \rightarrow \{book\} \rightarrow \{ass\&send\} \rightarrow \{ass\&send\}

From our definition, one step such as \{getinfo\}, \{getinfo\} or \{getbudget\}, is not allowed. Thus, the semantics of our model supports semi-true concurrency.

**Patterns**

It is natural that our model can describe the patterns of Sequence, AND-Split, AND-Join, XOR-Split, XOR-Join, and m-Choice. For convenience, we use a four-dimension vector \( (x,a,y,b) \) to express the information about the
property of a place. x and y indicate the degree of input arcs and output arcs respectively. (a,b) denotes the property of place mentioned above. (1,1,1,1),(a,a,i,j),(i,j,a,a),(x,1,i,j) and (i,j,x,1) respectively represent the patterns of Sequence, AND-Split, AND-Join, XOR-Split and XOR-Join. van der Aalst in [14] pointed out that the difficulty of describing OR-Join using Petri nets. Petri nets do not incorporate the construction of a “maybe” split or join. In our model, the PTNet overcome this weakness through introducing the variables in Λ.

As for the representation of OR-Join, there are two different models sketched in [13]: passing information from split to join or sending true/false tokens to each possible selection. [3] provides a method of passing information from split to join through adding a place and using the flexible arc of reference nets. The method of true/false tokens lead to explosion of nodes and difficulty of modeling a big process. Fig.4 is an example of our model to present combination process of OR-Split and OR-Join. Our model describes the number of routings indicated by variables and these variables can be changed by engine or tasks at runtime. Since we only describe the control flow of workflow process, rather than how to change the variables by engine and tasks. Since Fig.4 is not only a simple combination of OR-split and OR-Join, the method proposed in [3] will not work well and the method of true/false token will make the problem more complex. In addition, it is obvious that our model will be rapidly comprehended and can support rapid analysis such as computation of the relations between variables(in the Fig.4, \( a + (c - b) = e \)). For brevity, detailed contents will not be discussed.

3 Verification and Reduction

While building a large workflow specification, it is possible to introduce error situations. Such inconsistencies and errors may lead to undesirable execution of some or all possible instances of a workflow. It is obvious to revise such inconsistencies and errors at modeling phase rather than after deploying the
process into the workflow management. So the correctness of the business process supported by the workflow system is an key for an organization. In general, there are three types of analysis technique for workflow process: validation, verification and performance analysis [12]. In this section, we will focus on the technique on verification.

The definition of correctness or soundness:

**Definition 3.1** [Soundness [15]] A model $PTNet = (\Sigma \cup \{f, s\}, P, T, A, N, C \cup C, G, E \cup E, I \cup I, \Theta, \Lambda)$ is sound if and only if:

(i) For every state $M$ reachable from state $i$, there exists a firing sequence leading from state $M$ to state $o$. Formally:

$$\forall M(i \rightarrow M) \Rightarrow (M \rightarrow o)$$

(ii) State $o$ is the only state reachable from state $i$. Formally:

$$\forall M(i \rightarrow M \land M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in $(PTNet, i)$. Formally:

$$\forall t \in T \exists M, M' i \rightarrow M \rightarrow M'$$

The soundness is a property that all instances must have. In a workflow model, there often exist some structural conflicts such as dead nodes or lack of synchronization. In general, the soundness on control flow is called conflict-free. In the literature of verification technique, reduction is a convenient and simple technique to verify the structural conflicts. By iteratively applying a conflict-preserving reduction process, we can remove all the local structures that are definitely correct. The reduction progress will eventually reduce a structurally correct workflow model to a single place. However, a workflow model with structural conflicts cannot be completely reduced. [11] described some rules to verify workflow model based on graph, but their rule were not complete. The rules in [11] cannot verify the correctness of structure in Fig.5.

![Fig. 5. An Example with Complex Control Flow](image)

We propose some new rules to detect the existence of structural conflicts. The structure illustrated by Fig.5 is proved to be conflict-free through applying those rules.

Our rules contain six categories: Sequence rules, One-Choice rules, Parallel
rules, Loop rules, Equivalent rules, m-Choice rules. In the following, $\rightarrow$ means there exists a transition connecting the left place and the right place of the arrow. $\Rightarrow$ means there are two transitions connecting the left place and the right place of the arrow. $\Rightarrow$ means the left is reduced into the right.

1 **Sequence Rules**

(i) $p_1(1, 1, 1, 1) \rightarrow p_2(x, a, y, b) \Rightarrow p(x, a, y, b)$

(ii) $p_1(x, a, y, b) \rightarrow p_2(1, 1, 1, 1) \Rightarrow p(x, a, y, b)$

(iii) $p_1(x, 1, y, 1) \rightarrow p_2(1, a, 1, b) \Rightarrow p(x, a, y, b)$

Fig. 6. Sequence Rules

Fig. 6 shows the **Sequence rules**. Applying those rules, the sequential structure of a model can be reduced. Through applying these rules, the correct structure is removed without influencing the rest of the model. Therefore, using these rules the model keeps sound or conflict-preserving[11].

2 **XOR-Choice Rules**

(i) $p_1(p, q, i, 1) \rightarrow p_2(1, 1, j, 1) \Rightarrow p(p, q, i + j - 1, 1)$

(ii) $p_1(i, 1, 1, 1) \rightarrow p_2(j, 1, p, q) \Rightarrow p(i + j - 1, 1, p, q)$

(iii) $p_1(p, q, i, 1) \Rightarrow p_2(j, 1, x, y) \Rightarrow p_1(p, q, i - 1, 1) \rightarrow p_2(j - 1, 1, x, y)$

Fig. 7. XOR-Choice Rules

Fig. 7 illustrates the application of **XOR-Choice rules**. Item 1 indicates two XOR-Split can be combined into a single XOR-Split. This reduction rule does not influence the rest of the model. Item 2 means two XOR-Join can merge into one single XOR-Join. Item 3 indicates XOR-Split and XOR-Join can be reduced. In item 3, we only need to focus on two transitions between XOR-Split and XOR-Join. Through applying those rules, the correct structure is reduced without modifying the soundness property of the remaining part.

3 **Parallel Rules**
(i) \( p_1(p, q, i, i) \rightarrow p_2(1, 1, j, j) \Rightarrow p(p, q, i + j - 1, i + j - 1) \)

(ii) \( p_1(i, i, 1, 1) \rightarrow p_2(j, j, p, q) \Rightarrow p(i + j - 1, i + j - 1, p, q) \)

(iii) \( p_1(p, q, i, i) \rightarrow p_2(j, j, x, y) \Rightarrow p_1(q, q, i - 1, i - 1) \ p_2(j - 1, i - 1, x, y)(i > 1, j > 1) \)

Item 1 and item 2 are shown in Fig.8 and these two rules keep the correctness of workflow model. It is likely that there is no similar rule as item 3 in papers of recent years. Fig.9(a) can be reduced to Fig.9(b). Our rule only focuses on one transition between AND-Split and AND-Join, rather than all branches.

![Fig. 8. AND-Split or AND-Join Reduction](image)

![Fig. 9. AND-Split Connecting with AND-Join Reduction](image)

![Fig. 10. Loop Reduction Rule](image)

4 Loop Rules

\( p(i, 1, j, 1) \rightarrow p(i, 1, j, 1) \Rightarrow p(i - 1, 1, j - 1, 1)(i, j > 1) \).

Fig.10 illustrates the structure which can be reduced using this rules. Usually, the pattern of loop is made up of OR-Split and OR-Join. However, this rule only deals with loops with XOR-Splits to avoid infinite loop, which may exist in OR-Split and OR-Join based loop.

Place \( p_1 \) and place \( p_2 \) are reduction equivalent, iff:

- \( p_1 \) and \( p_2 \) have the same property \((p, q, i, i)\), i.e. they represent AND-Split;
- \( \text{PRE}(\text{PRE}(p_1)) = \text{PRE}(\text{PRE}(p_2)), \ \text{PRE}(\text{PRE}(p_1)) \) are all AND-Split;
- \( \text{POST}(\text{POST}(p_1)) = \text{POST}(\text{POST}(p_2)), \ \text{POST}(\text{POST}(p_1)) \) are all AND-Join.

5 Equivalent Rules if \( p_1(p, q, i, i) \) and \( p_2(p, q, i, i) \) are reduction equivalent, one of them can be removed, and the value of \( \Theta(\text{PRE}(\text{PRE}(p_1)))[2] \) and \( \Theta(\text{POST}(\text{POST}(p_1)))[1] \) should be subtracted by one.

In Fig.11(a), any one of \( p_1 \) and \( p_2 \) can be removed and the property of \( p_3, p_4, p_5, p_6 \) will be updated, since \( p_1 \) and \( p_2 \) are equivalent. The outcome of reduction is Fig.11(b). Because one place can be replaced by its equivalent
place, the function of the place is replaced by another and the rule will is soundness-preserving.

6 m-Branching and Synchronization Rules

\[ p_1(p, q, m, k) \xrightarrow{m} p_2(m, k, x, y) \Rightarrow p(p, q, x, y) \quad (if \ m \geq k \lor m = 1) \]

The model with structures of m-Split connecting with m-Join shown by the Fig.12 can be reduced. Through applying this rule, all branch between m-Split and m-Join will be reduced. The structure with m-Split and m-Join is sound structure and this rule do not influence the rest of the model. Therefore, this rule is conflict-preserving.

**Theorem 3.2 (Correctness Theorem)** If a model of PTNet can be reduced to one single place with property \([1, 1]\), the PTNet is sound in control flow.

Because all the rules keep soundness-preserving, the correctness theorem follows naturally.

We keep on iterating the reduction process using the above rules as long it can reduce the size of the model. If the outcome is one single place, it means that the original model before reduction does not contain any structural conflicts. Otherwise, it contains structural conflicts such as deadlock or lack of synchronization [11].

In [11], the authors claimed that their rules were complete. Unfortunately, if the sound model shown by Fig.5 is transferred into their respective graph model, the sound graph model cannot be reduced through applying their rules. However, Fig.13-17 are some steps of our reduction process.

Comparing with the rules in [11], Our rules have some obvious advantages. Our rules are simpler, more atomic and precise than those in [11]. Also, our
rules support advanced branching and synchronization patterns.

4 Conclusions and Further Work

In this paper, we present a Petri-net-based approach utilizing property transition nets. The formal definitions and semantics of nets are discussed in detail. We separate the functions of transitions and places so that our model will be suitable to represent the workflow process. We believe our model can be easily handled by non-experts who merely understand the basic meaning of the property of place. This paper also shows that our model can express all static advanced routings. In the end, we present some new rules for soundness verification, and give an illustration that method of [11] fails to handle.

At present, we have developed a simulator and a small checker based on property transition nets. In the further work, we will use temporal logic to describe the expressions of arcs and the specifications of tasks so that many advance properties can be verified.

Acknowledgement

We appreciate Prof. C.Y. Yuan and three reviewers for some advices about this article. Also, we are grateful to all the students of theory lab and the center of national software engineering.
References


