Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 595897, 13 pages
http://dx.doi.org/10.1155/2013/595897

Research Article

A Model Reduction Method for Parallel Software Testing

Tao Sun and Xinming Ye

College of Computer Science, Inner Mongolia University, Hohhot 010021, China

Correspondence should be addressed to Tao Sun; cssunt@imu.edu.cn

Received 1 February 2013; Accepted 9 June 2013

Academic Editor: Guiming Luo

Modeling and testing for parallel software systems are very difficult, because the number of states and execution sequences expands significantly caused by parallel behaviors. In this paper, a model reduction method based on Coloured Petri Net (CPN) is shown, which could generate a functionality-equivalent and trace-equivalent model with smaller scale. Model-based testing for parallel software systems becomes much easier after the model is reduced by the reduction method. Specifically, a formal model for software system specification is constructed based on CPN. Then the places in the model are divided into input places, output places, and internal places; the transitions in the model are divided into input transitions, output transitions, and internal transitions. Internal places and internal transitions could be reduced if preconditions are matching, and some other operations should be done for functionality equivalence and trace equivalence. If the place and the transition are in a parallel structure, then many execution sequences will be removed from the state space. We have proved the equivalence and have analyzed the reduction effort, so that we could get the same testing result with much lower testing workload. Finally, some practices and a performance analysis show that the method is effective.

1. Introduction

Many software applications are parallel software systems. For example, most of cloud computing software systems contain parallel behaviors. However, parallel behaviors lead to the number of states and execution sequences expanding significantly. As a result, it is very difficult to validate correctness of this kind of software.

Software testing technology is the primary means of software correctness validation. Recently, the size of software is rising significantly, so model-based software testing technology becomes a focus in software testing area [1, 2].

However, model-based testing technology for parallel software systems has not been solved well. Many traditional testing technologies cannot work effectively for parallel systems with masses of states.

Many formal languages, like Finite State Machine (FSM), are not suitable for parallel software modeling. FSM describes system states and system messages directly, and it is very difficult to build an FSM model for a parallel system, because of its numerous states. Although testing methods based on FSM models have been studied in depth [3], there is almost no existing literature addresses on parallel system testing based on FSM. To solve this problem, utilizing Coloured Petri Net (CPN) models for software testing is a good choice. CPN is better than many other formal languages for parallel behaviors modeling. In CPN modeling, firing of transitions and moving of tokens directly express parallel behavior of the system.

Many formal languages, like CPN and Input Output Symbolic Transition System (IOSTS), could model parallel software; however, testing methods based on these languages could not work well because the number of states in the model is too large. In CPN modeling, state space diagram of the model can be calculated automatically. The CPN model of a parallel software system is usually simple, but the number of states in its state space diagram is still very large, so traditional testing methods on CPN could not work well because of the large number of states.

In this paper, a model reduction method based on CPN is shown, which could generate a function-equivalent and trace-equivalent model with smaller scale. The aim of the method is to get an external behavior equivalent model and remove many internal places and transitions from the model, cutting down the number of states, so that the number and the size of execution sequences become much smaller. So that model-based testing for parallel software systems becomes much easier after the model is reduced by the method.

Specifically, a formal model for parallel software system under testing is constructed by CPN tools, called system model. In the model, places which match with input and output ports are recorded, called input and output places, or visible places; other places are called internal places. Transitions which match with input and output behaviors are recorded, called input and output transitions, or visible transitions; other transitions are called internal transitions. Internal places and internal transitions could be reduced if preconditions are matching. In the process of reduction, the place and the transition will be removed from the model, and some arcs should be removed or redirected, while some expression or function should be modified, so that the reduced model is functionality equivalent and trace equivalent to the original model.

There are three basic structures in CPN models, including sequence structure, fork and joint structure, and parallel and synchronization structure. The method is effective for all of these structures. After reduction, the trace is equivalent with the original model; the fork joint structure and the parallel synchronization structure are reserved. Some internal places and internal transitions will be removed from the sequence structure model fragment or branches of the fork joint structure and the parallel synchronization structure model fragment. As a result, the number of states of the model is reduced largely, so the number and the size of execution fragments become much smaller. If the place and the transition being removed are in a parallel synchronization structure, then many execution sequences will be removed from the state space, so that we could get the same testing result with much lower testing workload. So the reduction method is particularly effective for parallel software testing.

The major contribution of this paper is to propose a CPN model-based reduction method for parallel software testing, which could reduce the model automatically. Model-based testing for a parallel software system becomes much easier after the model is reduced by the method. We have proved the equivalence and have analyzed the reduction effort, so that we could get the same testing result with much lower testing workload. The method is effective for all kinds of CPN models, including sequence structure, fork and joint structure, and parallel and synchronization structure.

The rest of this paper is organized as follows. Section 2 shows the related work. Section 3 will give some definition of key terms. Section 4 focuses on the model reduction algorithm based on trace-equivalence principle, including the description of the algorithm, the proof of the algorithm, examples, and effort analysis of the algorithm. Section 5 describes some practical applications of the algorithm, and we conclude the paper in last section.

2. Related Work

In recent years, there are many studies concerning model-based software testing technology [1, 2]. However, there are few notable studies about model-based testing for parallel software, because the number of states and execution sequences are very large caused by parallel behaviors. Many formal languages, like FSM, are not suitable for parallel software modeling. Although testing methods based on FSM models have been studied in depth [3], there is almost no existing literature that addresses parallel system testing based on FSM. Many formal languages could model parallel software; however, testing methods based on these languages could not work because the number of states in the model is too large. Overall, there are only a few literatures addressing testing methods for parallel system, and the testing effect in these literatures is not very good. For example, the literature [4] is based on activity diagram of UML and shows a method to test parallel behaviors about critical resource. However, this method ignores the interleaving path coverage between concurrent processes and only requires each process to cover every resource for once. The testing coverage is relatively low, so it is hard to get good testing results.

This paper argues model-based testing for parallel software based on CPN, because CPN is very suitable for parallel behaviors modeling. In some literatures, CPN is used in modeling and analysis of railway network control logic, and railway network is a typical parallel system.

There are some researches addressing testing based on CPN; however, few of them are for parallel software, so their testing effect for parallel software is not very good. Literature [5] gives a relatively simple test case generation method, which directly calculates the state reachable tree of system model and generates test sequences based on traversal of tree; literature [6] is based on a simple CPN model, constructs a causal relationship net which is made up of key transitions, and extracts test input and output to form a complete test case; literature [7] shows sequence coverage criteria (branch coverage, edge cover, etc.) and parallel coverage criteria (interleaving node or edge coverage, etc.), and test sequences are generated from random walk algorithm on the state space of CPN model. Overall, these methods are based on simply searching or traversal for the state space of CPN models. But state spaces of parallel software systems are often large, so these methods will generate many useless test sequences.

Modeling and testing for a parallel software system is very difficult for all kinds of modeling languages. CPN is suitable for parallel behavior modeling, but testing based on CPN is very difficult too. The reason is that although the CPN model of a parallel software system is usually with small scale, but its state space is also very big caused by parallel behaviors. To solve this problem, model reduction technology becomes breakthrough. Model reduction process generates an external behavior equivalent model with smaller scale, so the number of system states and execution sequences are reduced. There are some researches about reduction method of Petri Nets models, and there are some researches about reduction method of other formal modeling language [8, 9], but there are few studies about reduction method of CPN models. CPN is much more complex than Petri Nets, so reduction method of CPN models is much more difficult.

In this paper, a CPN model reduction approach for parallel software testing is proposed. In CPN-based testing for a parallel software system, many testing methods could not work because its state space is too big; if the system model is reduced by the reduction method of this paper, then some testing methods will work well. So a trace-equivalent reduction method is advantageous for model-based parallel software testing.

3. Preliminaries

This section presents some key concepts in the CPN model reduction approach for parallel software testing.

Definition 1. A Colored Petri Net is a nine-tuple CPN $= (P, T, A, \Sigma, V, C, G, E, I)$, where

(1) P is a finite set of places;

(2) T is a finite set of transitions such that $P \cap T = \Phi$;

(3) $A \subseteq P \times T \cup T \times P$ is a set of directed arcs;

(4) Σ is a finite set of nonempty color sets;

(5) V is a finite set of typed variables, $\text{Type}[v] \subseteq \Sigma$ for all variables $v \in V$;

(6) $C : P \rightarrow \Sigma$ is a color set function that assigns a color set to each place;

(7) $G : T \rightarrow \text{EXPR}_V$ is a guard function that assigns a guard to each transition t such that $\text{Type}[G(t)] = \text{Bool}$;

(8) $E : A \rightarrow \text{EXPR}_V$ is an arc expression function that assigns an arc expression to each arc a such that $\text{Type}[E(a)] = C(p)_{\text{MS}}$, where p is the place connected to the arc a;

(9) $I : P \rightarrow \text{EXPR}_\Phi$ is an initialization function that assigns an initialization expression to each place p such that $\text{Type}[I(p)] = C(p)_{\text{MS}}$.

In parallel software testing, the system is often divided into two parts: implementation under test (IUT) and simulating test environment. Ports between the two parts are called points of control and observation (PCOs), including input PCOs which send data to IUT and output PCOs which receive data from IUT.

Definition 2. In CPN models, input places are corresponding to input PCOs; output places are corresponding to output PCOs; other places are all internal places. Input transitions are successor of input places in the IUT part, so that firing behavior of input transitions will move tokens out of input places, representing input behaviors; output transitions are successor of output places in the simulating test environment part, so that firing behavior of output transitions will move tokens out of output places, representing output behaviors; other transitions are all internal transitions.

As shown in Figure 1, the system model called M is divided into two parts by a dotted line. The right part is IUT and the left part is simulating test environment. The place named a is an input place and its input transition is TA; the place named b is an output place and its output transition is TB. When TA is firing, tokens of a will moving into IUT; when TB is firing, tokens of b will moving out of IUT.

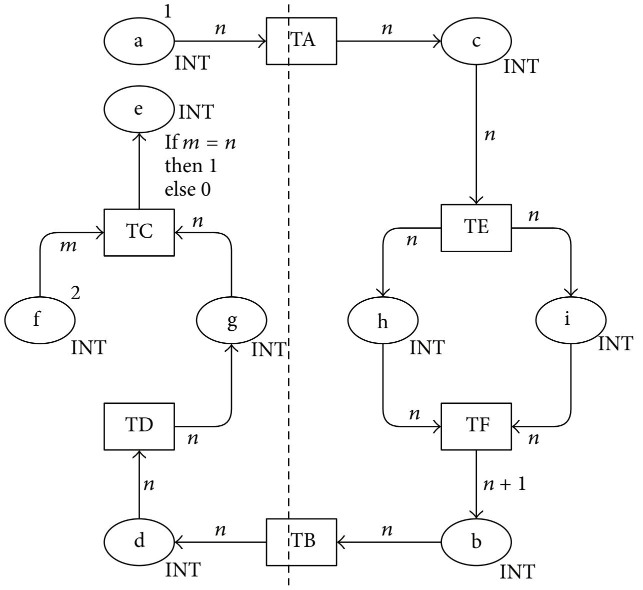


Figure 1: Example of system model.

Definition 3. A testing-oriented Coloured Petri Net (ToCPN) is a nine-tuple ToCPN $= (P, T, A, \Sigma, V, C, G, E, I)$, whose definitions are all same as CPN.

(1) P is partitioned into three sets $P = P\text{Input} \cup P\text{Output} \cup P\text{Internal}$, where PInput is the set of places representing input places, POutput is the set of places representing output places, PInternal is the set of places representing internal and invisible places, and $P\text{Input} \cup P\text{Output}$ is also called visible places.

(2) T is partitioned into three sets $T = T\text{Input} \cup T\text{Output} \cup T\text{Internal}$, where TInput is the set of transitions representing input actions, TOutput is the set of transitions representing output actions, TInternal is the set of transitions representing internal and invisible actions, and $T\text{Input} \cup T\text{Output}$ is also called visible transitions.

(3) An input place in PInput must be a predecessor of an input transition in TInput; an output place in POutput must be a predecessor of an output transition in TOutput.

CPN Tools is a tool for editing, simulating, and analyzing CPNs, which is originally developed by the CPN Group at Aarhus University from 2000 to 2010. A CPN model will be saved as an xml file by CPN tools. A ToCPN model is also a CPN model. However, six sets should be recorded additionally, including PInput, POutput, PInternal, TInput, TOutput, and TInternal. We can edit a model in CPN Tools and record these six sets in an additional file. Algorithms in this paper will act on the xml file of CPN tools and the addition file. ToCPN models of parallel software system are called system models.

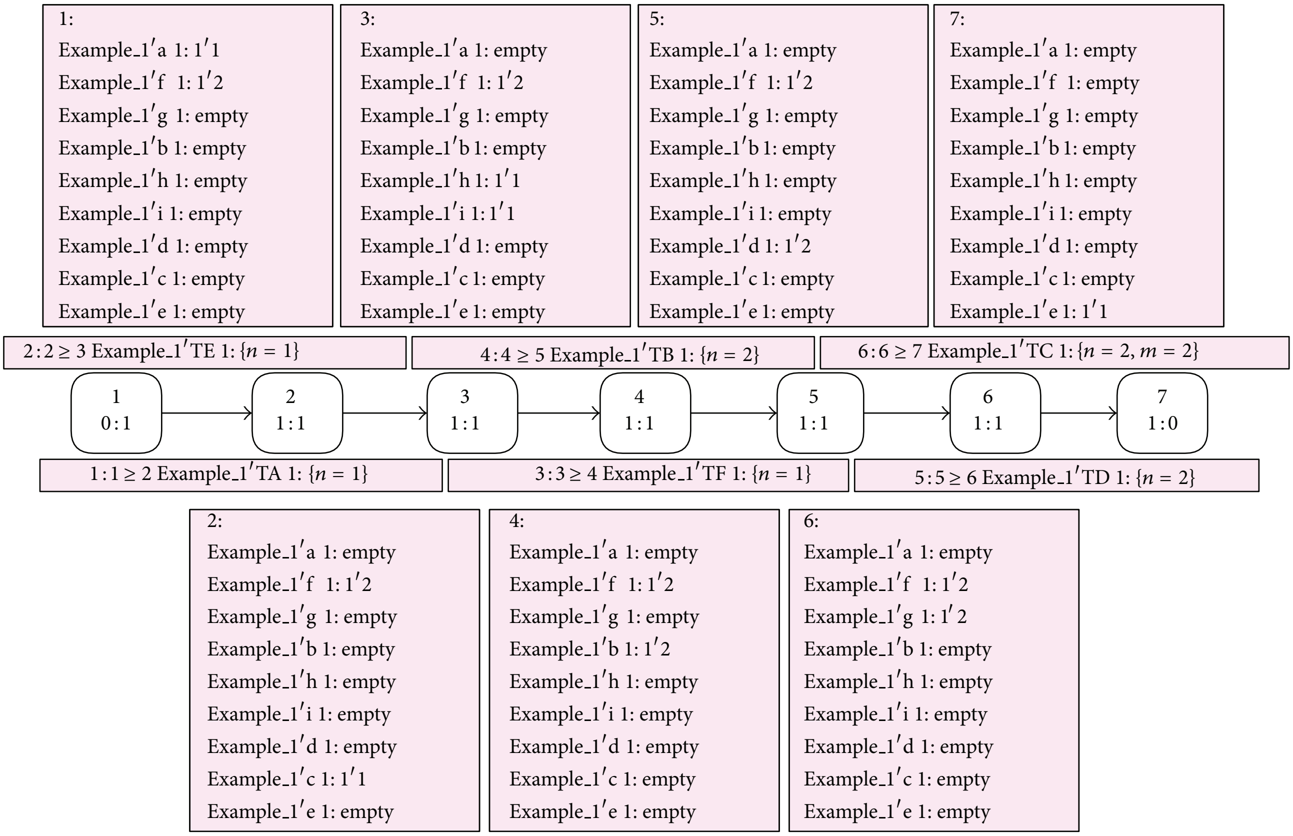


FIGURE 2: State space of model in Figure 1.

Definition 4. For a place $p \in P$, the set of predecessor transitions of p is called pred(p); the set of successor transitions of p is called succ(p). Similarly, for a transition $t \in T$, the set of predecessor places of t is called pred(t); the set of successor places of t is called succ(t).

As shown in Figure 1, pred(b) = {TF}, succ(b) = {TB}, pred(TF) = {h, i}, succ(TF) = {b}.

Definition 5. A marking of CPN model is a function marking that maps each place $p \in P$ into a multiset of tokens Marking(p) $\in C(p)_{\mathrm{MS}}$. Marking describes tokens distribution in all places at a time, that is, the system state at a time.

Definition 6. A transition firing behavior removes tokens from its predecessor places and adds tokens to its successor places. When a firing behavior t' occurs in a marking M_1, producing a new marking M_2, we say that the marking M_2 is directly reachable from M_1 by the firing behavior t'.

Definition 7. The state space of a CPN model is a directed graph with a node for each reachable marking and an arc for each occurring firing behavior.

The state space of model in Figure 1 is shown in Figure 2. Nodes named 1 to 7 are all of markings of the model; arcs named 1 to 6 are all of transition firing behaviors of the model. Information of nodes and arcs is shown in rectangular boxes. For example, in marking 1, only place a and place f have tokens; arc1 represents the firing of TA with variable $n = 1$, leading to marking 1 transferring to marking 2.

Definition 8. An execution fragment of a model is a sequence of alternating markings and firing behaviors. So any sequence in the directed graph of state space is an execution fragment. An execution is an execution fragment starting in an initial marking.

In a state space diagram, if we denote nodes as M, denote transitions as t, and denote transition firing behaviors as t', then all executions and execution fragments appear as form $M(t'M)^*$. All executions are in state space of the model.

As shown in Figure 2, there is only one execution in the state space as follows:

$$p = M_1 \mathrm{TA}' M_2 \mathrm{TE}' M_3 \mathrm{TF}' M_4 \mathrm{TB}' M_5 \mathrm{TD}' M_6 \mathrm{TC}' M_7. \quad (1)$$

Definition 9. The transition firing behavior sequence of an execution p is the projection of p on the set of transition firing behaviors, called transition(p).

As shown in Figure 2, transition(p) = TA′TE′TF′TB′ TD′TC′.

Definition 10. A transition firing behavior is an input firing behavior if its transition is in TInput, and it leads to token-losing of places in PInput. A firing behavior is an output firing

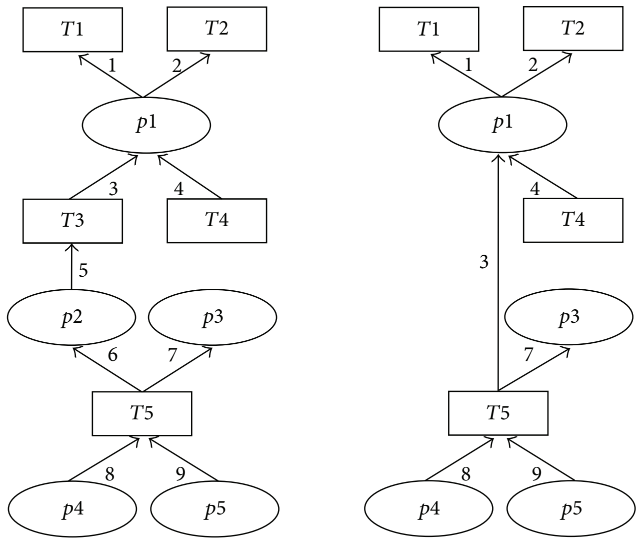


Figure 3: Effect of the reduction algorithm.

behavior if its transition is in TOutput, and it leads to token-losing of places in POutput. Input firing behaviors and output firing behaviors are visible firing behaviors, and other firing behaviors are invisible firing behaviors.

As shown in Figure 1, the firing of TA and the firing of TB are visible firing behaviors; other firings are invisible firing behaviors.

Definition 11. The trace of an execution p is the projection of p on the set of visible firing behaviors, called trace(p). The set of traces of a CPN model M is the set of all traces of all executions of M and is denoted by traces(M).

As shown in Figure 2, trace(p) = $\mathrm{TA}'\mathrm{TB}'$. There is only one execution in the state space, so traces(M) = $\{\mathrm{TA}'\mathrm{TB}'\}$.

4. Model Reduction Method

In model-based parallel software testing, input and output firing behaviors of the IUT are visible firing behaviors, and other firing behaviors are all internal transitions, which are invisible firing behaviors. So only visible firing behaviors should be cared about when the software system is tested. That is, we should input data to IUT according to input firing behaviors and compare whether outputs of IUT are same as output firing behaviors.

When an execution p is tested, the trace(p) represents all input and output information flow of p, so the trace of an execution is the key content of software testing. In the example, input and output information of $\mathrm{TA}'\mathrm{TB}'$ is the tested content when execution $M_1\mathrm{TA}'M_2\mathrm{TE}'M_3\mathrm{TF}'M_4$ $\mathrm{TB}'M_5\mathrm{TD}'M_6\mathrm{TC}'M_7$ is tested.

In CPN-based software testing, executions of the model are sequences to be tested. If we could give testing sequences by any generation method, like depth first search algorithm, then traces of these testing sequences are content of testing. However, for parallel software testing, the number of states and executions are very large, so it is very difficult to get and test all executions.

In this section, a CPN model reduction method based on trace-equivalence principle is shown and applied on system model M. The aim of the method is to get an external behavior equivalent model RM with smaller scale. Specifically, RM is gained from M by many internal places and internal transitions are removed, and traces(RM) = traces(M). So the number of states and executions will be cut down, and we could complete the testing process with lower workload. The testing result on RM is same as testing result on M, because traces(RM) = traces(M). Before parallel software testing, we could apply this model reduction method on system model, and then the testing workload will be cut down and testing result will be unchanged. In other words, testing for parallel software systems becomes much easier after the model is reduced by the method.

In a CPN model, there are three kinds of structures that should be discussed, including sequence structure, fork and joint structure, and parallel and synchronization structure. If places and transitions are single-input and single-output, it is sequence structure; if a place is multioutput, it is fork structure; if a place is multi-input, it is fork and joint structure; if a transition is multioutput, it is concurrency structure; if a transition is multi-input, it is parallel and synchronization structure. These structures should be discussed when reduction.

When we reduce the model, the following four principles should be complied.

(i) The connectivity of the model should be conformed after reduction, so some arcs should be redirected.

(ii) The executability of the model should be conformed after reduction, so variables of some arcs should be changed.

(iii) Functions of places and transitions which are removed should be reserved in the model, so these functions should be added to other places and transitions which are reserved. Some internal places or transitions should not be removed. For example, fork points, joint points, parallel points, synchronization points, and transitions with guard should be reserved; otherwise, some behaviors of the model may be changed.

(iv) After reduction, traces of the model should be equivalent to the original model, and many internal places and transitions should be removed, so the size of state space should be cut down, and the number of states and executions should be decreased.

In this section, the reduction algorithm and the proof of the algorithm will be shown; some examples of the algorithm will be given; effort analysis of the algorithm for three structures will be discussed too.

4.1. Reduction Algorithm and Proof of the Algorithm. As shown in Figure 3, the left part is a model fragment, and the right part is the reduction result. We will reduce the model fragment complying with the four previous principles. The aim of the reduction is that internal places and transitions are removing, and the state space of the model is reduced the number of states and executions are decreased, but traces of the model are same as the original model. The algorithm is shown in Algorithm 12.

Algorithm 12. Reduction algorithm based on trace-equivalence.

Symbols. In the model fragment as shown in left part of Figure 3, suppose that

places are called $p1, p2, \ldots, p5$;

transitions are called $T1, T2, \ldots, T5$;

arcs are called arc1, arc2, . . . , arc9.

Variables of arc$x$ are called $v(x)$, and arc expression of arc$x$ is defined as function called $Fx(v(x))$, where x is a number between 1 and 9.

Variables of transition Tx are called $v(Tx)$, and guard of transition Tx is defined as function called $GTx()$,where x is a number between 1 and 5.

Initialization of place px is defined as function called $Ipx()$, where x is number between 1 and 5.

Preconditions. $T3$ is an internal transition and $p2$ is an internal place; $T3$ and $p2$ are single-input and single-output; guard of $T3$ is empty.

Step 1. Transition $T3$, place $p2$, arc5, and arc6 should be removed; arc3 should start from $T5$.

Step 2. Operation $v'(3) = v(6)$ should be done, where symbol "′" represents the new version after reduction algorithm. In other words, variables of arc3 should be changed to variables of arc6 after reduction.

Step 3. Two operations should be done, as follows.

(1) Arc expression of arc3 should be changed as follows:

$$\begin{aligned} F3'\left(v'(3)\right) &= F3'(v(6)) \\ &= F3(v(3) \mid F5(v(5)) := F6(v(6))); \end{aligned} \tag{2}$$

(2) initialization of place $p2$ should be changed as follows:

$$Ip1'() = Ip1() + +F3\left(v(3) \mid F5(v(5)) := Ip2()\right). \tag{3}$$

Operator "|"represent that variables in left operand and the same variables in right operand should have the same value; operator ":="represent that left operand and right operand should have the same value.

Theorem 13. *Reduction algorithm in Algorithm 12 is trace-equivalent.*

In other words, if CPN model M is reduced by Algorithm 12, and RM is the result, then traces(M) = traces(RM).

If a fragment of the model matches the precondition of Algorithm 12, then the algorithm will work, until no fragment in the model matches the precondition. In other words, the algorithm may be executed for many times, and many internal places and transitions in M are removed.

Proof of Theorem 13. There are four properties that should be proved: the connectivity of RM; the executability of RM; the functionality-equivalence of M and RM; the trace-equivalence of M and RM.

(1) Proof of the Connectivity. The connectivity is the precondition of other properties.

In the reduction algorithm, an internal transition $T3$ and an internal place $p2$ will be removed. $T3$ and $p2$ are single-input and single-output, specifically, $\text{pred}(p2) = \{T5\}$, $\text{succ}(p2) = \{T3\}$, $\text{pred}(T3) = \{p2\}$, $\text{succ}(T3) = \{p1\}$.

In Step 1, the transition $T3$, the place $p2$, the arc5, and the arc6 are removed, and the arc3 is changed to be from $T5$ to $p1$. So the connectivity of RM is conformed.

(2) Proof of the Executability. In Step 1, the start of the arc3 is changed from $T3$ to $T5$, so the arc3 will take part in the firing of $T5$.

In Step 2, $v'(3) = v(6)$, variables of the arc3 in RM are same as variables of the arc6 in M, so that $T5$ could be banding, enabling, and firing in RM. So the executability of RM is conformed.

(3) Proof of the Functionality Equivalence of RM and M. In Step 1, the place $p2$ is removed, so tokens of the $p2$ should be further processed, including two aspects: tokens gaining from the firing of $T5$; initialization tokens of $p2$. So in the proof of the functionality equivalence, there are two properties that should be proved: tokens gaining from the firing of $T5$ are equivalent; initialization tokens of $p2$ are equivalent.

(a) Proof of tokens gained from the firing of $T5$ are equivalent.

In model $M, T3$ and $p2$ are single-input and single-output, and the guard of $T3$ is empty, so tokens gain from the firing of $T5$ will move to $p1$ by three steps.

(i) Tokens will move into the place $p2$ when $T5$ is firing, according to expressions of arc6.

(ii) Tokens will move out of the place $p2$ when $T3$ is firing, according to expressions of arc5.

(iii) Tokens will move into the place $p1$ when $T3$ is firing, according to expressions of arc3.

In model RM, the place $p1$ is successor of the transition $T5$ by the arc3. In Step 3, arc expression of the arc3 is changed as $F3'(v'(3)) = F3'(v(6)) = F3(v(3) \mid F5(v(5)) := F6(v(6)))$. When $T5$ is firing, $F3'(v'(3))$ is calculated as follows.

(i) $F6(v(6))$ is calculated, and the result is the set of tokens moving into the place $p2$ in M.

(ii) $F5(v(5)) := F6(v(6))$ is calculated, and the result is the set of tokens moving out of the place $p2$ in M.

(iii) $F3(v(3) \mid F5(v(5)) := F6(v(6)))$ is calculated, and the result is the set of tokens moving into $p1$ in M.

So $F3'(v'(3)) = F3(v(3) \mid F5(v(5)) := F6(v(6)))$ makes that when $T5$ is firing in RM, the result is same as expressions of arc6, arc5, and arc3 are all calculated in M. So tokens gained from the firing of $T5$ are equivalent.

(b) Proof of initialization tokens of $p2$ is equivalent.

In model $M, T3$ and $p2$ are single-input and single-output, and the guard of $T3$ is empty, so initialization tokens of $p2$ will move to $p1$ by two steps.

(i) Tokens will move out of the place $p2$ when $T3$ is firing, according to expressions of arc5.

(ii) Tokens will move into the place $p1$ when $T3$ is firing, according to expressions of arc3.

In model RM, initialization of place $p1$ should be changed as $Ip1'() = Ip1() + +F3(v(3) \mid F5(v(5)) := Ip2())$. $Ip1'()$ is calculated as follows.

(i) $F5(v(5)) := Ip2()$ is calculated, and the result is the set of initialization tokens moving out of $p2$ in M.

(ii) $F3(v(3) \mid F5(v(5)) := Ip2())$ is calculated, and the result is the set of tokens moving into $p1$ in M.

So $Ip1'() = Ip1() + +F3(v(3) \mid F5(v(5)) := Ip2())$ makes that initialization tokens of $p1$ in RM are same as initialization tokens of $p1$ and tokens of $p1$ gaining from initialization tokens of $p2$ in M. So initialization tokens of $p2$ are equivalent.

(4) Proof of the Trace Equivalence of RM and M. In model M, if an execution E_1 contains a firing of $T3$, then a firing of $T5$ is a must in the execution and before the firing of $T3$. Suppose that E_1 is as follows:

$$\begin{aligned} E_1 &= M_1T_1\cdots M_pT_pM_{p+1}T_{p+1}\cdots M_{p+n-1} \\ &\quad\times T_{p+n-1}M_{p+n}T_{p+n}M_{p+n+1}T_{p+n+1}\cdots, \\ \text{Transition}\left(E_1\right) &= T_1\cdots T_pT_{p+1}\cdots T_{p+n-1}T_{p+n}T_{p+n+1}\cdots. \end{aligned} \tag{4}$$

In E_1, p, n, and e are positive integers, $p \geq 1$, $n \geq 1$, $e \geq p + n + 1$; the firing of $T3$ is T_{p+n}, and the firing of $T5$ is T_p. After T_p, $T3$ is enabling, and some other transitions may be enabling too. So there may be firings of other transitions between T_p and T_{p+n}, which are $T_{p+1}\ldots T_{p+n-1}$.

In model RM, $T5$ moves tokens into place $p1$ directly, so token gain of $p1$ in M_{p+n+1} will be acting in M_{p+1}, which will not change the enabling and firing of $T_{p+1}\ldots T_{p+n-1}$. The reason is $T3$ and $p2$ are single-input and single-output. So the execution E_1 will be changed as follows:

$$\text{Transition}\left(E_1'\right) = T_1\cdots T_pT_{p+1}\cdots T_{p+n-1}T_{p+n+1}\cdots. \tag{5}$$

The number of places in model RM is less than the number of places in model M, so all markings of RM are different from M, but firings of transitions are the same. T_{p+n} is removed from E_1, and other firings of transitions exist in the original order. $T3$ is internal transition, so trace(E_1') = trace$(E1)$.

So trances(RM) = trances(M), and the trace equivalence of RM and M is conformed. □

In model M, there may be several executions that are similar. In these executions, only the position of T_{p+n} is different from each other. After reduction, these executions are reduced into an execution. For example, in the following sequences, T_p is a firing of $T5$, and T_{p+3} is a firing of $T3$:

$$\begin{aligned} \text{Transition}\left(E_1\right) &= T_1\cdots T_pT_{p+1}T_{p+2}T_{p+3}\cdots, \\ \text{Transition}\left(E_2\right) &= T_1\cdots T_pT_{p+1}T_{p+3}T_{p+2}\cdots, \\ \text{Transition}\left(E_3\right) &= T_1\cdots T_pT_{p+3}T_{p+1}T_{p+2}\cdots. \end{aligned} \tag{6}$$

After reduction algorithm,

$$\begin{aligned} \text{Transition}\left(E_1'\right) &= \text{Transition}\left(E_2'\right) = \text{Transition}\left(E_3'\right) \\ &= T_1\cdots T_pT_{p+1}T_{p+2}\cdots. \end{aligned}$$

So the reduction algorithm makes all markings of the model become smaller; makes some markings be not in the state space; makes some executions become shorter; especially makes some executions be not in the state space.

4.2. Examples of the Reduction Algorithm. In this section, two simple examples will show the execution process of the algorithm and the effect of the algorithm on the state space.

4.2.1. Example to Show the Execution Process of the Algorithm. In the model fragment of Figure 3 left, take the calculation of $F3'(v'(3))$, for example, as follows.

Precondition. Consider

$$\begin{gathered} v(6) = \{a\}, \quad v(5) = \{b\}, \quad v(3) = \{b\}, \\ F6(v(6)) = F6(\{a\}) = \text{if } (a > 0)\, a^*2 \text{ else } a^*10, \\ F5(v(5)) = F5(\{b\}) = b, \\ F3(v(3)) = F3(\{b\}) = \text{if } (b > 5)\, b^*2 \text{ else } b^*10. \end{gathered} \tag{7}$$

Calculation of $F3'(v'(3))$. Consider

$$\begin{aligned} v'(3) &= v(6) = \{a\}, \\ F3'\left(v'(3)\right) &= F3(v(3) \mid F5(v(5)) := F6(v(6))) \\ &= F3(\{b\} \mid F5(\{b\}) := F6(\{a\})) \\ &= F3(\{b\} \mid F5(\{b\}) := \text{if } (a > 0) \\ &\qquad a^*2 \text{ else } a^*10) \\ &= F3\left(\{b\} \mid b := \text{if } (a > 0)\, a^*2 \text{ else } a^*10\right) \\ &= \text{if } (b > 5)\, b^*2 \text{ else } b^*10, \end{aligned} \tag{8}$$

where $b := \text{if } (a > 0) a^*2 \text{ else } a^*10$.

Result of $F3'(v'(3))$. Consider

$$F3'\left(v'(3)\right) = F3'(\{a\}) = \text{if } (b > 5)\ b^*2 \text{ else } b^*10, \tag{9}$$

where b:= if $(a > 0)\, a^*2$ else a^*10.

So in model RM, if $T5$ is firing with $a = 10$, then $p1$ will get 1000, and the result is same as model M. Variables of arc3

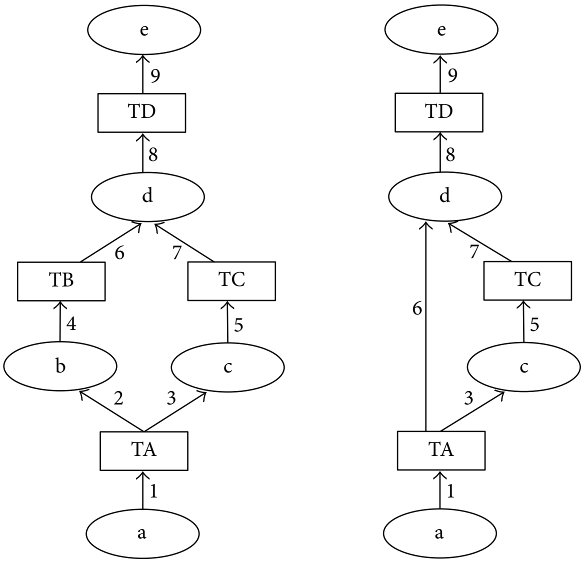


FIGURE 4: The original model and the reduced model.

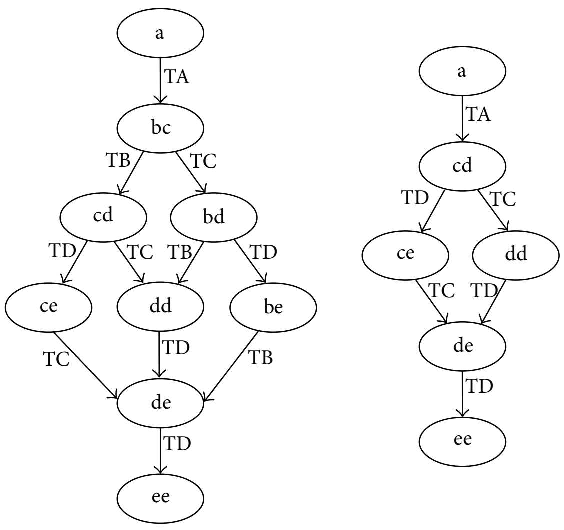


FIGURE 5: The state space diagrams of models in Figure 4.

changed from $\{b\}$ to $\{a\}$. Both $\{a\}$ and $\{b\}$ are in the function of $F3'(\{a\})$; however, $\{a\}$ is the real variable of the arc, and $\{b\}$ is the temporary local variable of the function which is not variable of arc for binding when transition firing.

The reduction algorithm could remove the internal place and internal transition. After reduction, the connectivity and executability of RM and the functionality equivalence of RM and M are conformed.

4.2.2. Example to Show the Effect of the Algorithm on the State Space. As shown in Figure 4, a simple software model M is on the left of Figure 4, and the model after reduction is called RM which is on the right of Figure 4. The transition TB is internal transition; the place b is internal place; the guard of TB is empty. After reduction, TB and b and arc4 and arc2 are removed.

As shown in Figure 5, the state space diagram of M is on the left, and the state space diagram of RM is on the right. In Figure 5, a node represents a marking; letters in a node represent names of places with token at the marking; letters on an arc represent the firing transition between two markings. For example, the firing of transition TA transforms the marking a into the marking b; in the marking a, only the place a has token; in the marking bc, place b and place c have tokens.

There are 4 executions in the state space of M as follows:

$$\begin{aligned} &\text{Transition}\left(E_1\right) = \text{TA}'\text{TB}'\text{TD}'\text{TC}'\text{TD}', \\ &\text{Transition}\left(E_2\right) = \text{TA}'\text{TB}'\text{TC}'\text{TD}'\text{TD}', \\ &\text{Transition}\left(E_3\right) = \text{TA}'\text{TC}'\text{TB}'\text{TD}'\text{TD}', \\ &\text{Transition}\left(E_4\right) = \text{TA}'\text{TC}'\text{TD}'\text{TB}'\text{TD}'. \end{aligned} \tag{10}$$

There are 2 executions in the state space of RM as follows:

$$\begin{aligned} &\text{Transition}\left(E_1'\right) = \text{TA}'\text{TD}'\text{TC}'\text{TD}', \\ &\text{Transition}\left(E_2'\right) = \text{TA}'\text{TC}'\text{TD}'\text{TD}'. \end{aligned} \tag{11}$$

The transition TB is internal transition, so trace(E_1) = trace(E_1'), trace(E_2) = trace(E_2'), trace(E_3) = trace(E_2), trace(E_4) = trace(E_2'), and traces(M) = traces(RM).

4.3. Effect Analysis of the Reduction Algorithm. In a CPN model, there are three kinds of basic structures that should be discussed, including sequence structure, fork and joint structure, and parallel and synchronization structure. If places and transitions are single-input and single-output, it is sequence structure; if a place is multioutput, it is fork structure; if a place is multi-input, it is joint structure; if a transition is multioutput, it is concurrency structure; if a transition is multi-input, it is synchronization structure.

The effects of the reduction algorithm used in all three kinds of basic structures are good.

(1) Reduction of Sequence Structure. As shown in Figure 6, a sequence structure model fragment is on the left. The transition TB is internal transition; the place c is internal place; the guard of TB is empty. All preconditions of the algorithm are matching, so the sequence structure model fragment could be reduced by the algorithm and the result is on the right on Figure 6.

If the sequence structure model fragment is not in a branch of a parallel structure, and then some executions in the state space will become shorter, but the number of executions will be unchanged; if the sequence structure model fragment is in a branch of a parallel structure, then some executions in the state space will become shorter, and the number of executions will be smaller, like the example in Figure 4.

So the algorithm is effective for sequence structure model fragment, and the effect of the reduction is determined by whether the model fragment is in a parallel structure or

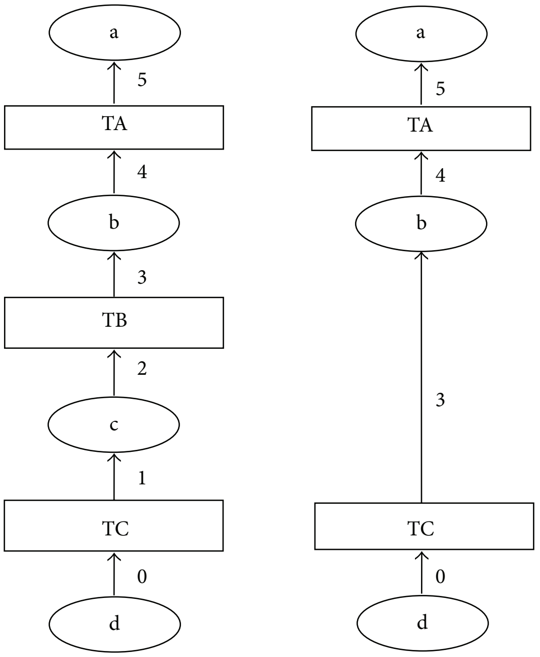


Figure 6: Reduction of sequence structure model fragment.

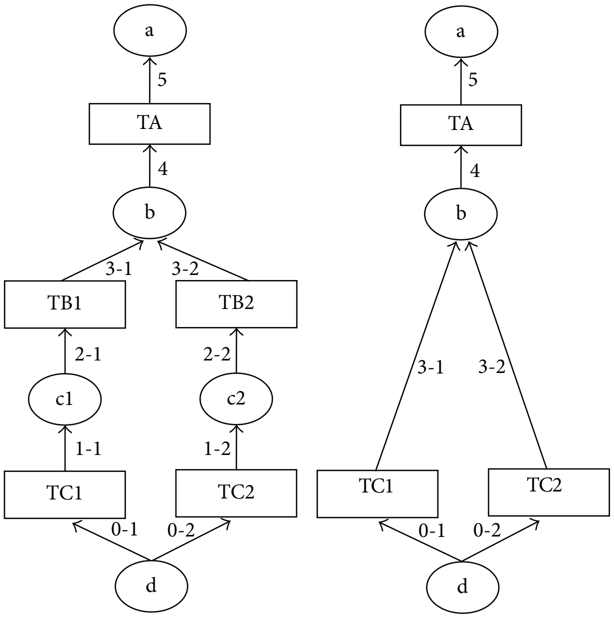


Figure 8: Reduction of fork and joint structure model fragment.

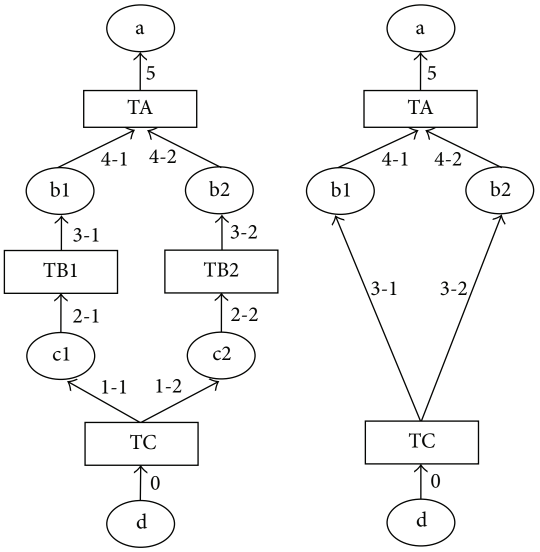


Figure 7: Reduction of parallel and synchronization structure model fragment.

not. The algorithm is especially effective for parallel structure model.

(2) Reduction of Parallel and Synchronization Structure. As shown in Figure 7, a parallel and synchronization structure model fragment is shown on the left. TC is multioutput and TA is multi-input; TB1, TB2, c1, and c2 are all internal; guards of TB1 and TB2 are empty. Each of the two branches matches the precondition of the algorithm, and two branches could be reduced, respectively. The reduction result is shown on the right of Figure 7.

If visible places and transitions exist in a branch, then visible places and transitions will be reserved after reduced; if there is no visible place or transition in a branch, then only a single place will be reserved in the branch after reduced, as shown on the right of Figure 7.

The internal transitions and places in a parallel and synchronization structure model fragment are removed, then some executions in the state space will become shorter, and the number of executions will be smaller, like the example in Figure 4. Particularly, if there is no visible place or transition in a branch, then only a single place will be reserved in the branch after reduction, as shown on the right of Figure 7, and then the number of executions in the model will be 1. In other words, the parallel behaviors are invisible outside, so we could reduce it to be happening at the same time, when TC is firing.

So the algorithm is especially effective for parallel and synchronization structure model fragment, especially many internal transitions and places matching the precondition of the algorithm being in a branch of the parallel and synchronization structure model.

(3) Reduction of Fork and Joint Structure. As shown in Figure 8, a fork and joint structure model fragment is shown on the left. The place d is multioutput and b is multi-input; TB1, TB2, c1, and c2 are all internal; guards of TB1 and TB2 are empty. Each of the two branches matches the precondition of the algorithm, and two branches could be reduced, respectively. The reduction result is shown on the right of Figure 8.

If visible places and transitions exist in a branch, then visible places and transitions will be reserved after reduced; if there is no visible place or transition in a branch, then only a single transition will be reserved in the branch after reduced, as shown on the right of Figure 7.

The internal transitions and places in a fork and joint structure model fragment are removed, and then some executions in the state space will become shorter. If the sequence structure model fragment is in a branch of a parallel structure, then some executions in the state space will become shorter, and the number of executions will be smaller, like the example in Figure 4.

So the algorithm is effective for fork and joint structure model fragment, and the effect of the reduction is determined by whether the model fragment is in a parallel structure or not. The algorithm is especially effective for parallel structure model.

Three kinds of basic structures have been discussed, the algorithm could be used in all of three situations, and its effect on reduction is good. The reduction algorithm makes all markings of the model become smaller; makes some markings be not in the state space; makes some executions become shorter; especially makes some executions be removed from the state space, when the reduction is in a parallel structure model. So the reduction algorithm is especially effective for parallel software system testing.

5. Practical Applications of the Algorithm

In practice, a software tool has been built based on the reduction algorithm. When the CPN model of the system under testing is given, the tool could give a reduced model automatically.

Several systems are tested based on the original model and the reduced model, and practices show that testing based on the reduced model is much more efficient.

In this section, a simple introduction of the tool is given, and a testing performance contrast analysis is given, and a BitTorrent (BT) software practical application of the algorithm is shown.

5.1. Practices of the Algorithm. The processing of model reduction and software testing is as follows.

(i) Before testing, the system model is built by CPN tools, and all information of the model is saved as an xml file by CPN tools.

(ii) Our software tool reads information from the xml file and reconstructs all information into its own data structure.

(iii) The tool will show all places and transitions of the model in text areas, and user should divide the set of PInput, POutput, PInternal, TInput, TOutput, and TInternal. And the six sets are recorded in another additional xml file.

(iv) The tool will traverse the model automatically, to judge whether a transition and its predecessor place are matching preconditions of the algorithm; if true, the algorithm will be performed to the transition and the place.

(v) Then the reduced model is calculated automatically, and the state space diagram of the model is calculated too.

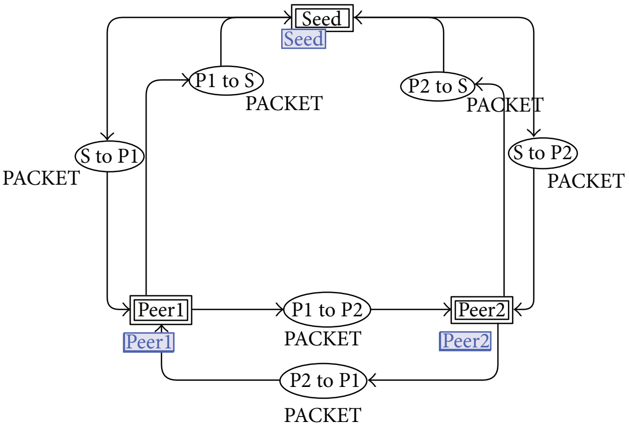


FIGURE 9: The top page of the BT software system.

(vi) The testing sequence generation algorithm is executed on the state space diagram of the reduced model. In the tool, the depth first algorithm is used as the testing sequence generation algorithm.

The tool could give the testing sequences automatically, and every testing sequence is an execution in the reduced model. We could test the system according to the input and output information in the testing sequence.

5.2. Testing Performance Analysis. For parallel software testing, the reduction algorithm is very effective, because the number of executions will be greatly reduced, and the traces are equivalent. In other words, we could get the same test effort with much lower workload.

Supposing that only one internal transition called TT is reduced, and TT is in a parallel structure, when TT is enabling, there are m transitions that are enabling too, where m is a positive integer. If m transitions are enabling in turn, then m executions will be reduced by the reduction of TT; if m transitions are enabling at the same time, then $m!$ executions will be reduced by the reduction of TT. In practice, two types of cases usually mixed, and then the effort is between m and $m!$.

For one reduced transition, the more parallel behaviors occurring at the same time, the more executions being reduced by the reduction. For the system model, the more transitions are reduced, the more executions will be reduced. The effort of reduction is determined by the following two factors:

(i) the number of transitions and places which match preconditions of the algorithm;

(ii) the number of transitions which are enabled at the same time of the reduced transition is enabled.

In practice of parallel system testing, the two numbers are always big, so the effort of reduction is very good.

5.3. A BT Software Practical Application of the Algorithm. BT protocol is a $p2p$ file transfer protocol, which is typical

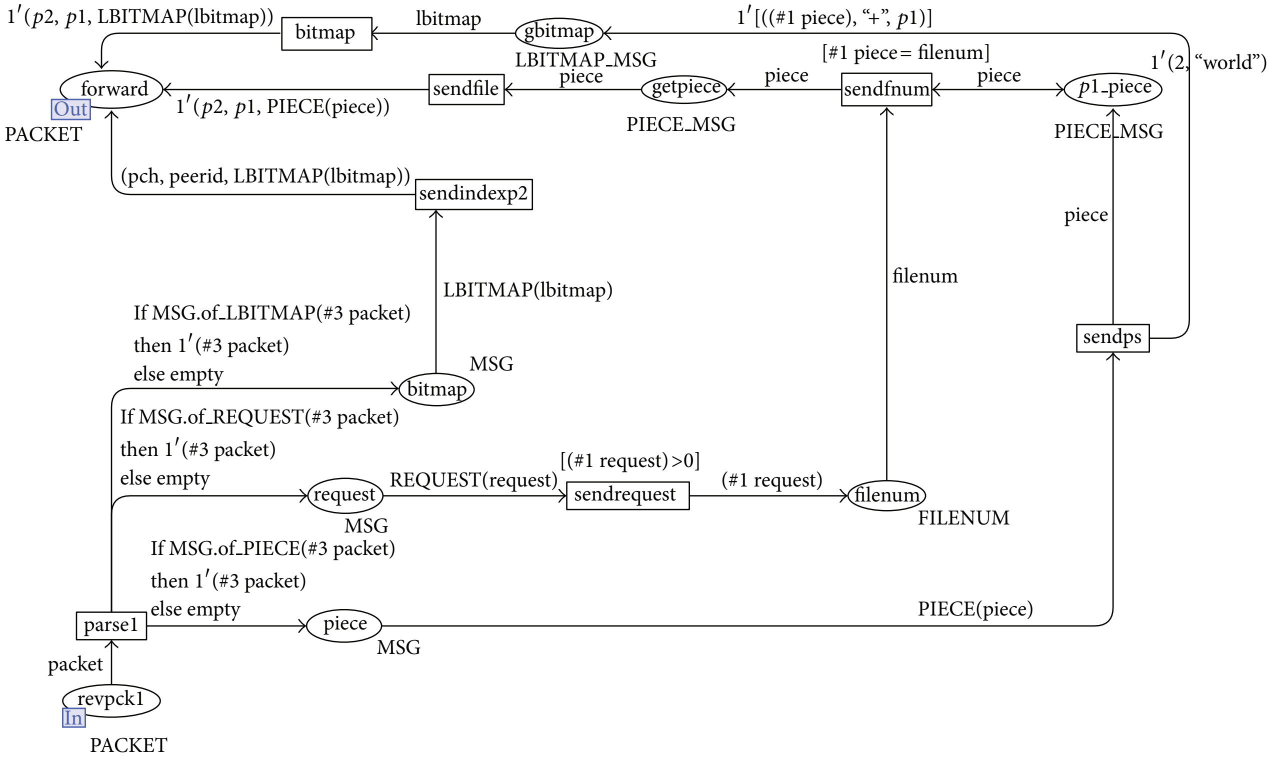


Figure 10: The receive page of a client in BT software system.

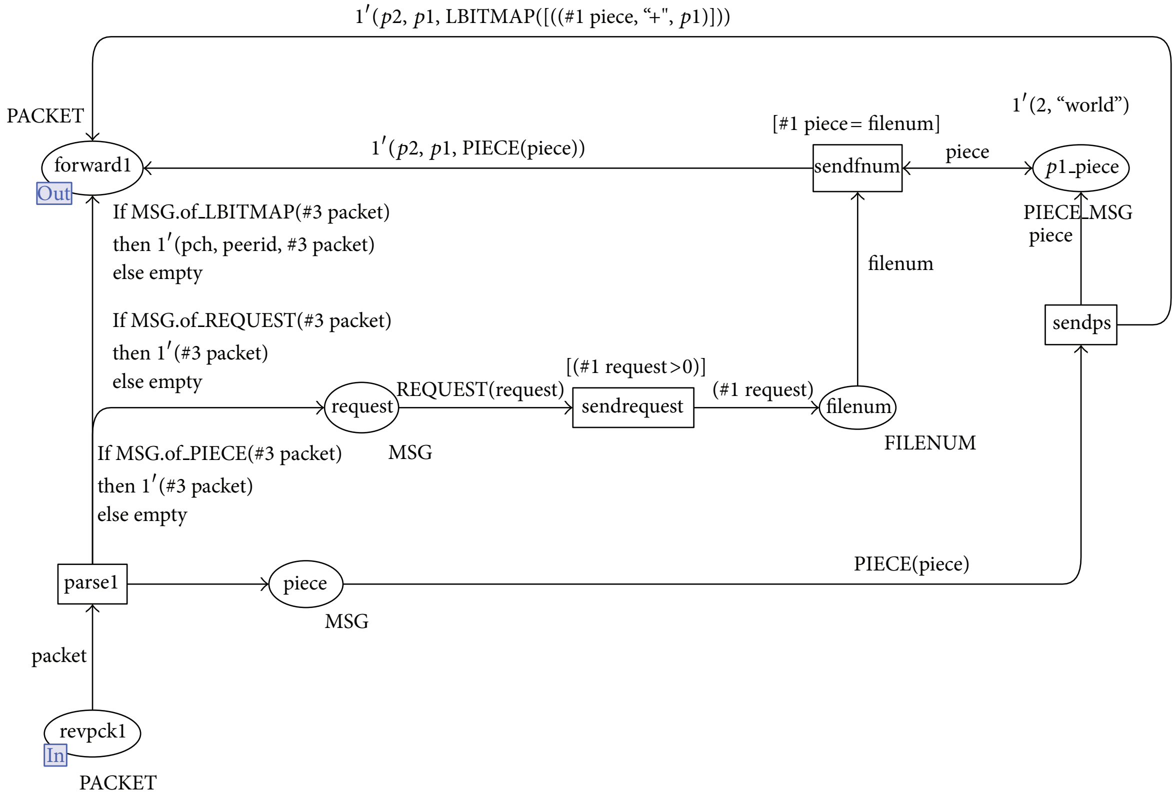


Figure 11: The receive page after reduction.

Table 1: The reduction results of every page.

	Peer1	Peer1-send	Peer1-receive	Peer2	Peer2-send	Peer2-receive
Places reduced	2	1	3	2	1	3
Transition reducted	2	1	3	2	1	3

Table 2: The comparison of state reports before and after reduction.

	Number of markings in state space	Number of arcs in state space	Status of state space
Before reduction	3409	9414	Full
After reduction	1329	3620	Full
Markings reduced		2080	
Arcs reduced		5794	
Markings reduction rate		About 61%	
Arcs reduction rate		About 61.5%	

parallel system. In the protocol, many clients download files from each other. A software system based on BT protocol is built, which is under testing.

First of all, a hierarchical CPN model based on the requirement specification of the software is built. There are 3 layers and 7 pages in the model. For example, the top page of the software system is shown in Figure 9, and the receive page of a client in BT software system is shown in Figure 10.

Secondly, a test sequence generation method based on the state space diagram of the model will be used [10], by which test sequences related to a test purpose will be generated automatically. However, the scale of state space diagram for a parallel system is always big, so that the number and size of testing sequences will be big too, and it is hard to get good testing results.

The reduction algorithm is used for the model. Many places and transitions and arcs are removed from the model, which are matching preconditions of the algorithm. For example, the receive page after reduction is shown in Figure 11. Transition bitmap and sendfile and sendindexp2 have been removed; place gbitmap and getpiece and bitmap have been removed; several arcs have been removed too. Other transitions and places could not match preconditions of the algorithm.

Several model elements are reduced in these model pages except for the top page. Table 1 shows the reduction results of every page, and Table 2 shows the comparison of state reports before and after reduction.

Behaviors in the model are parallel with each other. As a result, the number of model-elements being reduced is not big, but the number of states is reduced very much. In other words, the effect of reduction is good.

After reduction, the scale of state space diagram is reduced, so that the number and size of testing sequences will be small, and it is easy to get good testing results. We have proved that traces of the reduced model are equivalent to the original model, so we could get the same testing result with much lower testing workload.

6. Conclusion

Model-based testing of parallel software systems plays a significant role in software testing, which is quite difficult, because the number of executions in the system is too big. Many traditional testing methods cannot work effectively for this kind of software.

In this paper, a CPN model reduction method based on trace-equivalence principle is shown and applied on system model. It could remove many internal transitions and places and cut down the number of executions. The method is effective for all kinds of CPN models, including sequence structure, fork and joint structure, and parallel and synchronization structure. The method makes all markings of the model become smaller; makes some markings be not in the state space; makes some executions become shorter; especially makes some executions be removed from the state space, when the reduction is in a parallel structure model. So the reduction algorithm is especially effective for parallel software system testing.

We have proved that traces of the reduced model are equivalent to the original model, and the number of executions is significantly reduced after reduction when the reduction is in a parallel structure model fragment, and some practices and a performance analysis have shown the effort of reduction. So model-based testing for a parallel software system becomes much easier after the model is reduced by the method.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant no. 61163011; the National Natural Science Foundation of China under Grant no. 61262017; the Natural Science Foundation of Inner Mongolia of China under Grant no. 2012MS0922; the Natural Science Foundation of Inner Mongolia of China under Grant no. 2011MS0912; the Research Foundation of University Doctoral Program under Grant no. 20101501110003.

References

[1] S. R. Dalal, A. Jain, N. Karunanithi et al., "Model-based testing in practice," in *Proceedings of the International Conference on Software Engineering*, pp. 285–294, May 1999.

[2] J. Yan, J. Wang, and H. Chen, "Survey of model-based software testing," *Computer Science*, vol. 31, no. 2, pp. 184–187, 2004.

[3] M. Broy, B. Jonsson, J. P. Katoen et al., *Model-Based Testing of Reactive Systems*, vol. 3472 of *LNCS*, Springer, Heidelberg, Germany, 2005.

[4] Y. Zeng, L. Zhang, Y. Zhang et al., "Activity diagram-based method to generate test sequence of concurrent software," *Computer Science*, vol. 34, no. 12, pp. 286–290, 2007.

[5] H. Watanabe and T. Kudoh, "Test suite generation methods for concurrent systems based on coloured petri nets," in *Proceedings of the 2nd Asia-Pacific Software Engineering Conference*, pp. 242–251, 1995.

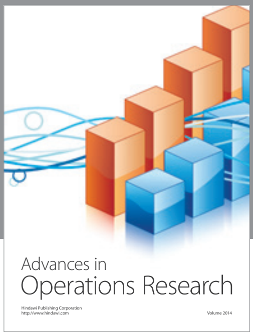
[6] J. Desel, A. Oberweis, T. Zimmer, and G. Zimmermann, "Validation of information system models: petri nets and test case generation," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3401–3406, October 1997.

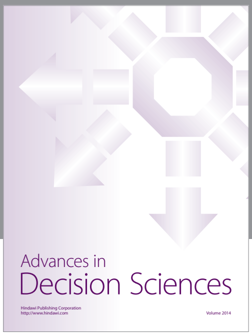
[7] U. Farooq, C. P. Lam, and H. Li, "Towards automated test sequence generation," in *Proceedings of the 19th Australian Software Engineering Conference (ASWEC '08)*, pp. 441–450, Perth, Australia, March 2008.

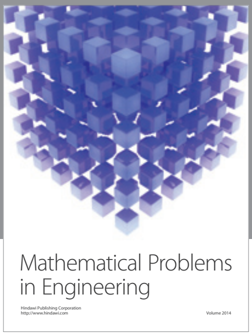
[8] V. Rusu, H. Marchand, and V. Tschaen, "From safety verification to safety testing," in *Proceedings of the IFIP 16th International Conference on Testing Communicating Systems*, pp. 160–176, 2004.

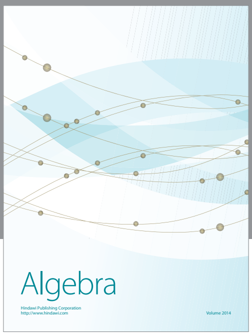
[9] C. Constant, T. Jéron, H. Marchand, and V. Rusu, "Integrating formal verification and conformance testing for reactive systems," *IEEE Transactions on Software Engineering*, vol. 33, no. 8, pp. 558–574, 2007.

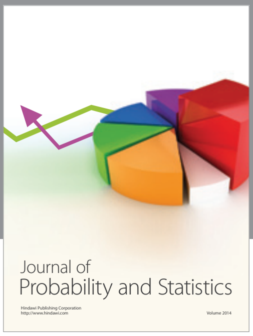
[10] T. Sun, X. Ye, J. Liu et al., "CPN based protocol testing sequence generating method," *Journal of PLA University of Science and Technology (Natural Science Edition)*, vol. 13, no. 2, pp. 165–170, 2012.


Advances in
Operations Research

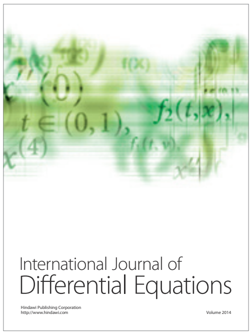

Advances in
Decision Sciences

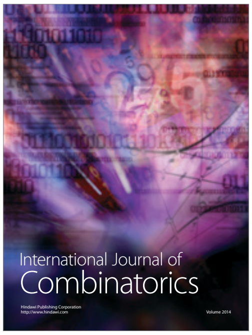

Mathematical Problems
in Engineering

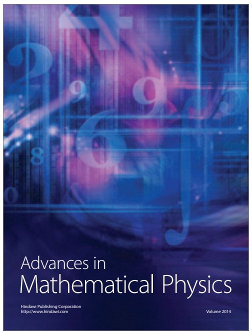

Algebra

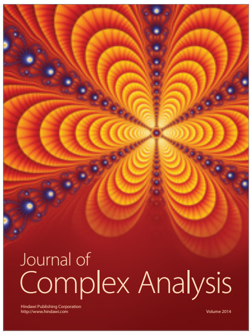

Journal of
Probability and Statistics

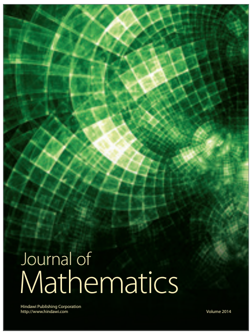
The Scientific
World Journal

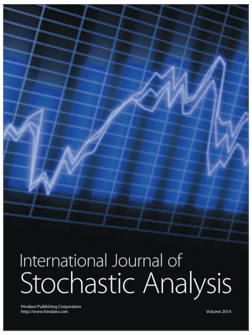

International Journal of
Differential Equations

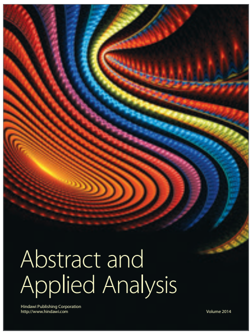

International Journal of
Combinatorics

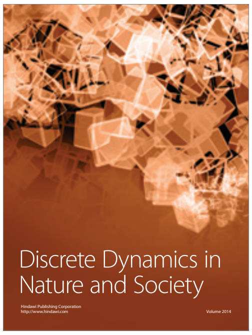

Advances in
Mathematical Physics

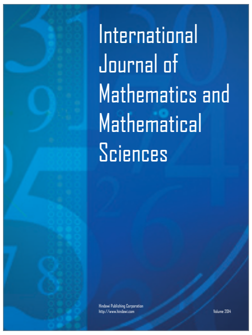

Journal of
Complex Analysis


Journal of
Mathematics


International Journal of
Stochastic Analysis

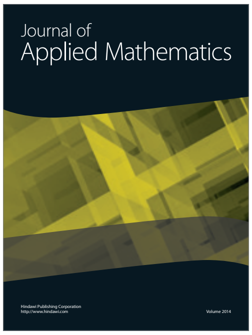

Abstract and
Applied Analysis


Discrete Dynamics in
Nature and Society


International
Journal of
Mathematics and
Mathematical
Sciences

Journal of
Discrete Mathematics

Journal of
Function Spaces


Journal of
Applied Mathematics

Journal of
Optimization