Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080605

I/O Conformance Test Generation with Colored Petri Nets

Jing Liu[1], *Xinming Ye*[1], *Jiantao Zhou*[1,∗] *and Xiaoyu Song*[2,∗]

[1] College of Computer Science, Inner Mongolia University, Hohhot, China
[2] Department of Electrical & Computer Engineering, Portland State University, Portland, Oregon, USA

Received: 7 Oct. 2013, Revised: 5 Jan. 2014, Accepted: 6 Jan. 2014
Published online: 1 Nov. 2014

Abstract: This paper explores Input-Output Conformance (IOCO) test generation with Colored Petri Nets (CPN). A test generation oriented CPN model and CPN based IOCO relation is proposed. Feasible test cases are generated by model simulation with the proof of its soundness. The method integrates the merits the IOCO testing theory and the CPN modeling synergistically, and is applied as a nontrivial and competent test case generation approach for practical testing projects. The effectiveness of this test generation approach is demonstrated by a concrete software system. Since the model simulation based test generation process is irrespective with the model size, the effectiveness of the method is enhanced with scalability.

Keywords: Software testing, test generation, colored Petri nets, input-output conformance, model simulation

1 Introduction

The Conformance testing [1] aims at checking whether the software implementation conforms to its function specification. To make conformance testing more effective and efficient, we should consider the test automation in both test case generation and test execution phases in practical testing projects [2]. General practice these days mainly concentrates on the automatic test execution, such as TTCN-3 based technologies [3]. However, test cases are still generated manually in most testing projects, which is time-consuming, error-prone and costly. In this context, Model Based Testing (MBT) is introduced, which attracts more and more attention of industry-scale testing projects [4,5,6]. It allows for generation of test cases with test oracles from a formal model that specifies software behaviors explicitly. It improves the low-level efficiency and avoids inaccuracy of manual test case generation process.

Network based reactive software systems are ubiquitous, so we treated them as the System Under Testing (SUT) in our studies. Concerning conformance testing towards such kind of software systems, Input-Output COnformance (IOCO) relation based test case generation approach [6] is well recognized for its solid theoretical support [9] and high feasibility in testing practices [10,11] in MBT related studies [5,6,7,8]. It is quite feasible and applicable to black-box conformance testing for network based reactive software systems. The reason is that IOCO relation formally defines what external output should be observed through practical test execution and how to determine the conformance based on these observations.

In this paper, IOCO testing theory is characterized and implemented with Colored Petri Nets (CPN) [12] modeling and a novel conformance test generation approach is proposed consequently. Compared with the test generation approach based on the Labeled Transition System (LTS) model in original IOCO testing theory, our CPN model based test generation approach has several advantages. First, CPN has better formal capabilities to specify and analyze complicated and concurrent software behaviors. They are quite helpful for validating the accuracy of system models, which are the basis for model based testing technology. Second, CPN models can execute dynamically, which is directed by the data-dependent control flow of system behaviors. Generating by such model simulation process, test cases certainly contain actual test data and test oracles, so they are quite feasible for guiding practical test execution. Third, since model simulation based test generation is irrespective with model size, its effectiveness is enhanced

∗ Corresponding author e-mail: cszjtao@imu.edu.cn, song@ece.pdx.edu

with scalability. In a word, a CPN model based IOCO test case generation approach tends to be a promising approach to validate the correctness of reactive network software systems more efficiently and more effectively.

The paper is organized as follows. The framework of our CPN model based IOCO test generation approach is introduced in Section 3. The Test Generation oriented CPN (TGCPN) is proposed as the basic formal model for specifying software functionalities and its implementation behaviors in Section 4. The pioco relation is defined in the context of TGCPN models to precisely specify what it means for an implementation to conform to its specification in Section 5. Finally, a novel test case generation algorithm is developed in Section 6 using the TGCPN model simulation technology to guarantee that all test cases are feasible for the practical test executions. Besides, test generation are also proved to be sound for the conformance determination, that is, as long as the implementation fails one test case, it will definitely not conform to its specification. To show the effectiveness and the feasibility of this test generation approach, we perform test case generation and test execution procedures with the simplified file sharing protocol (SFSP) system as an exemplification.

2 Related Work and Preliminaries

In the original IOCO relation based testing approach [6], LTS with input and output, as IOLTS, is defined to model the specification of a software system, and then IOLTS which is input-enabled, is further defined as IOTS to specify the behavior model of the system implementation. Then, a specific IOCO relation is defined to indicate what actions should be observed during test executions and what it means for an implementation to conform to its specification. Finally, test cases are generated recursively based on the execution paths from a system LTS model. However, in this paper we aim to integrate the IOCO testing theory with the CPN model, and finally develop a CPN model based IOCO test generation approach as our main contribution. That is, we aim to integrate the merits the IOCO testing theory and the CPN modeling synergistically, and apply it as a nontrivial and competent test case generation approach into testing network based reactive software systems.

As for the Petri nets model based testing studies in literatures, most of them are specific application scenarios oriented, for example, stochastic Petri nets based performance testing for network protocols [13], workflow net based distributed testing for service infrastructure components [14], algebraic Petri nets based functionality testing for business process [15], and k-safe Petri nets based conformance testing under several specific fault model assumptions [16].

As for the related work of test case generation approaches based on the CPN models, Watanabe and Kudoh [17] propose a basic test generation algorithm, which could be considered as the first step in this field. First, the reachability tree of a CPN model is constructed, and all input-output sequences from root node to leaf nodes in this tree are traversed to form test cases, and then, equivalent markings in that tree are combined to construct corresponding reachability graph, and FSM model based test case generation approaches are applied directly based on this graph. Farooq et al. [18] use random walk technologies to randomly traverse the model state space to generate test cases, where several sequential coverage criteria and concurrent coverage criteria are proposed to guide test selection. Zhu and He [19] have proposed four specific testing strategies towards the high-level Petri nets. For each strategy, they first define a set of schemes to observe and record testing results, and they also define a set of coverage criteria to measure test adequacy. But, no detailed test case generation algorithms are explicitly presented. We have proposed the introductory idea of CPN model based IOCO testing approach in our short paper [20]. While in this paper, we make better and more complete formal definitions to all key concepts, and propose a totally revised test generation approach with its soundness prove for the conformance determination. Furthermore, we could compare the computation cost in test generation among above methods. In context of CPN, the size of a state space of a system tends to grow exponentially in the number of its actions and variables, where the base of the exponentiation depends on the number of enabled transitions an action has and the number of values a variable may store. But, model simulation just needs linear computation cost to produce a feasible trace. Therefore, state space traversal based test generation methods proposed in [17,18] definitely cost much more than simulation based test generation method proposed in [20] and this paper.

CPN is advantaged for modeling and validation of systems where concurrency and communication are key characteristics. Its formal definitions are referred as [12]. Besides, other key definitions concerning the behavior simulation of CPN models which are used in following sections are listed as follows.

Definition 1. For a *CPN* = (P, T, A, Σ, V, C, G, E, I):
(1) **pre-set** and **post-set** of a place: $\forall \mathrm{p} \in \mathrm{P}$:
pre(p)=$\{\mathrm{t} \in \mathrm{T} \mid (\mathrm{t},\mathrm{p}) \in \mathrm{A}\}$; *post*(p)=$\{\mathrm{t} \in \mathrm{T} \mid (\mathrm{p},\mathrm{t}) \in \mathrm{A}\}$.
pre-set and **post-set** of a transition: $\forall \mathrm{t} \in \mathrm{T}$:
pre(t)=$\{\mathrm{p} \in \mathrm{P} \mid (\mathrm{p},\mathrm{t}) \in \mathrm{A}\}$; *post*(t)=$\{\mathrm{p} \in \mathrm{P} \mid (\mathrm{t},\mathrm{p}) \in \mathrm{A}\}$.
(2) $M \overset{\sigma}{\Rightarrow} =_{def} \exists M_i : M \overset{\sigma}{\Rightarrow} M_i$, where
$\sigma = (\mathrm{t}_0,\mathrm{b}_0),(\mathrm{t}_1,\mathrm{b}_1),\ldots(\mathrm{t}_{i-1},\mathrm{b}_{i-1})$,
$M \xrightarrow{(\mathrm{t}_0,\mathrm{b}_0)} M_1 \xrightarrow{(\mathrm{t}_1,\mathrm{b}_1)} \ldots \xrightarrow{(\mathrm{t}_{i-1},\mathrm{b}_{i-1})} M_i$;
if $|\sigma| = 1$, $M \overset{\sigma}{\to}$ is used instead.
(3) $\textbf{trace}(M) =_{def} \{\sigma \in BE(\mathrm{T})^* \mid M \overset{\sigma}{\Rightarrow}\}$.
(4) $M\ \textbf{\textit{fires}}\ \sigma =_{def} \{M_n \mid M \overset{\sigma}{\Rightarrow} M_n,\ \sigma \in BE(\mathrm{T})^*\}$.
(5) *CPN* is **Deterministic**, if $\mid M\ \textbf{fires}\ \sigma \mid \leq 1$.
(6) *CPN* has **Finite Output**, if $\mid M\ \textbf{fires}\ \sigma \mid \leq \mathrm{n}(\mathrm{n} \in \mathrm{N})$;

(7) *CPN* has **Finite Behavior**, if $\exists n \in N$, $\forall \sigma \in \textbf{trace}(M_0) : |\sigma| < n$.

3 Methodology Overview

We aim to integrate the IOCO testing theory with the CPN model, and finally develop a CPN model based IOCO test generation approach as our main contribution. However, such integration does not just means to simply replace LTS with CPN. Three specific problems need to be resolved when concretize the IOCO testing theory with the CPN model. Each of them plays a significant part in test generation process, and they work together to form final test case generation approach, where its framework is presented in Fig. 1.

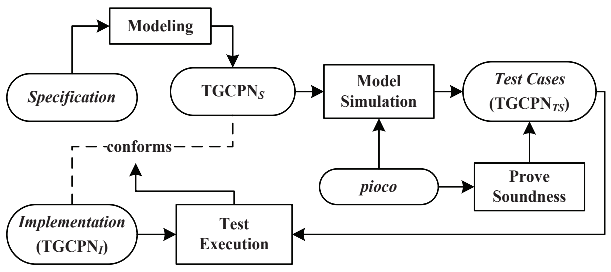

Fig. 1: The framework of the CPN model based test generation approach.

First, we need to propose a new kind of CPN model that accurately specifies key characteristics and requirements for the conformance testing scenario. As the central idea of IOCO testing theory is to compare all external visible actions between system models and actual system implementations during test executions, the new kind of CPN model should explicitly specify such external visible actions, i.e., to make the most of both place and transition elements in CPN model to distinguish visible actions from internal actions. Especially, to deal with the special output actions, such as the quiescence or deadlock [9], we should introduce new kind of transitions to model them accurately. In section 4, TGCPN is proposed to resolve the preceding problems, and acts as formal models to specify function behaviors for a software (TGCPN$_S$) and its implementations (TGCPN$_I$).

Second, we need to propose a new implementation relation in the context of TGCPN model to precisely specify what it means for an implementation to conform to its functional specification. In original IOCO relation definition, the inclusion relation of external output actions is formally specified for determining IOCO conformance. While in the TGCPN context, system behaviors are simulated with specific data, and the problem becomes how to determine the IOCO conformance via comparing output actions with specific data. In section 5, a *pioco* relation is proposed to resolve the preceding problems, where markings produced during model simulation are used to accurately define such output data comparison.

Third, based on the TGCPN model and the *pioco* relation, we need to develop a feasible test case generation approach with two desired requirements. One is to make the test generation process scale for dealing with more complicated system models, and the other one is to make all test cases feasible for the practical test executions. In section 6, the model simulation technology is utilized to generate sound and feasible test cases, which are modeling as TGCPN$_{TS}$. It not only satisfies the preceding two requirements, but also highlights the advantages of our CPN model based test generation approach. Furthermore, the generated test cases are also proved to be sound for the conformance determination, i.e., as long as an system implementation fails one test case, it will not conform to its specification.

To sum up, constructing the CPN model based IOCO test generation approach is challenging but quite promising and such novel approach has solid foundation to be used as a competent and effective choice among the Petri nets model based conformance testing approaches. Since model simulation based test generation is irrespective with model size, its effectiveness is enhanced with scalability, so it is quite suitable for testing reactive network software systems.

4 TGCPN Modeling

Software specification and its implementation should both be formal objects in MBT, so TGCPN$_S$ is proposed as formal model for system specifications, and TGCPN$_I$ is proposed as formal model for system implementations.

4.1 Specification modeling

Definition 2. A TGCPN$_S$ model is a triple (*CPN*, P$_S$, T$_S$):
(1) *CPN* is a a basic colored Petri nets model.
(2) $P_S = P$, $P_S = P_S^O \cup P_S^E$:
P_S^O is the set of **Observable Places**; P_S^E is the set of **Internal Places**; $P_S^O \cap P_S^E = \phi$.
(3) $T_S = T$, $T_S = T_S^I \cup T_S^O \cup T_S^E$:
T_S^I is the set of **Input Transitions**; T_S^O is the set of **Output Transitions**; T_S^E is the set of **Internal Transitions**; $T_S^I \cap T_S^O = T_S^I \cap T_S^E = T_S^O \cap T_S^E = \phi$.
(4) TGCPN$_S$ has **Finite Output**.
(5) TGCPN$_S$ does not have infinite sequences, composed of internal actions, i.e., $\neg \exists M \ : \ M \overset{\sigma}{\Rightarrow} M, \sigma \in BE(T_S^E)^*$.

In TGCPN$_S$ modeling, token data in the observable places could present externally observed data, so an observable place is always the post-set of an input transition or output transition to display what data should be observed after executing those external transitions. The input transition models input actions where input data is provided by the testing environment (i.e. tester). The output transition models output actions which can produce visible output observations. Thus, observable places and input/output transitions are used together to explicitly specify external visible system behaviors. Besides, internal transitions and internal places could represent unobservable executions of system behaviors.

In TGCPN$_S$ modeling, some conformance testing oriented modeling constrains should be made. First, given definite input data, the system must produce finite output results. Otherwise, within finite test execution steps, we cannot make a definite decision. Second, the model must not have the loop of internal transitions, because it will make system implementation having no response, and we cannot distinguish this scenario from the deadlock.

The TGCPN$_S$ model for the SFSP system is presented in Fig. 2. In this protocol, file data are downloaded piece by piece controlled by a packet number. When a downloader gets a correct data piece, it increases the packet number, and then sends it to the sender as an acknowledgement, and requires a new data piece with that packet number. In this model, *A/C/Received* are observable places. *SendPacket* is an input transition and *ReceivePacket* is an output transition. The rest are internal places or internal transitions. If *SendPacket* fires, we could observe which data packet is sent according to the tokens in *A*. Furthermore, if *ReceivePacket* fires, we could both observe which data packet is downloaded according to the tokens in *Received*, and which packet number should be sent according to the tokens in *C*. In this way, necessary external behaviors of a system for its conformance testing are modeled accurately. However, in system modeling practice, the selection of observable places should consider actual observation points in actual test execution, such as the Points of Control and Observation in TTCN based testing method [3].

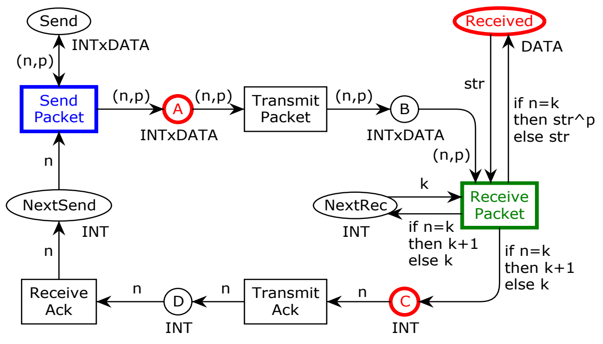

Fig. 2: The TGCPN$_S$ model for the SFSP system.

4.2 Implementation modeling

Since IOCO relation is a formal conformance relation between system specification and implementation, they both need to be formal object for formal reasoning. However, implementations of a system are real physical things, which consist of software, hardware, or a hybrid system, and only testing experiments could be performed on them. Therefore, the test hypothesis mentioned in [9] assumes that the formal model of implementation is available a-priori, but cannot be explicitly modeled. Therefore, we propose TGCPN$_I$ to just formally specify the system implementation.

Definition 3. A TGCPN$_I$ model is a triple (CPN, P_I, T_I):
(1) *CPN* is a a basic colored Petri nets model.
(2) $\mathrm{P}_I^O = \mathrm{P}_S^O$, $\mathrm{T}_I^I = \mathrm{T}_S^I$, $\mathrm{T}_I^O = \mathrm{T}_S^O$.
(3) $\{\delta\} \subset \mathrm{T}_I$: δ is the **suspension transition**, and $M \xrightarrow{\delta} M$.

For the same system, its implementation model and related specification model have same observable places and input/output transitions. Besides, since test outputs are controlled completely by the implementation, any possible output may produce, such as real output data, deadlock or just quiescence. The TGCPN$_I$ model should support specifying these output scenarios. Especially, the quiescence indicates that an implementation has no visible output as it is just waiting for an input to proceed. In TGCPN$_I$, quiescence is defined as:
$\exists\sigma \in BE(\mathrm{T})^*,\ \forall(\mathrm{t},\mathrm{b}) \in BE(\mathrm{t}), \mathrm{t} \in \mathrm{T}^O \cup \mathrm{T}^E$:
$\neg((M_0\ \mathbf{fires}\ \sigma) \xrightarrow{(\mathrm{t,b})})$.

Producing the quiescence is a kind of a special output action, modeled as the suspension action δ. Firing a suspension action indicates that the implementation stays in the same state and an input is expected as a trigger to let system continue execution.

In a word, TGCPN$_S$ modeling is very significant for performing actual software test generation practice, because a well understandable and accurate model is the first and crucial step to the success of MBT technologies.

5 PIOCO Relation

As system behaviors are executed with specific data in the TGCPN context, the conformance relation used in our testing approach should also be determined according to specific data. Thus, we propose the *pioco* relation as follows.

Definition 4. *pioco* is a binary relation with **ss** $\in$ TGCPN$_S$ and **ii** $\in$ TGCPN$_I$, where:
ii *pioco* **ss** $=_{def}$ $\forall\sigma \in SPtrace(M_S)$:
outtoken(M_I **fires** σ) = ***outtoken***(M_S **fires** σ)
(1) $SPtrace(M_S)$ $=_{def}$ $\{\sigma \in (BE(\mathrm{T}) \cup \delta)^* \mid M_S \overset{\sigma}{\Rightarrow}\}$; it enumerates all feasible traces of the model **ss**, including

suspension transitions. M_S and M_I are initial markings of respective models.
(2) ***outtoken***$(M) =_{def} \{M(\mathrm{P}) \mid \mathrm{P} \in \mathrm{P}_S^O \cup \mathrm{P}_I^O\}$, it represents the observable output token data. In model **ss**, it records token data of observable places under a specific marking. While in model **ii**, it shows actual observable output data produced by system implementation during test execution.

Guided by above *pioco* definition, the conformance is determined by comparing token data in observable places along a specific *SPtrace* with actually observed output from system implementation under testing. If they are the same, we could determine that system implementation executes as expected. However, if actual output data are different from what prescribed in **ss** model, we conclude with the non-conformance. The equivalence of above two ***outtoken*** sets indicates that all prescribed observations should be observed in practical test executions, i.e. prescribed functionalities must be completely implemented. Therefore, the implementation that has valid but partial functionalities will not be determined to conform to its specification model any more.

Given definite M_S and M_I, several feasible traces that may contain suspension actions are generated to produce test case models. As a representative, **t** is one of them. We put input data in the **t** to the implementation **i** and observe its output. If actual observations are all exactly prescribed as valid outputs in related observable places in the **t**, " **i** pass **t** " is determined. If all test cases in T_S that generated from all possible initial markings are passed, " **i** *pioco* **s** " is consequently determined, i.e. **i** *pioco* **s** $\Leftrightarrow$ **i** *pass* T_S. However, in practical conformance testing, generating all test cases in T_S is almost infeasible. In addition, conformance testing just aims to find non-conformance faults between an implementation and its specification, rather than to prove their conformance relation. Hence, a weaker requirement is usually considered in conformance testing practice, that is, as long as an implementation does not pass one test case, it definitely does not conform to its specification. This weaker requirement corresponds to the left-to-right implication in " **i** *pioco* **s** $\Leftrightarrow$ **i** *pass* T_S ", and is referred as the ***soundness*** of test case generation.

In this paper, under the guidance of the *pioco* relation, a sound test case generation approach based on the TGCPN$_S$ model simulation technology is developed and demonstrated in the following section.

6 Model Simulation based Test Case Generation

In Section 3, we mention that developing the test case generation approach based on a TGCPN model and the *pioco* relation should satisfy two significant requirements. First, the test generation approach should be with high scalability for dealing with large-scale TGCPN models. Second, all test cases should be sound and feasible for the practical test executions. Accordingly, we propose a model simulation based test case generation approach to satisfy both requirements.

The intuitive idea of our test case generation approach is essentially a traversal towards a system TGCPN$_S$ model. This kind of traversal is performed by model simulation process. That is, a specific initial marking conducts simulating system TGCPN$_S$ model once, and during simulation, several testing sequences that composed of external places and transitions are constructed and related data-dependent test oracles are added for validating the actual observations with respect to the prescribed output. When there are no transitions are enabled, simulation is terminated, and consequently test case models are generated. To present feasibility and effectiveness of this approach, we adopt SFSP system to demonstrate its detailed application procedure.

6.1 TGCPN based test case model

Before presenting our detailed test generation algorithm, we first formally define how to model a test case as a TGCPN$_{TS}$ model in the context of TGCPN.

Two kinds of data information should be specified in the TGCPN$_{TS}$ models explicitly. One is input action with input data and the other is output action with output data, i.e. test oracles. Besides, we need some supplementary places and transitions to record the provisional or final conformance decisions. Several constrains should also be fulfilled about test case modeling. First, TGCPN$_{TS}$ should be deterministic and its every feasible trace should have finite length, otherwise, test execution based on this model cannot terminate in finite steps and with definite testing results. Second, TGCPN$_{TS}$ should have only one input transition enabled at each step, which is fundamental constrain for test case specification.

Definition 5. A TGCPN$_{TS}$ model is a triple (CPN, P_{TS}, T_{TS}):
(1) CPN is a a basic colored Petri nets model.
(2) $\mathrm{P}_{TS} = \mathrm{P}$, $\mathrm{P}_{TS} = \mathrm{P}_{TS}^I \cup \mathrm{P}_{TS}^O \cup \mathrm{P}_{TS}^{TO} \cup \mathrm{P}_{TS}^V$:
P_{TS}^I is the set of **Input Places**; P_{TS}^O is the set of **Observable Places**; P_{TS}^{TO} is the set of **Test Oracle Places**; P_{TS}^V is the set of **Test Verdict Places**; each two sets have no intersections.
(3) $\mathrm{T}_{TS} = \mathrm{T}$, $\mathrm{T}_{TS} = \mathrm{T}_{TS}^I \cup \mathrm{T}_{TS}^O \cup \mathrm{T}_{TS}^\delta \cup \mathrm{T}_{TS}^V$:
T_{TS}^I is the set of **Input Transitions**; T_{TS}^O is the set of **Output Transitions**; T_{TS}^δ is the set of **Suspension Transitions**; T_{TS}^V is the set of **Test Verdict Transitions**; each two sets have no intersections.
(4) TGCPN$_{TS}$ is **Deterministic**.
(5) TGCPN$_{TS}$ has **Finite Behavior**.
(6) TGCPN$_{TS}$ has at most one input transition enabled in each step, i.e.

$$\neg\exists M : M \xrightarrow{(\mathrm{t}_1,\mathrm{b}_1)} \wedge M \xrightarrow{(\mathrm{t}_2,\mathrm{b}_2)}, \mathrm{t}_1 \in \mathrm{T}_{TS}^I \wedge \mathrm{t}_2 \in \mathrm{T}_{TS}^I.$$

TGCPN$_{TS}$ models, which are used as actual test cases, can facilitate actual test execution for its better feasibility and readability, because they not only prescribe specific data based detailed system behavior flows, but also provide necessary and definite test oracles for determining the conformance relation.

6.2 Test generation via model simulation

The TGCPN model simulation based test case generation algorithm is formally specified as following procedure *TestGen*.

Given: $s \in$ TGCPN$_S$, and M_0 as its initial marking, for every *SPtrace* $\sigma \in BE(\mathrm{T}_S)^*$, i.e., $M_0 \xrightarrow{(\mathrm{t}_0,\mathrm{b}_0)} M_1 \xrightarrow{(\mathrm{t}_1,\mathrm{b}_1)} \ldots \longrightarrow M_k$, one TGCPN$_{TS}$ model is generated along with this *SPtrace* generation process guided by the following procedure.

```
Procedure TestGen {
[beginning with the first firing transition in an SPtrace]
$t \in T_S : t = t_0; M = M_0; PRec = \phi;$
while (SPtrace generation is not terminated ) {

    if $t \in T_S^I$ {
      [insert suspension transition]
      Generate $t_s \in T_{TS}^I : t_s = t;$
      ForAll $p_t \in PRec \;\; pre(t_s) = p_t; PRec = \phi;$
      ForAll $p \in pre(t)$ {
        Generate $p_{sa} \in P_{TS}^I : p_{sa} = p; pre(t_s) = p_{sa};$
        Generate $t_{ss} \in T_{TS}^{\delta} : pre(t_{ss}) = post(t_{ss}) = p_{sa};$ }
    }

    if $t \in T_S^O$ {
      $t_s \in T_{TS}^O;$
      ForAll $p_t \in PRec \;\; pre(t_s) = p_t; PRec = \phi;$
    }

    ForAll $p \in post(t)$ {
      [eliminate inner places]
      if $p \in P_S^E$ and $pre(t) == post(t) == p$
        continue;
      [insert test oracle and verdict units]
      if $p \in P_S^O${
        Generate $p_{sb} \in P_{TS}^O : p_{sb} = p; post(t_s) = p_{sb};$
        Generate $p_{sb2} \in P_{TS}^{TO};$
        Generate $t_v \in T_{TS}^V, p_v \in P_{TS}^V$:
            $pre(t_v) = p_{sb} \cup p_{sb2}; \;\; post(t_v) = p_{sb} \cup p_v;$ }
    }

    if $t \in T_S^I$ or $t \in T_S^O$ {
      [fires t to store test oracle data]
      Firing t with b that $(t,b) \in \sigma : M \xrightarrow{(t,b)} M';$
      for each pair $p \in P_S^O$ and $p_{sb2} \in P_{TS}^{TO}$
        $M'(p_{sb2}) \;=\; M'(p);$
      $M \Leftarrow M'; t \Leftarrow$ next enabled transition; continue;
    }

    if $t \in T_S^E$ {
      [eliminate inner transitions]
      Firing t with b that $(t,b) \in \sigma : M \xrightarrow{(t,b)} M';$
      for each $p \in pre(t)$ and $p \in P_S^O$
        $PRec = PRec \cup \{p\};$
      $M \Leftarrow M'; t \Leftarrow$ next enabled transition;
    }
} [end of while]
} [end of Procedure]
```

The input of this algorithm is a validated TGCPN$_S$ model for a specific system and M_0 as its initial marking. Under this initial marking, several *SPtraces* could be produced. Because we suppose that every trace should have finite length, one *SPtrace* could be specified as: $M_0 \xrightarrow{(\mathrm{t}_0,\mathrm{b}_0)} M_1 \xrightarrow{(\mathrm{t}_1,\mathrm{b}_1)} \ldots \longrightarrow M_k$.
Then, along with each *SPtrace* generation by the traversal of above TGCPN$_S$ model, a corresponding TGCPN$_{TS}$ models is generated. When M_k is reached, the generation process is terminated. Therefore, one *SPtrace* leads to one TGCPN$_{TS}$ model.

Th model simulation based test generation process is essentially driven by transition firing. We should deal with all kinds of transitions that exist in the TGCPN$_S$ model. According to Definition 2 above, we should deal with input and output transitions, and internal transitions, so the test case generation algorithm is mainly composed of three parts as follows.

Part 1: deal with input and output transition

As for an input transition, every place in its pre-set becomes an input place in the TGCPN$_{TS}$, and suspension transition t$_{ss}$ is added to each input place for specifying the allowance for observing the quiescence. That is, when a suspension transition is fired, the implementation under testing remains in the same state without any observable output and an input from testers is expected as a trigger to make the implementation proceed. However, if token data in an input place cannot be controlled externally by testers, where no quiescence is produced, we should delete its suspension transition after the test generation.

An input or output transition should definitely be added into a test case model to specify external behavior for test execution, and its related input places and observable places should also be added into that test case model. Besides, we use a places set *PRec* to store observable place that generated in current step, and link them to the next processing input or output transition to keep connectivity of model elements.

Part 2: deal with test oracle and verdict unit

Every observable place p in model *s* becomes an observable place p$_{sb}$ in test case model, and its coupled test oracle place p$_{sb2}$ is inserted. The token data produced in p are copied to p$_{sb2}$ by firing the current transition. When TGCPN$_{TS}$ is applied into test executions, p$_{sb}$ stores the actual output data produced by system implementation, and token data stored in p$_{sb2}$ previously will act as its valid test oracles. Besides, we generate a test verdict transition t$_v$ which takes p$_{sb}$ and p$_{sb2}$ as its pre-set and a newly added test verdict place p$_v$ and p$_{sb}$ as its post-set. They work coordinately as a verdict unit to check whether token data in p$_{sb}$ and p$_{sb2}$ are consistent. Specifically, if we do not observe correct output compared with its test oracles in a TGCPN$_{TS}$ model, whether it is actual data output or just quiescence, a ***fail*** token is produced into the corresponding test verdict place. Otherwise, a ***pass*** token is produced to indicate continuing test execution. If the test execution terminates with all *pass* tokens in TGCPN$_{TS}$, the implementation under testing passed this test case. If just one *fail* token is produced, that implementation has non-conformance against the system TGCPN$_S$ model.

Part 3: deal with internal transition

As internal actions cannot be observed externally via input and output in real black box testing executions, we need not to add internal places and internal transitions into the TGCPN$_{TS}$ model. However, this kind of model reduction should guarantee no harm to system functionalities, that is, the data-dependant control flow of the system behavior should be kept. Therefore, we just let internal transitions fire, then record makings and use the *PRec* set mentioned above to keep behavior connectivity of the generated TGCPN$_{TS}$ model.

It should be noted that when a valid specific initial marking is assigned in an actual TGCPN$_S$ model, at least one *SPtrace* exist definitely, so at least one test case model are generated. Directed by different initial markings, several test case models are generated to test corresponding software behaviors. Based on our test case generation approach, every feasible trace takes M_k as the final marking, so the *SPtrace* has finite length, that is, the generation algorithm are terminated in finite steps, and produces a TGCPN$_{TS}$ model with finite behaviors.

Three more aspects should be noted concerning above test generation procedure. First, TGCPN models are not added new kinds of model elements, and the modeling constrains are just used to avoid generating infeasible traces for testing scenarios. So, the semantic rules defined in [12] are all kept in TGCPN models, that is, we still use its original enabling rules and occurrence rules to generate TGCPN$_{TS}$ models from corresponding TGCPN$_S$ model for a specific software system. Second, this generation approach could be applied into the hierarchical CPN models without any modifications, because test case model is generated through the model simulation process, which is irrelative with hierarchical or non-hierarchical model characteristic. Third, if initial marking is given, this test generation procedure could be executed automatically, so it is indeed a promising approach to improve the test automation in test generation phase.

6.3 Examplification: SFSP System

Take SFSP system as an example, its TGCPN$_{TS}$ model, presented in Fig. 3, is generated through simulating TGCPN$_S$ model in Fig. 2 under initial marking: $M_0(Send)$ = {(1,“first”), (2,“second”)} and $M_0(NextSend)$ = $M_0(NextRec)$ = {1}. It aims to test whether two specific data packets are sent sequentially.

Comments about the generation process are noted:

(1) Input place *Send* has a suspension transition *suspen*, which allows for waiting to send data packet. However, place *C* has no suspension transition, as it is an output place, and tokens in *C* cannot be controlled by testers externally.

(2) Input and output transitions are correspondingly modeled. As an example, when the binding element (*SendPacket*, {n=1, p=“first”}) fires, a new token (1, “first”) will be produced in place *A*. Then, test oracle place *TO_A* is generated to store this token; test verdict transition *verdict1* and test verdict place *v1* are generated to validate whether real output data are same to token data in *TO_A*, i.e. n=1 and p=“first”.

(3) *v1*, *v2*, *v3* are verdict places allowing for only *pass* or *fail* tokens, which could be modeled as a fusion set.

(4) Internal places and transitions are fired to record markings, but not included into the TGCPN$_{TS}$ model.

Test cases are generated with different initial markings, so the test selection issue [6] should be considered. In this paper, we just select initial markings randomly, and we will adopt more specific test selection policies in our further studies.

To further validate the feasibility and effectiveness of above test case generation procedure, we perform the practical test execution towards different SFSP system implementations. As shown in Table 1, three test cases are generated according to respective initial markings for different testing scenarios. Also, we program six implementations of SFSP system with pre-injected errors to act as system under testing. The detailed description of implementation errors are listed in Table 1.

We perform actual test execution using three test cases towards six implementations. As i1 passes all test cases, it conforms to the specification. As i3 $\sim$ i5 have fatal errors, they do not pass any test case where fail token appears in test case executions. In testing i2 with tc2, we do not observe the retransmission packet, though the suspension is allowed, so i2 does not pass tc2. While in testing i6 with tc2, a fail token is produced because of retransmission error, so i6 does not pass tc2 either. However i2 and i6 pass tc1 and tc3, because these test cases do not touch retransmission functionality. According to the soundness definition, i2 and i6 do not conform to the specification. Compared with basic IOCO relation based testing approach, i2 does not conform to the specification under the *pioco* relation, so partially correct implementations are no longer determined to conform to the specification. In a word, generated test

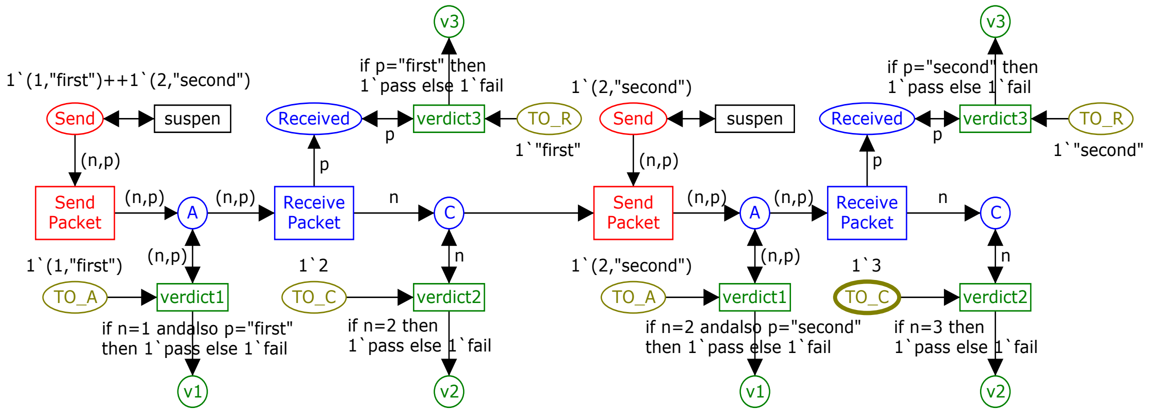
1`(1,"first")++1`(2,"second")
Send
suspen
(n,p)
Send Packet
(n,p)
A
(n,p)
Receive Packet
Received
p
verdict3
TO_R
1`"first"
if p="first" then 1`pass else 1`fail
v3
n
C
1`(1,"first")
TO_A
verdict1
if n=1 andalso p="first" then 1`pass else 1`fail
v1
1`2
TO_C
verdict2
if n=2 then 1`pass else 1`fail
v2
1`(2,"second")
if n=2 andalso p="second" then 1`pass else 1`fail
1`3
if n=3 then 1`pass else 1`fail
if p="second" then 1`pass else 1`fail
1`"second"

Fig. 3: An exemplified TGCPN$_{TS}$ model for the SFSP system.

cases have better feasibility and effectiveness for detecting such deficiency.

Through above practical conformance testing process, we find that our generated test cases are quite feasible for guiding the test execution intuitively. More importantly, TGCPN$_{TS}$ models are also effective for finding various implementation faults. Based on the final testing results, conformance relation between an implementation and its specification model is accurately determined.

Table 1: Protocol Implementations, Test Cases and Testing Result

Test Cases:

No.	testing scenarios	initial marking
tc1	one packet without retransmission	$M_0(Send)$={(1,“first”)}, $M_0(NextSend)$={1}, $M_0(NextRec)$={1}
tc2	one packet with retransmission	$M_0(Send)$={(2,“second”)}, $M_0(NextSend)$={2}, $M_0(NextRec)$={1}
tc3	two packets without retransmission	$M_0(Send)$={(1,“first”), (2,“second”)}, $M_0(NextSend)$={1}, $M_0(NextRec)$={1}

Protocol Implementations:

No.	type	description
i1	total correct	implement all functions correctly
i2	partial correct	have no packet retransmission function
i3	faulty	packet sending error
i4	faulty	packet data verification error
i5	faulty	packet number verification error
i6	faulty	packet retransmission error

Testing Results:

	i1	i2	i3	i4	i5	i6
tc1	*pass*	*pass*	*fail*	*fail*	*fail*	*pass*
tc2	*pass*	*fail*	*fail*	*fail*	*fail*	*fail*
tc3	*pass*	*pass*	*fail*	*fail*	*fail*	*pass*

6.4 Soundness of the test generation

Theorem. Let $ss \in$ TGCPN$_S$ be a specification, and let T_S be the completer set of test cases that generated from ss, let *TestGen*: TGCPN$_S$ $\rightarrow$ TGCPN$_{TS}$ be the test case generation function that satisfies $TestGen(ss) \subseteq T_S$, then *TestGen* is sound for *ss* with respect to *pioco* relation.

Proof. Supposing $ii \in$ TGCPN$_I$ and $tt \in TestGen(ss)$ satisfying: ***not*** *(ii **pass** tt)* and *ii **pioco** ss*, then:
not *(ii **pass** tt)*
$\Rightarrow \exists\, e \in \mathrm{M}(p),\ p \in \mathrm{P}^{V}_{TS},\ e \in \{fail\}$;
[a *fail* token appears in test verdict place p]
$\Rightarrow \exists\, r \in \mathrm{P}^{O}_{TS} \wedge \exists\, q \in \mathrm{P}^{TO}_{TS},\ \mathrm{M}(r) \neq \mathrm{M}(q)$;
[token data in observable place r and its coupled test oracle place q are different]
$\Rightarrow \exists \sigma \in SPtrace(M_S)$:
outtoken(M_I **fires** σ) $\neq$ *outtoken*(M_S **fires** σ), where $\mathrm{M}(r) \subseteq (M_I$ **fires** $\sigma)$ and $\mathrm{M}(q) \subseteq (M_S$ **fires** $\sigma)$.
[there exists a trace that results in unexpected output observations in the practical test execution]

Obviously, there exists an obvious contradiction with the assumption *ii **pioco** ss*. Then, we conclude that if one test case does not pass, *ii **pioco** ss* does not hold definitely. So, *TestGen* is sound for *ss* with respect to the *pioco* relation. □

7 Conclusion

To make the best of advantages of IOCO testing theory and CPN modeling, we integrate them directly to develop a non-trivial CPN model based conformance test generation approach. First, the TGCPN is proposed as basic formal models for specifying accurately software functional behaviors. Then, the *pioco* relation is defined as a new conformance relation in the context of TGCPN modeling and as a precise guidance for generating sound test case. Finally, we develop a model simulation based test case generation algorithm and prove its soundness. Throughout the practical test generation and test execution for SFSP system, the feasibility and effectiveness of the preceding test generation approach is well elaborated.

The advantages of our CPN model based test generation approach mainly come from three aspects. First, better formal modeling and analysis capabilities of the CPN facilitate validating the accuracy of the system specification models, which are the basis for the MBT technologies. Second, model simulation essentially reflects the data-dependent control flow of the system behaviors, and it executes with visible feedbacks. As test cases are generated by such model simulation process, actual test input data and test oracles are contain, and it is quite feasible for guiding practical test executions. Third, we do not need to produce the model state space in the generation process at all, so test case generation is irrespective with model size, and consequently we do not face with state explosion problem which is mainly affects the scalability of test generation. Our method was applied into a BitTorrent based protocol system [21], which acts as a representative of large-scale software models, to illustrate the better scalability of our method in practical conformance testing projects.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (61262017, 61262082, 61163011), the Key Project of Chinese Ministry of Education (212025), the Inner Mongolia Science Foundation for Distinguished Young Scholars (2012JQ03), the Introduction Foundation for High-Level Talents of Inner Mongolia University. This work was supported in part by the Bagui scholarship project.

References

[1] ISO/IEC 9646-1, Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts, (1994).

[2] R.M Hierons, K. Bogdanov, J.P Bowen, R. Cleaveland, etc., Using Formal Specification to Support Testing, ACM Computing Surveys, **41**, 9-84 (2009).

[3] I. Schieferdecker, J. Grabowski, T. Vassiliou-Gioles and G. Din, The Test Technology TTCN-3, Formal Methods and Testing, LNCS, **4949**, 292-319 (2008).

[4] M. Utting, The Role of Model-Based Testing, Verified Software: Theories, Tools, Experiments, LNCS, **4171**, 510-517 (2008).

[5] M. Broy, B. Jonsson, J.P Katoen, M. Leucker and A. Pretschner, Model-Based Testing of Reactive Systems, Heidelberg: Springer, Berlin, (2005).

[6] J. Tretmans, Model Based Testing with Labelled Transition Systems, Formal Methods and Testing, LNCS, **4949**, 1-38 (2008).

[7] B. Jeannet, T. Jeron and V. Rusu, Model-Based Test Selection for Infinite State Reactive Systems, Proceedings of the 5th International Symposium on Formal Methods for Components and Objects, 47-69 (2006).

[8] G. Fraser, F. Wotawa and P.E Ammann, Testing with Model Checkers: A Survey, Software Testing, Verification and Reliability, **19**, 215-261 (2009).

[9] J. Tretmans, Test Generation with Inputs, Outputs and Repetitive Quiescence, Software - Concepts and Tools, **17**, 103-120 (1996).

[10] J. Tretmans, E. Brinksma, TorX: Automated Model Based Testing, Proceedings of the 1st European Conference on Model-Driven Software Engineering, Nuremberg, 1-13 (2003).

[11] C. Jard, T. Jeron, TGV: Theory, Principles and Algorithms: A Tool for the Automatic Synthesis of Conformance Test Cases for Non-Deterministic Reactive Systems, International Journal on Software Tools for Technology Transfer, **7**, 297-315 (2005).

[12] K. Jensen and L.M Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Heidelberg: Springer, Berlin, (2009).

[13] M.W Xu, C. Lin, J.P Wu, Network Protocol Performance Testing based On Stochastic Petri Nets, Journal of Software, **10**, 248-252 (1999) (in Chinese with English abstract).

[14] L. Gonczy, R. Heckel and D. Varro, Model-Based Testing of Service Infrastructure Components, Proceedings of the 19th International Conference on Testing of Communicating Systems and the 7th International Workshop on Formal Approaches to Testing of Software, 155-170 (2007).

[15] D. Buchs, L. Lucio and A. Chen, Model Checking Techniques for Test Generation from Business Process Models, Proceedings of the 14th Ada-Europe International Conference, 59-74 (2009).

[16] G. Bochmann and G. Jourdan, Testing k-Safe Petri Nets, Proceedings of the 21st International Conference on Testing of Communicating Systems and 9th International Workshop on Formal Approaches to Testing of Software, 33-48 (2009).

[17] H. Watanabe and T. Kudoh, Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets, Proceedings of the 2nd Asia-Pacific Software Engineering Conference, 242-251 (1995).

[18] U. Farooq, C.P Lam and H. Li, Towards Automated Test Sequence Generation, Proceedings of the 19th Australian Conference on Software Engineering, 441-450 (2008).

[19] H. Zhu and X.D He, A Methodology of Testing High-level Petri Nets, Information and Software Technology, **44**, 473-489 (2002).

[20] J. Liu, X.M Ye and J. Li, Colored Petri Nets Model based Comformance Test Generation, Proceedings of the 16th Symposium on Computers and Communications, 967-970 (2011).

[21] J. Liu, H.B Wu, Make Systematic Conformance Testing for BitTorrent Protocol Feasible: A CP-nets Model Based Testing Approach, Proceedings of the 31th IEEE International Performance Computing and Communications Conference, 201-202 (2012).

Jing Liu received his Ph.D. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences in 2011. Currently he is a lecturer at the Inner Mongolia University. His main research interests include model based software testing, conformance testing theory, automatic test generation, formal methods, especially Petri nets based software verification and validation methods.

Xinming Ye is currently a professor at the Inner Mongolia University. His main research interests include computer network and distributed computing system, network protocol engineering, software testing theory and application, formal methods, especially Petri nets based software verification and validation methods.

Jiantao Zhou received her Ph.D. degree in computer science from Tsinghua University in 2005. She is currently a professor at the Inner Mongolia University. Her research interests include cloud computing, network software supported cooperative work, software testing theory and application, formal methods, especially Petri nets based software verification and validation methods.

Xiaoyu Song received his Ph.D. degree from the University of Pisa in 1991. He is currently a professor in the Department of Electrical & Computer Engineering at Portland State University. His main research interests include design automation, formal methods, and emerging technologies. In particular, He is currently focused on verification, hardware/software codesign for embedded systems, reversible circuits, and timing analysis. He served as an associate editor of IEEE Transactions on Circuits and Systems and IEEE Transactions on VLSI Systems.