A CPN-based Software Testing Approach

Lizhi Cai  
Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai, China  
Email: clz@ssc.stn.sh.cn

Juan Zhang 1 and Zhenyu Liu 2  
1Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai, China  
2School of Computer Engineering and Science, Shanghai University, Shanghai, China  
Email: {zhangj, lzy}@ssc.stn.sh.cn

Abstract—As a graphical and mathematical modeling tool, CPN (Colored Petri Net) is often used to describe the transition of states for an information system. The advantage of CPN model is that the model can be simulated dynamically. This paper presents an approach to generating test cases based on a transition graph of CPN model. The transition graph provides a solid basis for test cases generation in a form that can be easily manipulated. The case for vending machine illustrates the effectiveness of this method.

Index Terms—software testing, CPN, model-based, simulation

I. INTRODUCTION

Software testing is an important activity to assure the quality of software. Specification based testing, refers to the test method in which the generation of test cases is based on the system’s specification, without seeing an implementation of the information system. Furthermore, the test cases can be developed before the program even exists. The advantage of specification based testing is that the test cases can be independent of any particular implementation. In this case the software specifications are the main sources for generating test cases as these documents describe the software system to be developed in detail. Software specifications may be natural language description, formal language specification or formal model. The research on specification based testing focuses on test data generation methods for deriving test cases from formal specification[1,2,3]. Generating test cases based on specification can not only helps developer to test their program when they finish coding but also controls the developers to program the software as defined in the software specification.

Model based testing is one special specification based testing. In model based testing, the test cases can be developed from a model. The model can be thought as one formal specification, because it can provide clear high level description, sometimes in formal representations, for the software system. Usually, the specification described by a model can be validated and simulated. During a testing stage, software testers use test data as input to run the software program. The testers usually have an expected outcome from these data, which is also called test oracle. Testers have to compare between test oracle and actual result to decide whether a program executes correctly for the given test, and make a verdict of “pass” or “fail”. The construction the software’s oracle is labor-intensive and time-intensive in the process of test cases design. The simulation of the model can provide the oracle of the system effectively, which reduces one of the major costs of testing. Model based testing provides a solid foundation for generating test cases from a formal method. In the recent years with the development of the modeling techniques, such as UML (Unified Modeling Language), FSM (Finite State Machines), CPN (Colored Petri Net) and so on, the requirements of the software test automation, model based software testing gradually receives researchers’ attention[4,5,6].

In our approach, we focus on generate test data from CPN. Petri Net is originated from Carl Adam Petri’s dissertation [7], communication with automata, in 1962. A petri net is a graphical and mathematical modeling approach used to describe information processing systems that are characterized as concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic. CPN combines the strengths of ordinary petri net with the strengths of a high-level programming language together with a rigorous abstraction mechanism [8]. Petri net provides the primitives for process interaction. The CPN has a formal mathematical definition and a well-defined syntax and semantics. This formalization is the analysis foundation for the different behavioral properties [9, 10].

II. TESTING CRITERIA BASED ON CPN STATES SPACE

All the elements of petri net are strictly defined with normative semantics. The system model based on petri net is also very clear and strict. Petri net has sufficient ability to support all modeling primitives, and have been widely used in various fields of information system modeling analysis and control. Literature [13] gives the complete formal definition of a CPN as follows. The purpose of this definition is to give a mathematically sound and unambiguous description of a CPN.

Definition: A Colored Petri Net is a nine-tuple (Σ, P, T, A, N, C, G, E, I), where:
1) $\Sigma$ is a finite set of non-empty **types**, also called color sets. In the associated CPN Tool, these are described using the language CPN-ML. A **token** is a value belonging to a type.

2) $P$ is a finite set of **places**. In the associated CPN Tool these are depicted as ovals/circles.

3) $T$ is a finite set of **transitions**. In the associated CPN Tool these are depicted as directed edges. The sets of places, transitions, and arcs are pairwise disjoint, that is

$$ P \cap T = P \cap A = T \cap A = \emptyset. $$

4) $A$ is a finite set of **arcs**. In the associated CPN Tool these are depicted as directed edges. The sets of places, transitions, and arcs are pairwise disjoint, that is

$$ P \cap T = P \cap A = T \cap A = \emptyset. $$

5) $N$ is a node function. It is defined from $A$ into $T \times P \times T$. In the associated CPN Tool this depicts the source and sink of the directed edge.

6) $C$ is a color function. It is defined from $P$ into $\Sigma$.

7) $G$ is a guard function. It is defined from $T$ into expressions such that:

$$ \forall t \in T \exists (C(t) = \text{Bool} \land \text{Type}(\text{Var}(C(t))) \subseteq \Sigma) $$

8) $E$ is an arc expression function. It is defined from $A$ into expressions such that:

$$ \forall v \in \text{A}[\text{Type}(E(v))] = \text{C}[\text{C}(v)] $$

where $P$ is the place of $N(a)$ and $C(p)$MS denotes the multi-set type over the base type $C(p)$.

9) $I$ is an initialization function. It is defined from $P$ into closed expressions such that:

$$ \forall v \in \text{P}[\text{Type}(I(v))] = \text{C}[\text{I}(v)] $$

In the CPN Tool this is represented as initial marking next to the associated place.

In practice, a CPN model can be created using CPN tool [14] developed by University of Aarhus, Denmark. It is a graphical tool that allows one to create a visual representation of a CPN model. It is based on state machine theory and is an extension of place-transition petri net.

The vending machine program is a classic example for discussing in the software testing area as well as triangle program. The program simulates the behavior of a vending machine that issues drinks to customers. The specification is as follows.

1. A can of soft drink costs $0.15.
2. Only nickels and dimes are to be accepted as valid coins to a payment. If the sum of money has been accepted exceed $0.15, the machine will refuse more money.
3. The machine will return the correct change, if the customer insert too much money and in the same time the machine has the nickels to make change.

![State transition graph](image)

**Figure 1.** State transition graph.

Considering the node 3 in the Fig. 1, if the test set $T$ satisfies the transition-pair criterion, the path $P$ must include the following six subpathes: $1 \rightarrow 3 \rightarrow 3 (2) 1 \rightarrow 3 \rightarrow 5 (3) 2 \rightarrow 3 \rightarrow 3 (4) 2 \rightarrow 3 \rightarrow 3 (5) 4 \rightarrow 3 \rightarrow 3 (6) 4 \rightarrow 3 \rightarrow 5$. These subpathes cover the following pairs of transition $<(p_1,p_4),(p_1,p_5),(p_2,p_4),(p_2,p_5),(p_3,p_4),(p_3,p_5)>$.

## III. Modeling for Vending Machine

In general, there are infinitely many paths in state spaces and hence it is impossible to cover all these paths. Inspired by the traditional test data generation approach based on specification[5], this paper presents the following coverage criteria out of a potentially infinite family. Each of these criteria requires a different amount of testing. The state spaces can be represented as a directed graph. If $<S_0,S,T>$ is a state space, let $s_1 \in S$ and $s_0 \in S$ for $0 \leq i \leq n$. A sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n \rightarrow s_{n+1}$ is called a path if $s_0 \neq s_0$ and $s_1 \neq s_{n+1}$ for $0 \leq i \leq n - 1$. Let $P$ is a set of paths.

1. **State Coverage (SC):** The test set $T$ satisfies State Coverage if the path $P$ created by $S$ includes every $s \in S$.
2. **Transition Coverage (TC):** The test set $T$ satisfies Transition Coverage if the path $P$ created by $S$ includes every $t \in T$.
3. **Transition Pair Coverage (TPC):** The test set $T$ satisfies Transition Pair Coverage if the path $P$ created by $S$ includes every pairs of transition linked by a $s \in S$.
(4) In the case there are not enough coins in the machine, the machine will show the information “Inadequacy”.

(5) In the case there are not enough soft drink in the machine, the machine will show the information “NoDrink”.

(6) In the case there are not enough change in the machine, the machine will show the information “NoChange”.

(7) When the DrkBtn is pushed, a can of soft drink will be dispensed only if it is available and the payment is sufficient. And at same time the machine will return the correct change.

(8) When the BkBtn is pushed, all the coins just inserted into the machine by customer will be return.

There are three transitions in the model which respectively stand for different user action: insert the coins, push the drink button and push the change button. The machine has three outputs: coin slot used to send the change out, drink slot used to send the drink out, and information windows used to show the state information of machine. The model of vending machine is shown in the Fig. 2. The different characteristics of vending machine can be modeled very easily and explicitly by defining appropriate types for the associated tokens and function declarations using CPN ML which is based on the functional programming language ML. In the CPN model of each color corresponds to a data field. In order to denote the different type in the vending machine such as coin number, drink number and state information, we introduce the following color sets:

```
colset INFOSET = with Ready | Inadequacy | NoChange | NoDrink;
colset NickelNum=int;
colset DimeNum=int;
colset CoinNum=product NickelNum*DimeNum;
colset DrkNum=int;
```

The color set INFOSET is a enumerate type which include four different values. These different values show the different states presented by specification. To simplify the description of the problem and provide the convenience for discussion, we define CoinNum color set which combination nickel and dime. The value (1,0) means there is only a nickel and (0,1) means there is only a dime. The value (1,1) means there are a nickel and a dime at same time. All the places in the CPN model will contain data input by user by GUI Components or internal data as shown table1. Each place will be associated with appropriate types defined by the color sets as shown in table 1.

<table>
<thead>
<tr>
<th>No.</th>
<th>PlaceName (Type)</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CoinInsert_Slot (CoinNum)</td>
<td>It is used to store the coins which will be inserted into the vending machine.</td>
</tr>
<tr>
<td>2</td>
<td>CoinCache (CoinNum)</td>
<td>It is used to store temporarily the coins which have been inserted into the vending machine.</td>
</tr>
<tr>
<td>3</td>
<td>Coin_Pool (CoinNum)</td>
<td>It is the place used to store the coins which have been really accepted by the vending machine.</td>
</tr>
<tr>
<td>4</td>
<td>Drk_Pool (DrkNum)</td>
<td>It is the place used to store the soft drink in the vending machine.</td>
</tr>
<tr>
<td>5</td>
<td>Coin_Slot (CoinNum)</td>
<td>It is used to store the change or refundment returned by the vending machine.</td>
</tr>
<tr>
<td>6</td>
<td>Drk_Slot (DrkNum)</td>
<td>It is used to store the soft drink delivered by the vending machine.</td>
</tr>
<tr>
<td>7</td>
<td>InfoWindow (INFOSET)</td>
<td>It is used to show the state information for user.</td>
</tr>
</tbody>
</table>
All the actions are executed by the SML functions associated to each transition of the processes. The function start chk_insert is responsible for checking the validity of number which a user inserts the vending machine. The source code bellow shows such a function.

```sml
fun chk_insert(n1:NickelNum,d1:DimeNum, n2:NickelNum,d2:DimeNum)=
  if(
    (n1+n2<=3 andalso d1+d2=0) (*only nickels*)
  orelse (n1+n2=1 andalso d1+d2=1)
  (*one dime, not more than one nickel *)
  orelse (n1+n2=0 andalso d1+d2=2))
  (*two dimes, no nickel*)
  then
    (n1+n2,d1+d2)
  else
    (n2,d2);
```

When a user inserts the coins and pushes the button DrkBt, the vending machine will update the coin number of the place Coin Cache. If the transaction is successful, the number of Coin Cache will be set to zero. The three cases described in specification 5, 6 and 7 will lead to transaction failure. The following is source code of the function update_coinc.

```sml
fun update_coinc(n:NickelNum,n3:NickelNum, d:DimeNum,d3:DimeNum,drk:DrkNum)=
  if (drk=0 orelse (( (n*5+d*10) > PRICE) andalso n3=0)
    orelse n*5+d*10< PRICE)
  then
    1`(n,d)
  else
    1`(0,0);
```

The function pr_info is used to show the information of vending machine after a user push the drink button. If none of coins are inserted the machine, the state information will do not change. Otherwise the machine will show “NoDrink” if there is not any drink in the machine. The information “NoChange” and “Inadequacy” are shown respectively according to the number of changes in the machine and the sum of money inserted by a user.

```sml
fun pr_info(n:NickelNum,n3:NickelNum,d:DimeNum, drk:DrkNum,info:INFOSET)=
  if(n=0 andalso d=0) then info
  else
    if (drk=0) then  NoDrink
    else
      if (n3=0) then       NoChange
      else
        if (n*5+d*10<PRICE) then    Inadequacy
        else
          Ready;
```

The changes will minus one if a user buy a can of drink successfully use two dimes. In the case, the machine has enough changes and drink. If a user inserts neither more nor less than $.15, the machine will accept all the coins. Otherwise the coins in the pool will hold the line.

```sml
fun update_coinp(n:NickelNum,n3:NickelNum, d:DimeNum,d3:DimeNum,drk:DrkNum)=
  if (((n*5+d*10)>PRICE) andalso (n3>0) andalso (drk>0))
    then
      1`((n3-1,d3+d) (*need change*)
    else
      if (((n*5+d*10)==PRICE) andalso (drk>0))
        then
          1`(n3+n,d3+d)
        else
          1`(n3,d3) (* can not the drk*);
```

The function update_chg is used to send change to output slot. When a user does not buy the drink successfully, the machine will do not change the slot. Otherwise, the machine will use the new state to update the change slot.

```sml
fun update_chg(n:NickelNum,n1:NickelNum, n3:NickelNum, d:DimeNum,d1:DimeNum,d3:DimeNum, drk:DrkNum)=
  if (n3=0 orelse drk=0 orelse (n*5+d*10<PRICE))
    then
      1`(n1,d1)
    else
      if ((n*5+d*10)==PRICE )
        then
          1`(1,0)
        else
          1`(0,0);
```

In the same way, we can define all function for CPN model of the vending machine.

IV. SYSTEM STATE SIMULATION AND VALIDATION OF THE CPN MODEL CORRECTNESS

Relative to the FSM, UML or other model, CPN model has the advantage of dynamic simulation and automated analysis of the state space. Suppose a user want to buy the soft drink, but he has only a nickel. After he inserts the nickel, the nickel runs from his wallet to the cache in the vending machine. This process is shown in Fig. 3.

(a) initial marking
(b) marking after inserting a nickel

Figure 3. The process simulation for inserting a nickel.

When the user pushes the Drink Button after he inserted the nickel, The machine will change the information from “Ready” to “Inadequacy” as Fig. 4(a) and Fig. 4(b).
The action can be simulated by firing the transition Push_BkBtn. The coins in the coin cache place will be sent out from coin slot. The state of the vending machine come back to “Ready” from “Inadequacy” as shown in Fig. 5(a) and Fig. 5(b).

Just for funny the user pushes the Drink Button or Refundment Button before he inserts the nickel. We can fire the transition Push_BkBtn or Push_DrkBtn before firing the transition Insert Coin to simulation this situation. It is apparent that he could not get anything.

Simulation can only be used to consider a finite number of executions of the model being analyzed. This makes simulation suited for detecting errors and for obtaining increased confidence in the correctness of the model, and thereby the system. Full state spaces represent all possible executions of the model being analyzed. In CPN tools all the state will be explored by state space tools. The state space report provides some basic information about the size of the state space and standard behavioral properties of the CPN model. The part properties of the vending machine are shown in Fig. 6. From the report we can know that there is no dead markings and dead transition instance. This means all the state is reachable. This is an important evidence of the correctness of the model.

V. GENERATING TEST CASES BASED ON CPN

The state space of the vending machine with a nickel is shown in Fig. 7. Each node in the state space is inscribed with three integers. The topmost integer is the node number and the two integers separated by a colon give the number of predecessor and successor nodes. Node 1 corresponds to the initial marking, and the Fig. 7 shows all markings reachable by the occurrence of at most three binding elements starting in the initial marking.

A state is composed by the token of the all place in the CPN model. The fires which change the state of CPN model is shown in the bottom of the Fig. 7. Each line represents the conditions and the results of one change. The initial state and the termination state are also being express in the same line. The information provides the detail of model which is basis for generating test cases on the model.
node 3 can be visited by two different test sequences. The corresponding test cases are displayed in table II and table III.

![Figure 8. State cover tree](image)

**TABLE II.** TEST CASES FOR SCT IN FIGURE 8(A)

<table>
<thead>
<tr>
<th>No.</th>
<th>Covered States</th>
<th>Transition Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,4,5</td>
<td>Insert_Coin, Push_DrkBtn, Push_BkBtn, Push_DrkBtn</td>
</tr>
</tbody>
</table>

![Figure 9. the transition coverage tree](image)

**TABLE III.** TEST CASE FOR SCT IN FIGURE 8(B)

<table>
<thead>
<tr>
<th>No.</th>
<th>Covered States</th>
<th>Transition Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,5</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>2</td>
<td>1,2,4</td>
<td>Insert_Coin, Push_DrkBtn</td>
</tr>
</tbody>
</table>

The number of arc is far greater than the number of nodes in a state space. So the State Coverage criterion is relatively weak coverage criterion. The testing tree shown in Fig. 9 is created by Transition coverage Criterion. The transitions that do not change the state are also be tested. For example, in the initial state, transitions Push_DrkBtn and Push_BkBtn just generation the arc from node 1 to itself. The corresponding test cases are displayed in table IV.

In the same way, the pair transition coverage tree can be generated as shown in Fig. 10. The corresponding test cases are shown in table V.

![Figure 10. The transition pair coverage tree](image)

**TABLE IV.** TEST CASES FOR TCT IN FIGURE 8(B)

<table>
<thead>
<tr>
<th>No.</th>
<th>Covered Transition</th>
<th>Transition Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Push_DrkBtn</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Push_BkBtn Push_DrkBtn</td>
</tr>
<tr>
<td>3</td>
<td>1 Y 2Y</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>4</td>
<td>1 Y 2Y</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>5</td>
<td>1 Y 2Y</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>6</td>
<td>1 Y 2Y</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
</tbody>
</table>

**TABLE V.** TEST CASES FOR TCT IN FIGURE

<table>
<thead>
<tr>
<th>No.</th>
<th>The center node of Transition pair</th>
<th>Transition Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Push_DrkBtn, Insert_Coin</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Push_BkBtn, Insert_Coin</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Insert_Coin, Push_BkBtn</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Insert_Coin, Push_DrkBtn</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Insert_Coin, Push_DrkBtn</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Insert_Coin, Push_DrkBtn</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>Insert_Coin, Push_BkBtn</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>Insert_Coin, Push_BkBtn, Push_DrkBtn</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>Insert_Coin, Push_DrkBtn, Push_BkBtn</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>Insert_Coin, Push_DrkBtn, Push_BkBtn</td>
</tr>
</tbody>
</table>
VI. CONCLUSIONS

Model based testing is one special specification based testing technique. CPN combines the strengths of ordinary petri net with the strengths of a high-level programming language together with a rigorous abstraction mechanism. It is often used in system simulation and analysis. This paper presents a model-based approach to generation test cases using CPN. The vending machine is taken as a classic case to build CPN model. All the states in the state space of the model can be generated automatically by the CPN tools. Three coverage criteria have been presented in this paper for the generation of test cases. The test cases generation for vending machine illustrates the effectiveness of the method.

REFERENCES


Lizhi Cai was born in Zhejiang Province, China, 1972. He is currently a associate professor in Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai, China. He is the member of Special Interest Group of Software Engineering of CCF. He received his Ph.D. degree in computer application from Shanghai University. His current research interests include quality model, software process, software testing etc.