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]
[1]Shanghai Key Laboratory of Computer Software Evaluating & Testing, Shanghai, China
[2]School 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 work 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 ($\sum$, P, T, A, N, C, G, E, I), where:

This work is supported by National Torch Project of China（No. 2009GH510068 ） and fund of Science and Technology Commission of Shanghai Municipality (No. 10DZ2291800).

doi:10.4304/jsw.6.3.468-474

1) $\sum$ 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 rectangles.
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 $T \times P \cup T \times P$. 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 $\sum$.
7) G is a guard function. It is defined from T into expressions such that:

$$\forall t \in T[Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \sum]$$

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

$$\forall a \in A[Type(E(a)) = C(p)_{ms} \wedge Type(Var(E(a))) \subseteq \sum]$$

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:

$$\forall p \in P[Type(I(p)) = C(p)_{ms}]$$

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.

All the markings in the places of a CPN constitute the state of a system being modeled. The dynamic behavior of an information system is simulated in terms of the firing of transitions. The fire of the transition of model presents the reception of a request of a specific operation such as the receiving the input data, or a user action such as push down the button. The firing of a transition makes the system transform from one state into another. All the states of a system model constitute its state space. The state spaces are calculated fully automatically by the CPN state space tool using a state space construction algorithm[14]. The CPN tool supports a number of stop and branching options that makes it possible for the user to control the state space generation. The state space is a tuple <S_0,S,T> such that:

- $S_{0 \in} S$ is the initial state which is composed of all the initial markings in the CPN.
- S is the set of all states in state spaces. All the elements of S will be reached by the transition fire sequence from S_0.
- T is the set of transitions. All the elements of T will be enabled if the associated arc expressions of all incoming arcs can be evaluated to a multi-set, compatible with the current tokens in their respective input places, and its guard is satisfied.

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 $s_i \in S$ and $t_i \in T$ for $0 \leq i \leq n$. A sequence $(s_0, t_0), (s_1, t_1), \cdots, (s_{n-1} t_{n-1}, s_n$ is call a path if $s_0=S_0$ and $s_i \xrightarrow{t_i} s_{i+1}$ for $0 \leq i \leq n-1$. Let P is a set of paths.

(1) State Coverage (SC): The test set TS satisfies State Coverage if the path P created by TS includes every $s \in S$;
(2) Transition Coverage (TC): The test set TS satisfies Transition Coverage if the path P created by TS includes every $t \in T$;
(3) Transition Pair Coverage (TPC): The test set TS satisfies Transition Pair Coverage if the path P created by TS includes every pairs of transition linked by a $s \in S$;

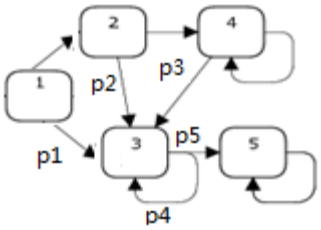


Figure 1. State transition graph.

Considering the node 3 in the Fig. 1, if the test set TS satisfy the transition-pair criterion, the path P must include the following six subpathes: 1-> 3 ->3 (2) 1->3->5 (3) 2->3->3 (4) 2->3->5 (5) 4->3->3 (6) 4->3->5. These subpathes cover the following pairs of transition :(p1,p4),(p1,P5),(p2,p4),(p2,p5), ,(p3,p4),(p3,p5).

III. MODELING FOR VENDING MACHINE

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 $.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 $.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.

(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.

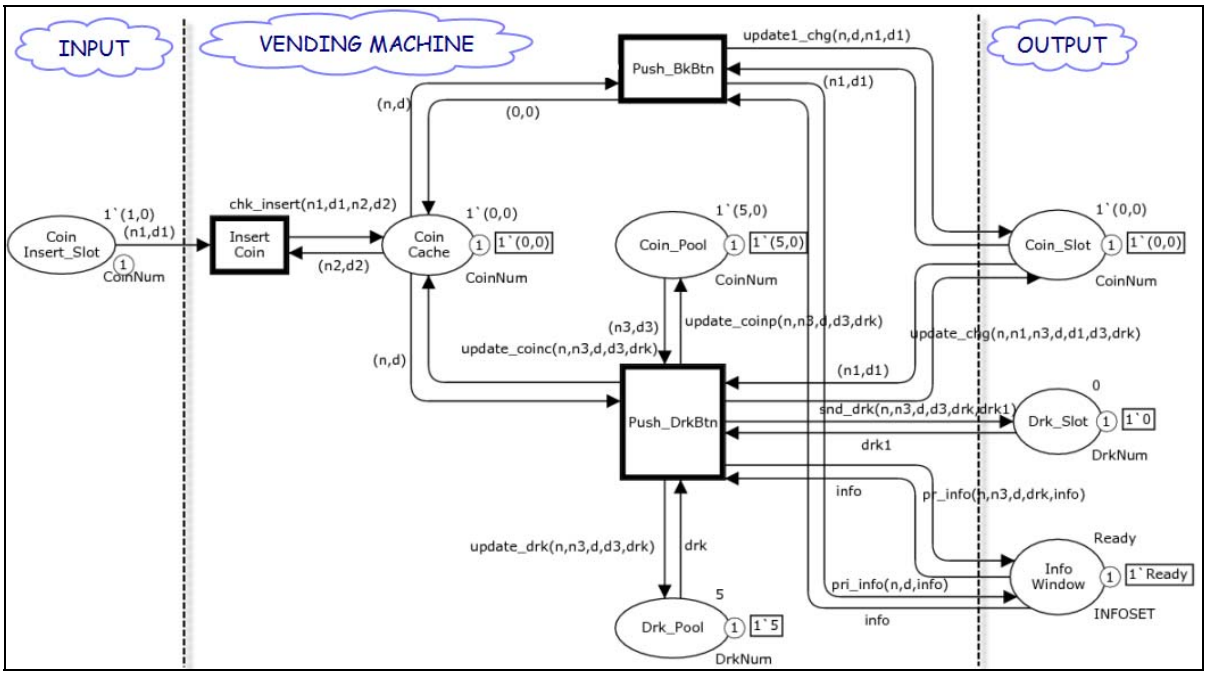


Figure 2. The CPN Model of Vending Machine

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 I. THE PLACES IN THE CPN MODEL

No.	PlaceName (Type)	Purpose
1	Coin Insert_Slot (CoinNum)	It is used to store the coins which will be inserted into the vending machine.
2	CoinCache (CoinNum)	It is used to store temporarily the coins which have been inserted into the vending machine.
3	Coin_Pool (CoinNum)	It is the place used to store the coins which have been really accepted by the vending machine.
4	Drk_Pool (DrkNum)	It is the place used to store the soft drink in the vending machine.
5	Coin_Slot (CoinNum)	It is used to store the change or refundment returned by the vending machine.
6	Drk_Slot (DrkNum)	It is used to store the soft drik delivered by the vending machine.
7	InfoWindow (INFOSET)	It is used to show the state information for user;

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.

```
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 DrkBtn, 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.

```
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.

```
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.

```
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.

```
fun update_chg(n:NickelNum,n1:NickelNum,
n3:NickelNum,   :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.

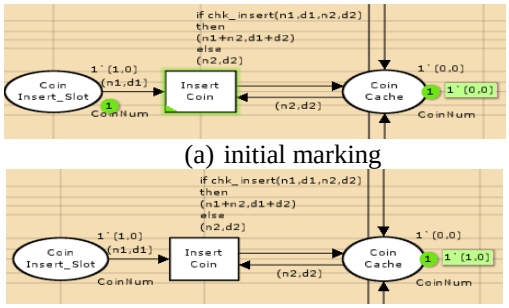
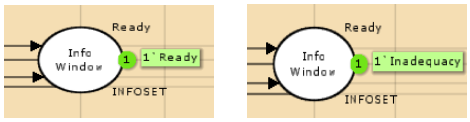


Figure 3. The process simulation for inserting a nickel.

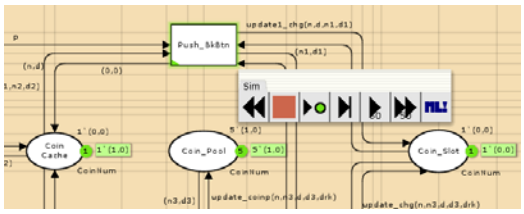
When the user pushes the Drink Button after he inserted the nickel, The machine will change the information from “Ready” to “Inadequacy” as Fig. 4(a) and Fig. 4(b).



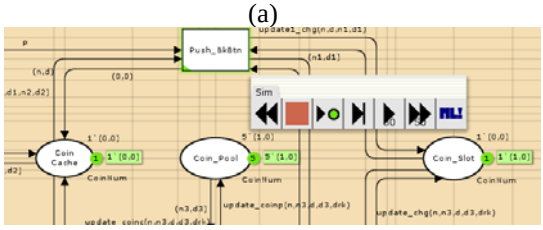
(a) (b)

Figure 4. The information change after pushing the Drink Button.

After that the user pushes the Refundment Button. 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).



(a)



(b)

Figure 5. The state change after BkBtn.

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.

```
Statistics
------------------------------------------

State Space                 Scc Graph
   Nodes:  5                   Nodes:  5
   Arcs:   11                  Arcs:   5
   Secs:   0                   Secs:   0
   Status: Full

Home Properties
------------------------------------------
  Home Markings                 [5]

Liveness Properties
------------------------------------------
  Dead Markings                 None
  Dead Transition Instances     None
  Live Transition Instances
     Page1'Push_BkBtn 1
     Page1'Push_DrkBtn 1

Fairness Properties
------------------------------------------
       Page1'Insert_Coin 1    No Fairness
       Page1'Push_BkBtn 1     No Fairness
       Page1'Push_DrkBtn 1    No Fairness
```

Figure 6. The properties of CPN model.

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.

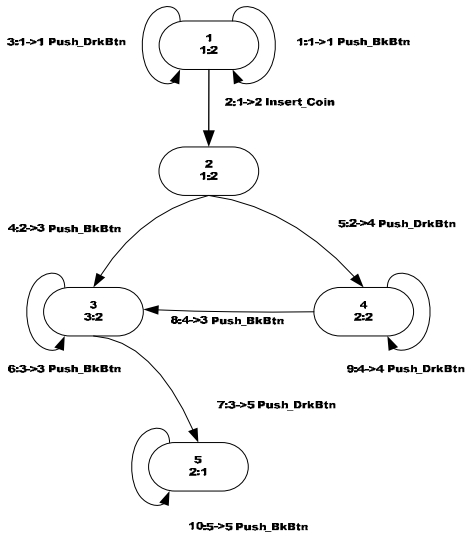


Figure 7. The State space of bending machine with only a nickel

The State Coverage requires that all the states in the state spaces should be covered by the test cases. According to section 3, we can get the state cover tree for state space shown in Fig 8(a). There are more than one test suite satisfy the State Coverage Criteria. Fig. 8(b) is another test tree to cover all the five states because

node 3 can be visited by two different test sequences. The corresponding test cases are displayed in tableII and table III.

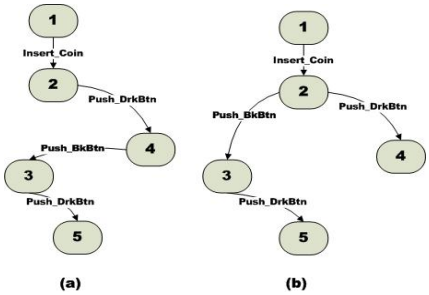


Figure 8. State cover tree

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

No.	Covered States	Transition Sequence
1	1,2,3,4,5	Insert_Coin Push_DrkBtn Push_BkBtn Push_DrkBtn

TABLE III. TEST CASE FOR SCT IN FIGURE8 (B)

No.	Covered States	Transition Sequence
1	1,2,3,5	Insert_Coin Push_BkBtn Push_DrkBtn
2	1,2,4	Insert_Coin Push_DrkBtn

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.

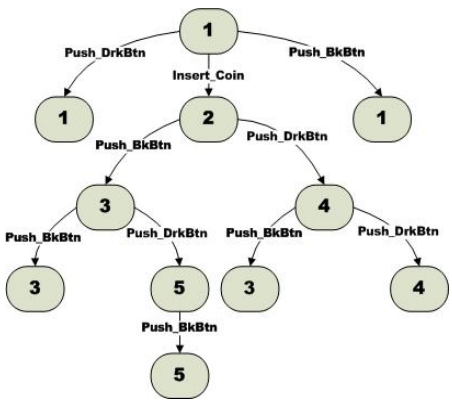


Figure 9. the transition coverage tree

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

No.	Covered Transition	Transition Sequence
1	1Y 1	Push_DrkBtn
2	1Y 1	Push_BkBtn
3	1Y 2Y 3Y 3	Insert_Coin Push_BkBtn Push_BkBtn
4	1Y 2Y 3Y 5Y 5	Insert_Coin Push_BkBtn Push_DrkBtn Push_BkBtn
5	1Y 2Y 4Y 3	Insert_Coin Push_DrkBtn Push_BkBtn
6	1Y 2Y 4Y 4	Insert_Coin Push_DrkBtn Push_DrkBtn

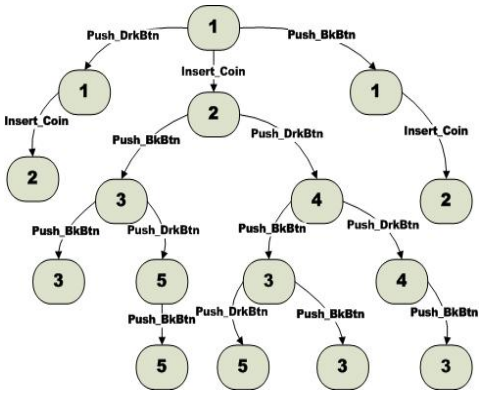


Figure 10. The transition pair coverage tree

TABLE V. TEST CASES FOR TCT IN FIGURE

No.	The center node of Transition pair	Transition Sequence
1	1	Push_DrkBtn Insert_Coin
2	1	Push_bkBtn Insert_Coin
3	2	Insert_Coin Push_BkBtn
4	2	Insert_Coin Push_DrkBtn
5	4	Insert_Coin Push_DrkBtn Push_BkBtn
6	4	Insert_Coin Push_DrkBtn Push_DrkBtn
7	3	Insert_Coin Push_BkBtn Push_BkBtn
8	3	Insert_Coin Push_BkBtn Push_DrkBtn
9	3	Insert_Coin Push_DrkBtn Push_BkBtn Push_DrkBtn
10	3	Insert_Coin Push_DrkBtn Push_BkBtn Push_BkBtn
11	5	Insert_Coin Push_DrkBtn Push_BkBtn Push_DrkBtn

12	5	Insert_Coin Push_BkBtn Push_DrkBtn Push_BkBtn

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 cases 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

[1] G. Bernot, M. C. Gaudel, and B. Marre. Software testing based on formal specification: A theory and a tools, *Software Engineering Journal*, 6(6), pp387-405,1991.

[2] Robert M. Hierons. Testing from Z specification. *The Journal of Software Testing, Verification, and Reliability*, 7, pp19-33, 1997.

[3] G. Laycock. Formal specification and testing: A case study. *The Journal of Software Testing, Verification, and Reliability*, 2, pp7-23,1992.

[4] Supaporn Kansomkeat, Jeff Offutt, Aynur Abdurazik and Andrea Baldini. A Comparative Evaluation of Tests Generated from Different UML Diagrams. *Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2008)*, pp867-872, Phuket Thailand, August 2008.

[5] Jeff Offutt, Shaoying Liu, Aynur Abdurazik and Paul Ammann. Generating Test Data From State-based Specifications. *The Journal of Software Testing, Verification and Reliability*, 13(1), pp25-53, March 2003

[6] Jeff Offutt and Aynur Abdurazik. Using UML Collaboration Diagrams for Static Checking and Test Generation. *The Third International Conference on the Unified Modeling Language (UML '00)*, pp383-395, York, UK, October 2000

[7] Carl Adam "Petri. Communication With Automata". *Tech. Rep. RADC TR-65-377*, Rome Air Dev. Center. New York, 1966.

[8] Ratzer A. V, Wells L, Lassen H. M, et al. "CPN Tools for Editing Simulating and Analysing Coloured Petri Nets". *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, Eindhoven, The Netherlands: Springer-Verlag, 2003. 450-462.

[9] Kyller Gorgˆonio, Fei Xia. Modeling and verifying asynchronous communication mechanisms using coloured Petri nets. The 8th International Conference on Application of Concurrency to System Design, 2008. ACSD 2008. pp138 – 147

[10] Vijay Gehlot, Thomas Way, Robert Beck and Peter DePasquale. Model Driven Development of a Service Oriented Architecture (SOA) Using Colored Petri Nets. *First Workshop on Quality in Modeling, ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Models* 2006, Genova, Italy, pp63-77, October 1-6, 2006

[11] Guy Helmer, Johnny Wong and Mark Slagell et al. Software fault tree and coloured Petri net–based specification, design and implementation of agent-based intrusion detection systems. *International Journal of Information and Computer Security* 1(2), pp. 109 – 14, 2007

[12] F. Gottschalk, M. H. Jansen-vullers, H. M. W. Verbeek. Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes. *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*. pp95-*110, 2008*

[13] K. Jensen. An Introduction to the Theoretical Aspects of Coloured Petri Nets. *A Decade of Concurrency, Lecture Notes in Computer Science* vol. 803, Springer-Verlag , pp.230-272, 1994.

[14] A. V. Ratzer, et al., CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets.*Proceedings of the 24th International Conference on the Application and Theory of Petri Nets (ICATPN 2003).* 2003, pp450-462.

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.