Colored Petri Nets Model based Conformance Test Generation

Jing LIU
Institute of Computing Technology,
Chinese Academy of Sciences
email: liujing@ict.ac.cn

Xinming YE
Inner Mongolia University
Hohhot, China
email: xmy@imu.edu.cn

Jun LI
Institute of Computing Technology,
Chinese Academy of Sciences
email: lijun@ict.ac.cn

***Abstract*—A novel Colored Petri Nets (CP-nets) model based test case generation approach is proposed to makes the best of advantages of the ioco testing theory and the CP-nets modeling, where the Conformance Testing orientated CP-nets (CT-CPN) is proposed for modeling certain software systems, and PN-ioco relation is defined as a new conformance relation, and finally test cases are generated through simulating the system CT-CPN models. CP-nets model simulation based test generation approach reflects the data-dependent control flow of the system behaviors, so all test cases are completely feasible for the actual test executions. Besides, better formal modeling and analytic capabilities in CP-nets modeling quite facilitate validating the accuracy of the system CT-CPN model. For effectively extending the applicability of the Petri nets based testing technologies, our novel CT-CPN model based test generation approach may well become a competent choice.**

Keywords- test case generation; colored Petri nets; ioco conformance; model simulation

I. INTRODUCTION

To promote the efficiency and effectiveness of the test generation in conformance testing, the Model Based Testing (MBT) technology is introduced [1,2]. It allows for generation of test cases with test oracles from a formal model that specifies the software behaviors explicitly, which improves the low-level efficiency and inaccuracy of the manual test case generation process. Towards the black-box conformance testing for the reactive software, the input-output conformance (ioco) relation based testing approach [3,4] is well-established and feasible, where the practical test execution is performed through input actions which are initiated and controlled by the tester, and output actions which are initiated and controlled by the software implementation itself.

In this paper, the ioco testing theory is concretized and instantiated with the colored Petri nets (CP-nets) [5] and a novel CP-nets model based test case generation approach is proposed. Compared with the test generation approach based on LTS models in the original ioco testing theory, our CP-nets model based test generation approach could introduce several advantages. First, CP-nets has better formal capabilities to specify and analyze complicated and concurrent system behaviors, which is quite helpful for validating the accuracy of the software CP-nets models. Second, CP-nets model could be simulated dynamically, directed by the data-dependent control flow of system behaviors. So, test cases, generated through such model simulation process, are predestined to be totally feasible for practical test executions. In a word, integrating the ioco relation based test generation approach with CP-nets is quite promising, and the novel CP-nets model based test generation approach could be taken as a competent and effective choice.

Our CP-nets model based test generation approach is composed of three related parts. First, Conformance Testing oriented CP-nets (CT-CPN) is proposed as the basic formal models for specifying software functionalities and implementation behaviors (in section II). Then, the new PN-ioco relation is defined to specify precisely what it means for an implementation to conform to its specifications (in section III). Finally, the test case generation algorithm is proposed through simulating the CT-CPN model to guarantee that all test cases are feasible for the practical test executions (in section IV).

II. CT-CPN MODELING

As the central idea of the ioco testing theory is to compare all external visible actions between the predefined system model and the actual system implementation during test executions, we need to propose a new kind of CP-nets model, which is able to accurately and explicitly specify such external actions. Herein, the CT-CPN is proposed to resolve the preceding problems, i.e., $CT\text{-}CPN_S$ is proposed as the formal model for the system specification modeling, and $CT\text{-}CPN_I$ as the formal model for the system implementation modeling.

Definition 1. A $CT\text{-}CPN_S$ is a triple (CPN, P_S, T_S):

1) CPN is 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) $CT\text{-}CPN_S$ has finite output, and does not have infinite sequences of internal actions.

In $CT\text{-}CPN_S$ models, an observable place is always the post-set of an input transition or an output transition to display what data (via token data) should be observed after firing these transitions. The input transitions model

input actions that accept input data provided by testers, and the output transitions model output actions that produce external-visible output observations by the implementations. So, observable places and input/output transitions are used together to explicitly specify the external behaviors of a certain software system. Besides, the last conditions in Definition 1 guarantee that the system produce finite and observable output results.

The CPN model for the SFDP system is presented in figure 1. In this protocol system, file data packets will be downloaded sequentially controlled by a packet number. When the file downloader gets a correct data packet, it increases the packet number, and then sends it to the file sender as an acknowledgement to require a new data packet with that packet number.

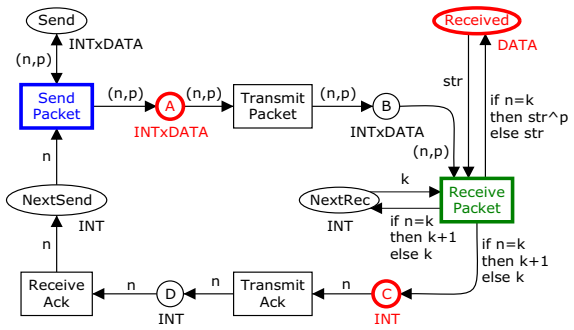

Figure 1. The CT-CPN$_S$ model for the SFDP system.

As an example, the CT-CPN$_S$ model for the Simplified File Downloading Protocol (SFDP) system is presented in figure 1. In the SFDP, file data packets will be downloaded sequentially controlled by a packet number. When the file downloader gets a correct data packet, it increases the packet number by one, and then uses it to require a new data packet. In figure 1, *A*/*C*/*Received* are observable places. *SendPacket* is an input transition and *ReceivePacket* is an output transition. The rest are internal places and transitions. If *SendPacket* fires, we could observe which data packet is sent via the tokens in *A*. If *ReceivePacket* fires, we could both observe which data packet is downloaded via the tokens in *Received*, and which packet number should be sent via the tokens in *C*.

As the system implementations are actual physical thing, i.e. software, hardware, or a hybrid system, rather that the formal objects, the test hypothesis [4] assumes that every system implementation correspond to an a-priori formal model, but these formal models cannot be explicitly constructed. Therefore, we propose CT-CPN$_I$ to just formally specify the system implementations.

Definition 2. A CT-CPN$_I$ is a triple (*CPN*, P_I, T_I):
1) *CPN* is a basic colored Petri nets model;
2) $P_I^O = P_S^O, T_I^I = T_S^I, T_I^O = T_S^O$: coupled CT-CPN$_I$ and CT-CPN$_S$ have the same observable places and external transitions;
3) $\{\delta\} \subset T_I$: δ is the **suspension transition**.

The suspension transition needs further explanation. The special quiescence output represents that an implementation has no visible output because it is just waiting for an input to proceed. Producing the quiescence is a kind of special output action, and modeled as the suspension transition δ. Firing a suspension action indicates that the implementation stays in the same state and needs input data as a trigger to continue execution.

The CT-CPN$_S$ based specification modeling for a software system is the first and the most significant step, because an accurate and well understandable model is crucial to the MBT technologies.

III. PN-IOCO DEFINITION

The system behaviors are simulated with the specific data in the CT-CPN context, and consequently the conformance relation should also be determined according to the specific data. Thus, we propose the PN-ioco relation as a new implementation relation to precisely specify what it means for an implementation to conform to its functional specification. The CP-nets model related concepts are referred to the CP-nets book [5].

Definition 3. *PN-ioco* is a binary relation with **ss**$\in$CT-CPN$_S$ and **ii**$\in$CT-CPN$_I$:
ii *PN-ioco* **ss** $=_{def} \forall \sigma \in SPtrace(M_S)$:
outtoken$(M_I$ ***fires*** $\sigma) =$ ***outtoken***$(M_S$ ***fires*** $\sigma)$.
1) $SPtrace(M_S) =_{def} \{\sigma \in (BE(T_S) \cup \delta)^* \mid M_S \stackrel{\sigma}{\Rightarrow}\}$; it enumerates all traces of the model **ss**, including the suspension transitions. M_S and M_I are initial markings, respectively.
2) ***outtoken***$(M) =_{def} \{M(P) \mid P \in P_S^O \cup P_I^O\}$; it represents the observable output token data. In the model **ss**, it records the token data of current observable places under a specific marking. In the model **ii**, it corresponds to the actual observable output data produced by the system implementations during the test execution.

Guided by our *PN-ioco* definition, the conformance is determined by comparing token data in the observable places along a specific *SPtrace* with the actually observed output from the implementation. If they are the same, we could determine that the implementation executes as expected. However, if the actual output data are different from what prescribed in the **ss** model, we could conclude with the non-conformance.

IV. SIMULATION BASED TEST GENENRATION

The intuitive idea of our test generation approach is essentially a traversal of the system CT-CPN$_S$ models. This kind of traversal is performed by model simulation execution under given initial markings. That is, a specific initial marking could conduct simulating the CT-CPN$_S$ model once, and during the simulation process, feasible testing sequences are constructed and the data-dependent test oracles are added for validating the actual observations with respect to the prescribed output.

As no further transitions are enabled, the model simulation will be terminated, and finally a corresponding test case model is generated. To present the feasibility and effectiveness of the preceding test case generation approach, we adopt the SFDP system as a representative to demonstrate the practical test case generation and test execution procedures.

Figure 2 presents our test case generation algorithm, which is composed of three major steps.

Given: s∈CT-CPN$_S$, and M_0 as its initial marking

$\exists\sigma \in BE(T)^*, \sigma = (t_0,b_0),(t_1,b_1)...(t_{k-1},b_{k-1}) : M_0 \xrightarrow{(t_0,b_0)} M_1 \xrightarrow{(t_1,b_1)} ... \rightarrow M_k$

Proc GenTestCase {
/* beginning with the first firing transition */
t∈ T_S : t=t_0 and ∀p∈pre(t); M=M_0;
/* Insert suspension transition */
tcg: if t∈ T_S^I
{ t∈ T_{TS}^I; p∈ P_{TS}^I; ***Insert*** t_s∈ T_{TS}^{δ}: pre(t_s) = post(t_s) = p; }
/* Insert test oracle and verdict units*/
if p∈ P_S^O
{ p∈ P_{TS}^O; ***Insert*** p_s∈ P_{TS}^{TO}: M(p_s) = M(p);
Insert t_v∈ T_{TS}^V and p_v∈ P_{TS}^V: pre(t_v) = p∪ p_s; post(t_v) = p_v; }
/* eliminate internal places */
if t∈ T_S^O
{ t∈ T_{TS}^O; if ∃pe∈ P_S^E: pre(t) = post(t) = pe ***Eliminate*** pe; }
/* eliminate internal transitions */
if t∈ T_S^e and ∃po∈ P_S^O: po∈pre(t) and ∃qe∈ P_S^E: qe∈post(t)
{ ***Fire*** t with b that (t, b)∈σ;
if | post(t) | >1 ***Integrate*** post(t) ***into*** qe;
Merge(po, qe): post(po) = post(qe); pre(po) = pre(qe);
Eliminate t; }
/* fire t to proceed */
Firing t with b that (t, b)∈σ: $M \xrightarrow{(t,b)} M'$; $M \Leftarrow M'$; $t \Leftarrow t'$ and p∈pre(t);
Goto tcg ***until*** M= M_k;
}// end of GenTestCase

Figure 2. The test case generation algorithm.

- ***Insert suspension transitions***

As for an input transition, every place in its pre-set becomes an input place in the test case, and suspension transition t_s is added to each input place for specifying the allowance for observing the suspension. However, if the token data in an input place cannot be controlled externally by the tester, no quiescence is allowed in this place, and we do not insert the suspension transition.

- ***Insert test oracle and verdict units***

Every observable place in model *s* becomes a corresponding observable place in the test case, and its coupled test oracle place p_s is inserted. The token data produced in p are copied to p_s. When this test case is applied in the practical testing, p will store actual output data produced by an implementation, and previously stored token data in p_s will act as its valid test oracles. Besides, we insert a corresponding test verdict transition t_v which takes p and p_s as its pre-set and a test verdict place p_v is newly added as its post-set. They work coordinately as an integrated unit to check whether token data in p and p_s are consistent. Only *pass* or *fail* tokens are valid in the test verdict place. So, if one *fail* token appears in this test verdict place, it is determined that the implementation does not pass this test case.

- ***Eliminate internal transitions and places***

Because internal actions cannot be observed externally via input and output in real black box testing executions, we need eliminate the internal places and transitions during the test generation. However, the elimination should guarantee no harm to system functionalities, i.e., the data-dependant control flow of the system behavior should be kept. Based on different transition scenarios, internal or external (input/output), three model structures are proposed in figure 3 as basic elimination units together with corresponding elimination rules.

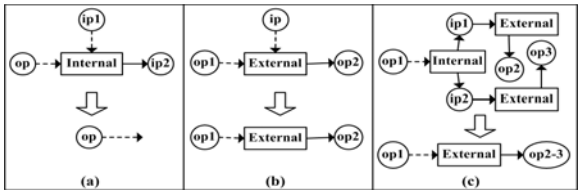

Figure 3. Elimination rules for internal elements.

In figure 3, dotted arrow stands for "as the pre-set" or "as both the pre-set and the post-set"; *ip* stands for internal place; *op* stands for observable place. Specifically, figure 5(a) presents the most common case, where many internal behaviors are modeled in the form of its composition. Its elimination is to make *ip2* glued into *op*, and *ip1* with internal transition deleted, and the pre-set of *ip1* and *ip2* are attached to *op* instead. In figure 5(b), the token data in *ip* just provide internal data, so it is omitted directly. A special case is presented in figure 5(c) where an internal transition has multiple internal places as its post-set. The branches are integrated into one flow to merge internal behaviors and the subsequent external behaviors. As it tends to introduce unnecessary nondeterministic behaviors, it is better to be avoided. The preceding eliminating rules could be applied iteratively until all the internal structures are appropriately handled. Furthermore, these eliminating operations are performed after the transition firing, so it does no harm to the subsequent model simulation, and token data in observable places are exactly the valid output data.

Through performing the CT-CPN$_S$ model simulation based test case generation algorithm, a test case model is finally generated, which is formally specified as the CT-CPN$_{TS}$ model in Definition 4.

Definition 4. A CT-CPN$_{TS}$ is a triple (*CPN*, P_{TS}, T_{TS}):
1) *CPN* is a basic colored Petri nets model;
2) $P_{TS} = P$, $P_{TS} = P_{TS}^I \cup P_{TS}^O \cup P_{TS}^{TO} \cup P_{TS}^V$: 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^{V}_{TS} is the set of **test verdict places**; each place sets have no intersections;

3) $T_{TS} = T$, $T_{TS} = T^{I}_{TS} \cup T^{O}_{TS} \cup T^{\delta}_{TS} \cup T^{V}_{TS}$: T^{I}_{TS} is the set of **input transitions**; T^{O}_{TS} is the set of **output transitions**; T^{δ}_{TS} is the set of **suspension transitions**; T^{V}_{TS} is the set of **test verdict transitions**; each transition sets have no intersections;

4) CT-CPN$_{TS}$ has finite and deterministic behavior, and has at most one input transition enabled in every step.

CT-CPN$_{TS}$ models could facilitate the actual test execution for its better feasibility and readability, because they not only prescribes the test sequences from the data-dependant control flow of the system behaviors, but also provides necessary and definite test oracles for determining the conformance relation.

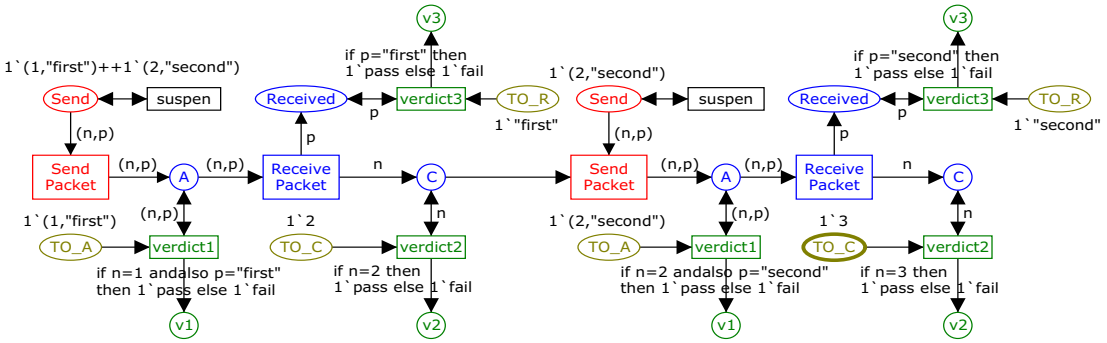

Figure 4. The CT-CPN$_{TS}$ model for the SFDP system.

Take the SFDP system as an example, we generate a CT-CPN$_{TS}$ model, presented in figure 4, through simulating the CT-CPN$_{S}$ model in figure 1 under the initial marking: M_0(*Send*) = {1`(1,"first")+1`(2,"second")} and M_0(*NextSend*) = M_0(*NextRec*) = {1`1}. It intends to test whether two data packets are sent sequentially.

Some comments are noted: (1) Input place *Send* has a suspension transition *syspen*, which allows for waiting to send the data packet. However, place *C* has no suspension transition, as it is an output place, and tokens in *C* cannot be controlled externally; (2) 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 for validating whether the actual output data is the same with the token data in *TO_A*, i.e. *n=1 and p="first"*; (3) *v1~v3* are fusion set of verdict places; (4) Internal structures are eliminated by applying the first and the second elimination rules iteratively during the test generation.

V. SUMMARY

To make the best of advantages of the ioco testing theory and the CP-nets modeling, we integrate them directly to develop a non-trivial CP-nets model based conformance test case generation approach.

Compared with other CP-nets model based test generation approaches [6,7], our approach has better formal testing theoretical basis. Besides, the model simulation based test case generation approach is irrespective with the model size, so it has better scalability.

Compared with the LTS model based test generation approach in original ioco testing theory, our approach is advantaged in two aspects: better modeling and analysis capabilities quite facilitate validating the accuracy of the system functional models; generating test cases through model simulation is bound to produce completely feasible test cases for the practical test executions.

ACKNOWLEDGMENT

This work was supported partly by National Natural Science Foundation of China under Grant No. 610701 88 and National High-Tech Research and Development Plan of China under Grant No. 2009AA01A344.

REFERENCES

[1] M. Broy, B. Jonsson, J.P. Katoen, etc. *Model-Based Testing of Reactive Systems*, LNCS 3472, Springer, 2005.

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

[3] J. Tretmans. "Model Based Testing with Labelled Transition Systems", *Formal Methods and Testing*, LNCS 4949, pp. 1-38, 2008.

[4] J. Tretmans. "Test Generation with Inputs, Outputs and Repetitive Quiescence", *Software - Concepts and Tools*, Vol. 17(3), pp. 103-120, 1996.

[5] K. Jensen and L.M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

[6] H. Watanabe and T. Kudoh. "Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets". *In Proc. of the 2nd Asia-Pacific Software Engineering Conference (APSEC 1995)*, Brisbane, Australia, pp. 242-251, 1995.

[7] U. Farooq, C.P. Lam and H. Li. "Towards Automated Test Sequence Generation". *In Proc. of the 19th Australian Conference on Software Engineering (ASWEC 2008)*, Perth, Australia, pp. 441-450, 2008.