Verification of Safety-Critical Software Requirement Based on Petri-Net Model Checking

Ning Ma, Xiaohong Bao, Zhen Li and Haifeng Li
Department of Engineering System and Engineering, BeiHang University, China
maning@dse.buaa.edu.cn

Abstract

Formal modeling and software requirement verification could improve dependability of safety-critical software. Model checking technology is widely recognized by industry. It has the advantage of a flexible input language, but its capabilities of debugging, graphical modeling and interpretation are limited. Petri net is a strictly defined formal modeling approach. In this paper, the semantics of Petri net can be extended and mapped to adapt to the modeling and verification of software requirements.

1. Introduction

Correct Requirement is essential for the development of safety-critical software [1]. Ref. [2][3]pointed out that the formal methods are beneficial to the development.

At present, the language editor of model checking tools is too weak for debugging and inspection, which makes troubles in editing requirements and specifications of inspections. Based on the current study on model checking 0, the semantics of Petri net can be extended in 5 ways illustrated in section 2.

2. Expansion of Petri Net

To meet requirements of complicated and dynamic behaviors of software, the definition of Petri net is extended as follows.

2.1 Zero-Weight Function

In the Petri net structure, transition of 1 as the default weight function has the meaning of delivering resources.

For software system, there are a large number of judgments and loop jump in the structure of programming language, which needs to be verified and eventually realized. The default weight function for a reduction in consumption does not reflect the judgments and recycling program semantics, meanwhile it causes the difference between the numerical value of judge and the value of cycle. Therefore the default value of weight function is defined as zero.

2.2‘Non’ Dashed Line

In traditional Petri net, migration can be triggered only if the relocation of token is available. However in practice, a lot of migrations triggered are based on the non-true state. But there is no semantics definition or graphical representation for failure migration. Therefore, we use $\dashrightarrow$ to represent ‘non’ triggered location and ‘non’ migration.

2.3 Threshold Value Migration

Threshold value is critical to system modeling. It is easier to construct migration with threshold as threshold value, accumulator and counter, etc, due to changed default value and added ‘non’ dashed.

2.4 Enumerated and Numeric Place

In Petri net, a place corresponds to a minimal state variable of system. Enumeration type takes the name of state as location mark, and black spots means it is activated. Numeric value identifies the place is located in parentheses after the initial marked, in the library of the right of the sequence indicated by the value scope.

2.5 Value-Assigned Transition

Traditional Petri token had characterized the state of the system changes in software systems often use of state variables to represent the numerical assignment. In many cases, places are required for direct re-assignment. Arrow with the icon expresses a mandatory assignment where N is a specific assignment number.

2.6 Extended Definition

EPN=(P,T,F,W,M)=(EN,W,M)

In which, EN is extended as follows:

Place P={PID,Pkind,Pvalue}, contains ids, types and values, in which, ,Pkind ∈{0,1}，0 is in Boolean state, and 1 is in numeric.

Pvalue ={My,Vmin,Vmax}， in which, My is current value, equals to finite number K，My is either T or F. Vmax and Vmin are the upper and lower limit of the Place, not considered when it is Boolean.

Transition: T={TID,Tkind}, in which, Tkind ∈ {TN,TS}, TN is normal transition, TS is Value-Assigned Transition

$F \subseteq (P,T)\times(T,P)$,is a set of arcs

$W: F \to Z$, Weight in token transition TN or value assigned in TS.

Transition conditions

t ∈ T, 't'='t ∪ t',is called extension of t. The conditions when t happens is as follows:

$$\forall p \in {}^{\cdot}t : M(p) \geq W(p,t) \wedge \forall p \in t^{\cdot}, tt_{kind} = T_N : \quad (1)$$
$$p.p_{value}v_{\min} \leq M(p) + W(t,p) \leq p.p_{value}v_{\max}$$
$$\forall p \in {}^{\cdot}t : M(p) \geq W(p,t) \wedge \forall p \in t^{\cdot}, tt_{kind} = T_S : \quad (2)$$
$$p.p_{value}v_{\min} \leq W(t,p) \leq p.p_{value}v_{\max}$$

- Transition results:

$$M'(p) = \begin{cases} M(p) + W(t,p), \\ (p \in {}^{\cdot}t \cap t^{\cdot}) \wedge (t \in T_N); \\ W(t,p), \\ (p \in {}^{\cdot}t \cap t^{\cdot}) \wedge (t \in T_S)。 \end{cases}$$

3. Petri Net Mapped to Nusmv

3.1 Nusmv Program Structure

NuSMV is a symbolic model checking tool. It can be applied to fields of industrial design verification, custom verification tools and formal verification of the test platform.

◇ Declare state variables, including Boolean, enumeration and integer.

◇To Initialize and change state variables by:

ASSIGN

init(b0) := 0; next(b0) := !b0;

◇ Operator expression: Arithmetic operators, Comparison operators, logical operators and set operators

◇Conditional expression:

case

e1 : e2;

e3 : e4;

...

Esac

3.2 Examples

Take Timer_4s counter as an example. The initial state is 0. The accumulation and reset of the counter are monitored by Timer_4s_state. And they change when threshold value changes. As shown in Fig 1, the semantics can be clearly mapped to the codes of NuSMV:

init(Timer_4s) := 0; next(Timer_4s) := case

Timer_4s_state= Start & Timer_4s < 40 : Timer_4s + 1;

next(Timer_4s_state) = Clear : 0;

next(Timer_4s_state) = Restart & Timer_4s < 40 : Timer_4s + 1;

next(Timer_4s_state) = End : 0;

1 : Timer_4s; esac;

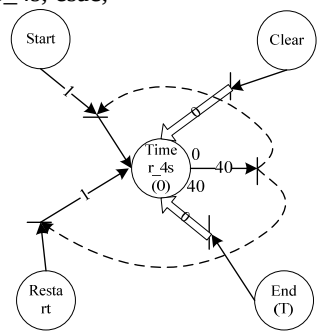

Figure 1 The comparison of fitness with Ohba

4. Requirement Verification

The path of counter-examples by NuSMV can be used to improve the design to requirement and consists of an integrated state-transform process contributed to the test case generation.

References

[1] Gregory Zoughbi. A UML profile for developing airworthiness-complaint (RTCA DO-178B) safety-critical software[D]. Carleton University, Ottawa, Canada. 2006.

[2] 'Trusted Software Basic Research' 2007 Annual Program Guide: http://www.nsfc.gov.cn/nsfc/fj.

[3] Liu Ke, Shan Guangzhi, He Jifeng, Zhang Zhaotian, Qin Yuwen. 'Trusted Software Basic Research' Summary of Major Research Program [J]. Science Foundation in China，2008，III：145-151.

[4] Cicirelli, F., Furfaro, A, Nigro, L. An approach to protocol modeling and validation. 39th Annual Simulation Symposium, 2-6 April 2006 Page(s):8 pp.