International Journal of Security and Its Applications
Vol.9, No.12 (2015), pp.323-330
http://dx.doi.org/10.14257/ijsia.2015.9.12.30

An Approach Converting XMI to SMV

Rongshang Chen[1], Jinyu Kai[2], MingXu[1] and LeiXiao[3]

[1]*College of Computer and Information Engineering, Xiamen university of Technology, Xiamen, China*
[2]*School of Computer Engineering and Science, AnYang Normal University, AnYang, China*
[3s]*Xiamen Institute of Software Technology*
rschen@xmut.edu.cn

Abstract

The technique of model checking is playing a more and more important role in formal verification and automated software testing. When using the model checker tool NuSMV, people have to program the code for the model they built firstly. During the course of programming with the input language of NuSMV, for non-expert users, some manual mistakes may be brought in such as making syntax errors or omitting some transfer conditions etc. This paper introduces a tool XMI2SMV which is an automatic generator to be used to generate NuSMV programming codes for XMI files. This tool aims to bridge between a UML tool and the model checker tool NuSMV. We just need build our behavioral system model using a UML tool and export its corresponding XMI file, and this tool can help us to generate NuSMV code automatically, avoiding manual errors.

Keywords: *Model Checking, NuSMV, FSM, UML State Diagram, XMI, SMV codes*

1. Introduction

The use of model checkers for formal verification and automated software testing has received more and more attention in the literatures 0[2]. When we use the model checker tool NuSMV, we have to program the code for the model we built firstly. During the course of programming with the input language of NuSMV, some manual mistakes may be brought in such as making syntax errors or omitting some transfer conditions *etc*.

In this paper, we introduce a tool XMI2SMV which is an automatic generator to be used to generate NuSMV programming codes for XMI files and it aims to bridge between a UML tool and the model checker tool NuSMV. We just need build our behavioral system model using a UML tool and export its corresponding XMI file, and this tool can help us to generate NuSMV code automatically, avoiding manual errors.

The rest of the paper is organized as follows. Section 2 presents the related conceptions and Section 3 discusses our tool in details. In this section, we introduce the mapping rule on how to convert the XMI file into SMV programming code and the architecture about this tool. Section 4 illustrates with a simple case study on how this tool works to generate smv programming code, we also run the generated smv code on the model checker tool NuSMV. Finally, we conclude our paper in Section 5 and give some suggestions for future work of this tool.

ISSN: 1738-9976 IJSIA

2. Related Conceptions

2.1. Model Checking

The use of model checking has received more and more attention in the literature. Originally, model checking was intended for formal verification: Given a system behavioral model and a temporal logic property, a model checker will exhaustively analyze the model's state space in order to prove property violation or satisfaction [3]. Sometimes model checking was intended for verifying the security of set protocol in some papers [15].

Verification method based on model checking can find errors early in the embedded software development, so that a large number of repetitive labors can be avoided, reducing the factors leading to serious consequences. Whether to prove the property violation or satisfaction or to generate test suits automatically with model checkers, the first and critical thing is to formalize the system behavioral model.

2.2. NuSMV [4]

NuSMV is a reimplementation and a reengineering of the SMV model checker developed by McMillan at Carnegie Mellon University during his PhD, and it is a software tool for the formal verification of finite state systems. It has been developed jointly by FBK-IRST and by Carnegie Mellon University.

NuSMV allows checking finite state systems against specifications in the temporal logic CTL. The input language of NuSMV is designed to allow the description of finite state systems that range from completely synchronous to completely asynchronous. The NuSMV language provides for modular hierarchical descriptions and for the definition of reusable components. The basic purpose of the NuSMV language is to describe the transition relation of a finite Kripke structure using expressions in proposition calculus.This provides a great deal of flexibility, but at the same time it can introduce danger of inconsistency for non experts users.

2.3. Finite State Machine

Finite state machine [5][6] is a kind of automaton frequently used in the research of automation theory and calculation theory, finite state machine is widely used in modeling the system behavior of an application**Error! Reference source not found.**-[8].

A Finite State Machine is a Quintuple $FSM = (I, O, S, \delta, \omega)$*:*

- *I is a set of inputs.*
- *is a set of outputs.*
- *S is a not null set of states.*
- $s_0 \in S$ *is an initial state.*
- $\delta : \mathrm{S} \times \mathrm{I} \to \mathrm{S}$ *is a state transition function, that is, for every* $s \in S$, $i \in I$ *there is a* $s' \in S$ *such that* $\delta(s,i) \to \mathrm{s}'$.
- $\omega : \mathrm{S} \times \mathrm{I} \to \mathrm{O}$ *is a output function, that is, for every* $s \in S$, $i \in I$ *there is a* $o \in O$ *such that* $\omega(s,i) \to o$.

For every finite state machine, we can use a statetransition directed graph to represent it, the Vertices in the graph correspond all of the states in the state set S, and edges correspond all of the transitions, tagging the related input and output.

A directed graph representing a finite state machine is showing as Figure 1.

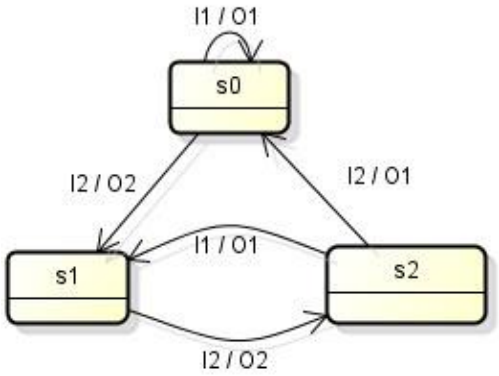


Figure 1. The Directed Graph of FSM

In terms of this FSM:
The inputs set I={I1,I2};
The outputs set O={O1,O2};
The states set S={s0, s1, s2} and the initial state is s0;
The states transition function is as the following:
$\delta(\mathrm{s0}, I1) \to \mathrm{s0}, \delta(\mathrm{s0}, I2) \to \mathrm{s1},$
$\delta(\mathrm{s1}, I2) \to \mathrm{s2}, \delta(\mathrm{s2}, I1) \to \mathrm{s1},$
$\delta(\mathrm{s2}, I2) \to \mathrm{s0};$
The output function is as the following:
$\omega(\mathrm{s0}, I1) \to \mathrm{O1}, \omega(\mathrm{s0}, I2) \to \mathrm{O2},$
$\omega(\mathrm{s1}, I2) \to O2, \omega(\mathrm{s2}, I1) \to \mathrm{O1},$
$\omega(\mathrm{s2}, I2) \to O1;$

2.4. UML State Diagram and XMI

Unified Modeling Language (UML) is such a graphical modeling language which includes graphical mechanisms that enable humans to quickly understand the structure and behavior of the modeled system and is standardized by the Object Management Group (OMG)[9] [9]. The current version of the standard is 2.5[10]-[11].

State diagrams show how the internal state of an object evolves during the time. It is directed graph which represents a state machine, whose nodes represent all the possible states of the objects, and the edges represent the transitions between states. A state diagram, thus, shows how objects evolve from their initial state toward a final state. Each state may contain another state diagram, which allows specifying its behavior at a deeper level of detail.

Transitions may be triggered both by external or internal events. An example of an internal event is the completion of an internal activity. An example of external event is the reception of an interrupt signal from other parts of the system. A state may have an optional name, and an internal transitions compartment containing a list of internal transitions or activities which are performed while the object is in that state. Actions have an optional label identifying the condition under which the action is performed.

3. From XMI to SMV Process

This process intends to transform the XMI files into the SMV codes to present the formal specification. To reach this aim, we rely on the XMI and SMV language and the transfer rules between them.

3.1. The NuSMV Input Language

The program coded by the NuSMV input language is called .smv program. We call it .smv program or .smv code. It provides constructs for hierarchical descriptions of FSMs and each FSM is described as a module. Similar to C program language, a smv program can consist of many modules, the syntax of a NuSMV program is like below:

Program ::

module_1

module_2

...

module_n

There must be one module with the name main and no formal parameters. The module main is the one evaluated by the interpreter.

As for one FSM semantic described by a Kripke structure, the state space, initial states, and the transition relation between states, all should be presented in the smv program.

The .smv program is divided into sections: variable declarations, assignments, specification and *etc*.

Section of variable declarations describes the state space of Kripke structure, each variable has its own data type that can be boolean, enumeration, array, *etc*; when we declare the variable, the keyword VAR or IVAR are used.

The transition relation of the FSM is expressed by defining the value of variables in the next state (*i.e*. after each transition), given the value of variables in the current states (*i.e*. before the transition). The case segment sets the next value of the variable state to the value busy (after the column) if its current value is ready and request is 1(*i.e*. true). Otherwise (the 1 before the column) the next value for state can be any in the set {ready,busy}. The variable request is not assigned. This means that there are no constraints on its values, and thus it can assume any value. The variable request is thus an unconstrained input to the system.

3.2. Mapping Rule

We present a simplified version of the actual rules. Let XMI be the input systematic model and .smv program code the to-be-generated output we hoped. The rules are difined as follows:

First, we introduce the mapping rule of main module.

Step 1: map all of the input affairs, in another words, trigger events into corresponding state variables, each state variable represents a trigger, and set the data type of the state variable Boolean.

Step 2: map all the states in UML state machine diagram into one whole state variable, and set it date type self-defined data type which is the sub-module name with input parameters, and the input parameters are the state variables mapped from the trigger events. The reason we do this is that whether the states can change from previous state to later one are decided by the happening of trigger events, if the trigger events happen, the changes of the states must happen.

Up to now, number of the state variables in the main module will be the sum of the number of trigger events plus one.

Step 3: Set the initial value of the state variable representing the first trigger with the init expression, and set its value true. The values of other state variables representing other triggers are expressed with the next expressions.

Second, define the state variables and transition relations in the sub-module. The sub-module is a module with input parameters and the input parameters are the variable mapping by trigger events.

Step 1: declare a state variable to represent the whole state set of the UML state machine diagram and the data type of it is enumeration types whose members are the elements of the state set.

Step 2: set the initial value of the state variable, the value is decided by the initial state in the UML state machine diagram which is pointed by the initial state point.

Step 3: map the transition relations with the next expression, and the value of next state is decided by the transfer line arrow, through exploring the XMI description file, we can get the present state value, the next state value and the trigger events inducing the state change.

Up to now, we finish the map from XMI file to .smv program code.

3.3. XMI2SMV Architecture

The XMI2SMV tool is illustrated by the Figure 1. The Analyze the XMI file box represents to read a XMI file and extract information and store it into the self-defined data structure of direct graph; The Automatic SMV code generation box aims to generate the corresponding input language of NuSMV according to the information and the mapping rule.

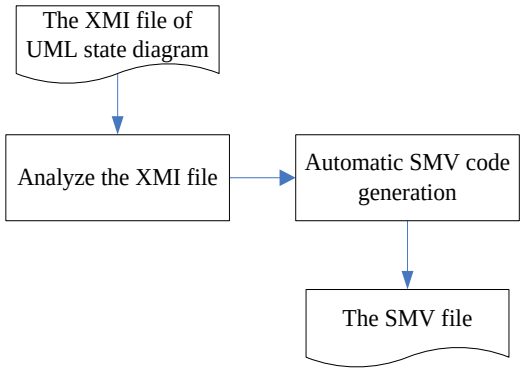


Figure 2. Basic Architecture of the XMI2SMV Tool

3.4. Realise XMI2SMV in Python

Python aims towards simplicity and generality in the design of its syntax, encapsulated in the mantra "There should be one — and preferably only one — obvious way to do it", from "The Zen of Python"[12]-[13].

Using the programming language Python [12], the tool of XMI2SMV is realized and the main source code lists as following:

#to using the first approach to parse the xmi file. To need user input the direction of the xmi file representing the state diagram.

4. A Case Study

4.1. Modeling based on FSM

We take the literature z[14]as an example, the system state set includes of Touch_Idle, Touch_Up, Touch_Down, Touch_ChkDown, Touch_ChkUp, when the trigger events as system inputs were triggered, the system output happen, at the same time, the state of system changes. In this case study, we emphasis the transitions of the system states and omit the ouput of system. The behavioral system model is shown as Figure. 3.

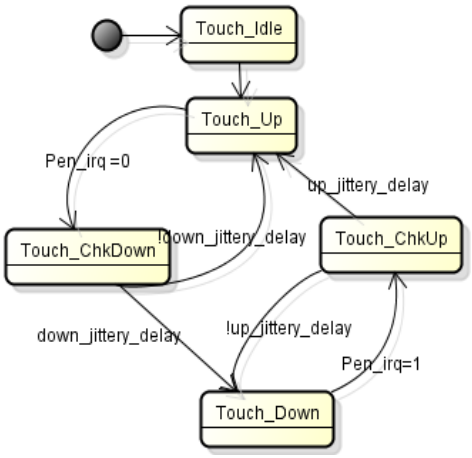


Figure 3. The system Behavior of Touch Screen

4.2. Generating Code

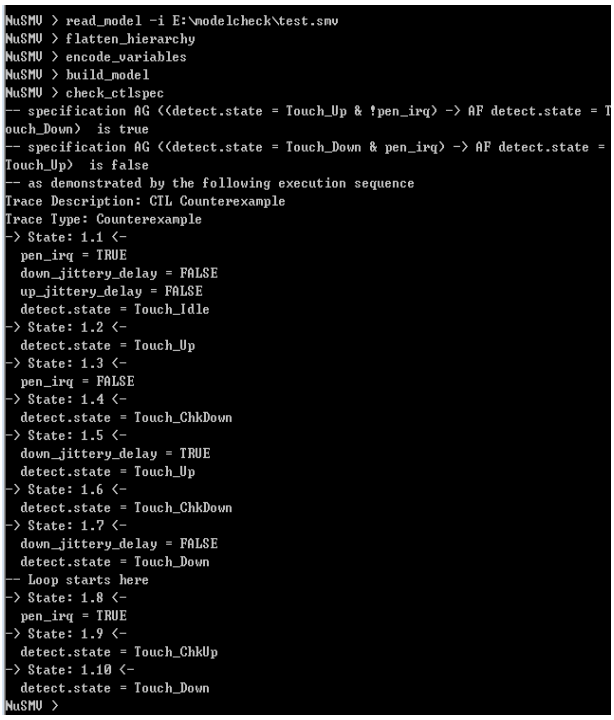
Using of the UML tool we export the XMI file of the Figure 2.Read the XMI file got from the step 2, the XMI2SMV tool can generate the .smv code automatically rapidly. The code generated from the Figure 2 is as following:

```
MODULE main
VAR
pen_irq :boolean;
down_jittery_delay:boolean;
up_jittery_delay:boolean;
detect: Touch_Detect(pen_irq, down_jittery_delay, up_jittery_delay);
SPEC
  AG(detect.state = Touch_Up & !pen_irq  -> AF detect.state = Touch_Down) (1)[14]
SPEC
  AG(detect.state = Touch_Down & pen_irq -> AF detect.state =Touch_Up) (2)[14]
MODULE Touch_Detect(pen_irq, down_jittery_delay, up_jittery_delay)
VAR
state :{Touch_Idle, Touch_Up, Touch_Down, Touch_ChkUp, Touch_ChkDown};
ASSIGN
        init(state) := Touch_Idle;
        next(state):= case
state =Touch_Down & pen_irq :Touch_ChkUp;
state =Touch_Idle |  state = Touch_ChkDown & !down_jittery_delay |
state = Touch_ChkUp & up_jittery_delay : Touch_Up;
state = Touch_Up  & !pen_irq  : Touch_ChkDown;
state = Touch_ChkDown & down_jittery_delay |
state =Touch_ChkUp & !up_jittery_delay  : Touch_Down;
1=1:state;
esac;
init(pen_irq) := TRUE;
next(pen_irq) := case
state =Touch_Up & pen_irq : FALSE;
state = Touch_Down  & !pen_irq :TRUE;
1=1: pen_irq;
esac;
next(down_jittery_delay):= case
```


state =Touch_ChkDown &
down_jittery_delay :FALSE;
state = Touch_ChkDown & !down_jittery_delay :TRUE;
1=1: down_jittery_delay;
esac;
next(up_jittery_delay):=case
state = Touch_ChkUp :FALSE;
state = Touch_ChkUp :TRUE;
1=1: up_jittery_delay;
esac;

4.3. Run Code

At last, we run the code generated by the XMI2SMV on the model checker tool of NuSMV. The code can run smoothly, and we get the result as following:



5. Conclusion

This paper introduce an approach converting XMI to SMV for model checking, and present a tool for automatically generating the SMV program based on an input XMI file. Using the architecture described in the previous section, we have completed a prototype version of this tool in Python language. When using it, we just need to specify the path and the name of the XMI file and that of the output file, and this tool can generate the smv program code automatically. This tool can avoid manual faults when describing a system behavioral model with the input language of the model checker tool NuSMV and makes it more flexible to use of NuSMV.

We will continue to extend the capabilities of the tool and improve its maturity, and derive the simulation model by automatically extracting information from the counterexamples producted by a model checker tool, and use this information to build a discrete-event simulation model, which will finally execute dynamically, and insert the simulation results back into the original UML state machine diagrams as tagged values so that providing user feedback at an intuitively way. We will take those as our future work.

Acknowledgements

This work is support in by a project of Xiamen science and technology program under grant No. 3502Z20133043 and 3502Z20133041.

References

[1] H. Shi, W. Ma, M. Yang and X. Zhang, "A case study of model checking retail banking system with spin," Journal of Computers, vol. 7, no.10, pp. 2503-2510, **2012**.
[2] A. Sagahyroon , G.Lakkaraj , M. Karunaratne. "Verification Components Reuse[J]. Journal of Computers", vol. 7, no. 11, **(2012)**, pp. 2641-2649.
[3] G. Fraser,F. Wotawa, P. Ammann, "Issues in Using Model Checkers for Test Case Generation[J]".Journal of Systems and Software, vol. 82, no. 9, **(2009)**, pp. 1403-1418.
[4] http://nusmv.fbk.eu/NuSMV/index.html
[5] Meng Yang,State Assignment for Finite State Machine Synthesis, Journal of Computers, vol.8, no.6, **(2013)** June **,** pp.1406-1410,
[6] P. Xue Xian, J. Tao, L.Lin, "checking the simulation model based on finite state machine [J] Computer Engineering and Science", vol. 34, no. (5), **(2012)**, pp.153-156. [In Chinese]
[7] R. De Nicola,F. Vaandrager,"Action versus state based logics for transition system [M]" **(1990)** pp. 407-419.
[8] Z. Manna,A. Pnueli, "Models for Reactivity[J].Acta Informatica", vol. 30, no. 7, **(1993),** pp. 609-678.
[9] I. Dietrich,C. Sommer ,F. Dressler, *et al*., "Automated Simulation of Communication Protocols Modeled in Uml 2 with Syntony[Z]".**(2007)**, pp. 104-115.
[10] OMG Unified Modeling LanguageTM(OMG UML)Version 2.5 FTF—Beta 1 http://www.omg.org/ spec/UML/2.5/Beta1/PDF/
[11] M. Marzolla, "Simulation-based Performance Modeling of Uml Software Architectures[Z]", **(2004)**.
[12] G. van Rossum, "The Python Programming Language",Http://python.org, Last viewed on January 30th, **(2008)**.
[13] PEP 20 – The Zen of Python http://www.python.org/dev/peps/pep-0020/
[14] Chenbo *et al*; "A method for verification of embedded software based on model checking [J]"; Microcontroller and embedded system application (5): pp. 66-68.[In chinese]
[15] T. Ling-mei, D. Rong-sheng,Symbolic model checking the set protocol[J],Journal of Guilin University of Electronic Technology, vol. (4), **(2005)**, pp. 1-5.

Author

Rongshang Chen. Male, engineer, research direction: intelligent information processing, network security.

