Combining Theorem Proving and Model Checking in the Safety-Critical Software Development through Translating Event-B to SMV

Liang Sen ^{1,a}, Luo Xiangyu^{1,2} and Chen Zuxi ¹

¹College of Computer Science and Technology, Huaqiao University, Xiamen, China

²Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China

Abstract. Model checking and theorem proving are two key verification techniques in the formal method, but each has its advantages and disadvantages. In this paper, we first try to present the general model transformation rules from Event-B to SMV in order to realize complementary advantages, and then design the model converter of Event-B to SMV according to the rules. ProB is the only tool to implement Event-B model checking, but it lacks the real-time property verification, while nuXmv is able to verify temporal logic formulas with the real-time property for SMV models. After completing the model transformation, we use ProB and nuXmv to make the verification of LTL and CTL formulas for two equivalent models respectively. Compared to the experimental results, in solving the same problem, nuXmv has more advantages over memory consumption, time efficiency and so on than ProB, especially when the formula to be verified is very complex. Therefore, if complex verifications of system models need to be performed, it is necessary to implement the model transformation between Event-B and SMV by using our translation rules, which is of great significance for the development, design and verification of safety-critical system softwares.

1 Introduction

Formal verification method can be divided into two broad categories: model checking and theorem proving. The advantage of model checking is that it can be automated and provide counterexample paths when a specification is discovered to be false; the disadvantage is that if the system has a high degree of concurrency, there exists a state explosion problem, moreover it generally can not verify the system with infinite states. Theorem proving uses mathematical reasoning and logic inference to prove the system correctness. Its advantage is that the model to be proved is independent of system states, so it has the ability to verify the system with infinite states; the disadvantage is that this approach is generally not fully automated and requires manual interaction.

SMV (Symbolic Model Verifier) is the first model checker based on BDDs (Binary Decision Diagrams), which uses the BDD to alleviate the state explosion problem, thereby implement the symbol model checking technology [1]. For the verification of concurrent systems with finite states, it first needs to use the SMV specification language to model the system, then, the transition system described by BDDs is extracted from the model, part properties of the system is expressed in CTL (Computation Tree Logic). Finally, the search algorithm based BDDs is used to identify whether the

system satisfies verified properties. If a finite state system has a given property, the True is output, else the False is output, SMV also shows counterexamples of the specification the system does not meet. SMV was originally used only to check whether a finite state system satisfies the CTL formula.

The currently widely used NuSMV [2] is the reconstruction and extension of SMV, NuSMV supports all specifications described by the CTL and LTL (Linear Temporal Logic). In addition, it integrates bounded model checking technique based on SAT. The latest version of nuXmv [3] extends from NuSMV. nuXmv is a new symbolic model checking tool that can analyze synchronous finite and infinite state systems. For the case of finite state, nuXmv is an efficient verification engine based on SAT algorithm. For the case of infinite state, it is based on SMT verification technology, tightly integrated with MathSAT5. We choose nuXmv to complete the experimental part of this paper.

B method [4] is a verification method based on theorem proving proposed on the basis of VDM and Z language. The B# method was first proposed, based on which it was created as Event-B [5]. Compared to the traditional B method, introducing event is the new feature of Event-B. The event indicates that actions are completed by triggering a state. Therefore, it is well suited for modeling periodic behavior. Event-B also

^a Corresponding author: liangsen@hqu.edu.cn

covers the entire development cycle of software from modeling, refinement and automatic verification to code generation, so it is applicable to complex software development.

Event-B reduces the modeling difficulty by the idea of preliminary abstract modeling and layer by layer refinement. Moreover, Event-B uses the proof obligations generated by axioms and theorems to ensure the consistency among different layers. It indicates the model meets software requirements through the fully proved obligations, but it can not verify specific model properties. While SMV is suitable for system property verification, and counterexamples can be generated to adjust the model, it is difficult to make a perfect SMV model for a complex system at once, whether the model is good or bad can not be proved.

Compared with the SMV language, the Even-B modeling language is more concise and easy to understand, but it is limited to model discrete systems, and can not describe system real-time properties. Both the current version of NuSMV and nuXmv support the description and verification of real-time systems. Combining model checking and theorem proving is a recent important trend of formal verification[6-8], however, the existing work focuses on the state explosion problem and seldom considers the complementarity of the two technologies in software engineering practice. Therefore, a practical scheme is proposed in this paper, which combines model checking and theorem proving for software development.

2 Model Structure Analysis

The formal Event-B model is based on first-order logic and typed set-theory [9], which is mainly used to describe the relationship among system components. Relatively speaking, SMV model has its own SMV input language. They are very different in terms of system modeling, but the two essentially describe the state transition of discrete systems. ProB is the only tool currently available for Event-B model checking, it is originally designed to verify program specifications written in the B language. ProB uses SICStus Prolog to execute model checking in a constrained logic programming (CLP) environment [10-11]. Because Event-B and B are similar in many ways, Event-B model checking is also supported now. In addition, it can be combined with B-Motion Studio [12] tools to realize the visual simulation and other functions in the software design process.

2.1 Event-B model structure

The core concept of Event-B formalization method is the model, a typical Event-B model contains a context that describes static system properties and an abstract machine that describes dynamic system behaviors, context and machine organize model elements into the basic model architecture [5]. Abstract machine and context have the following relationship: an abstract machine can be refined by another machine, and a context can be extended by another context, also, a machine can see a number of

contexts. The relationship between machine and context is described in Figure1.

Figure 1. the relationship between machine and context

The main model elements in the context include constant, set, axiom, and theorem. constants set and constants describe the static and intrinsic attributes for the system. Axioms specify the types, attributes, and relations of constants in the form of predicate formulas, which are used as fundamental assumptions in proof obligations. Theorems contain important inferences from axioms, which is the important basis for proof obligations. It ensures the correctness of axioms and eliminates the ambiguity.

Abstract machine contains the following main elements: variable, event, invariant and theorem. Variables are used to describe system states, all dynamic system properties can be characterized by variables. Events are used to describe changes for dynamic system properties and state transitions. The function of theorems of the abstract machine is analogous to theorems of the context. Before theorems are proved, axioms and theorems in the seen context, invariants of a refined machine and the current machine have been proved.

Event has three definitions, as shown in Table 1. In definition 3, the any statement defines the parameter t in event, the where statement is used to introduce the guard of event, the guard condition is the triggering condition for event, which exists in the form of predicate formula. The event is allowed to be triggered only when the guard condition is true, that is, when the constraint $G(v)$ is satisfied, the action $S(v)$ is executed under the then statement, where v is the system variable. $S(v)$ may contain one or more different actions, actions describe the evolution of state variables when event is triggered. Guard conditions contain attribute restrictions for parameters, if there are no parameters in the event, the where statement is reduced to the when statement in definition 2. In definition 1, an event without parameter is used to describe the initialization event.

Table 1. Definition of event

Three possible definitions for an event
Defination1: $E = \text{begin } S(v) \text{ end}$
Defination 2: $E = \text{when } G(v) \text{ then } S(v) \text{ end}$
Defination 3: $E = \text{any } t \text{ where } G(t,v) \text{ then } S(t,v) \text{ end}$

2.2 SMV model structure

SMV model is used to describe Kripke structure and specification to be verified. The Kripke structure is often called the Finite State Machine (FSM) in the SMV. The corresponding SMV program can be used as the input language for nuXmv, the SMV program is composed of modules. A module consists of name and definition, and the module definition is composed of parameters and the main body. The main body of a module is divided into three categories: Variable section, Constraint section and Specification section. The Variable section describes the state set, the Constraint section describes the transition relationships and some restrictions of the model, the Specification section describes the properties to be verified. SMV program must have at least one main module, and the main module has no formal parameter. Multiple modules can be used to describe the FSM, and then combined into a whole FSM, a typical SMV program structure is as follows:

```

MODULE main
VAR...
ASSIGN...
SPEC (or LTLSPEC,CTLSPec)
...
MODULE...
    
```

There are two main categories in the Variable section:

Figure 2. Development framework of safety-critical softwares

syntax, the file format includes bcc, bcm, bpo, etc. The bcc file is used to record the machine, and the bcm file is used to record the context. ProB can be used as a plugin for Rodin to support Event-B, and the implementation principle is to extract file whose suffix is .eventb from the Event-B model. It can also make model checking by the .eventb file directly in the command line version of ProB. After refinement and verification by Rodin and ProB, we can get the Event-B XML file directly from record files, the basic structure of the Event-B XML file is as follows:

```

<org.eventb.core.scMachineFile...>
  <org.eventb.core.seesContext.../>
  <org.eventb.core.scInternalContext >
    
```

State Variable (state assignment indicates a specific state) and Input Variable (state represented by labeled relationships), they are represented by the keyword VAR or IVAR, respectively. The Variable type can be boolean, integer and enum, etc. The Constraint type can be assign, trans, init, invar and fairness, etc., respectively, using keywords ASSIGN, TRANS, INIT, INVAR, FAIRNESS as declarations. The Specification section uses the SPEC, LTLSPEC and CTLSPec as keyword declarations, using CTL and LTL as specific property description.

3 Translation rules

After introducing composition structures of the two models, we propose a combination of Event-B theorem proving and SMV symbolic model checking for the safety-critical software development. The software model is constantly modified and perfected by formal verification technologies, the framework is shown in Figure 2.

The Event-B model design and proof obligation to be proved can be implemented by Rodin platform [13]. Rodin is a tool platform based on the Eclipse framework, which extends the advantages of open source feature and plugin scalability. With rich plugin support system, it can complete visual modeling, graphic simulation, automatic proof, model checking and so on. Models in Rodin will generate a variety of record files in the form of XML

```

<org.eventb.core.scAxiom.../>
<org.eventb.core.scConstant.../>
<org.eventb.core.scInvariant.../>
<org.eventb.core.scVariable.../>
<org.eventb.core.scEvent>
...
</org.eventb.core.scEvent/>...
</org.eventb.core.scMachineFile/>
    
```

Firstly we present the model translation rules of Event-B to SMV, then design a converter. The converter reads various components of Event-B model from the XML file and outputs SMV model, then we can add the verified model properties to implement model checking in nuXmv.

3.1 Variable declaration

The process mutual exclusion as an example, B1 and B2 are the variable declaration in Event-B, and S1 is the variable declaration in SMV, the correspondence between them is shown in Figure 3. In the context description section of B1 and the machine variable declaration and definition section of B2, system property definitions are extracted. The relationships are described by sets and partitions in Event-B model, but in SMV, they are implemented by the enumeration type. For example, we can get $STATE: \{non_crit, wait, crit\}$ from $partition(STATE, \{non_crit\}, \{wait\}, \{crit\})$ and $proc \in STATE$. Thereby the process $proc$ is one of the elements in $\{non_crit, wait, crit\}$, that is, $proc1: \{non_crit1, wait1, crit1\}$; the range of integer y is 0 to 1, which is expressed as $inv3$ in Event-B: $y \in 0..1$, corresponding to $y: 0..1$ in SMV.

The two processes, $proc1$ and $proc2$ can not enter critical section at the same time, in Event-B, the invariant description $inv4: \neg (proc1 = crit1 \wedge proc2 = crit2)$. In SMV, the keyword **INVAR** is declared, then, the $inv4$ is equivalent to $!(proc1=crit1 \& proc2=crit2)$.

Figure 3. Correspondence between B1, B2 and S1

3.2 Variable initialization

All Event-B variables are initialized in the event INITIALISATION, as shown by B3 in Figure 4, SMV sets the initial value for each variable in the ASSIGN section: $init(variable) := value$, the equivalent SMV expression of B3 is shown in S2.

Figure 4. Correspondence between B3 and S2

3.3 State transition

For the expression of state transition, Event-B takes it as multiple Actions to be executed when the event satisfies the guard, and multiple variables may be changed in an event. For example, modeling the corresponding request event, enter event and release event for each process, the value of the mutex y also varies with process states, the details are shown as B5 and B6 in Figure 5.

The state changes of $proc1$ are executed in the $act1$ of B4, B5, and B6 respectively, meanwhile the variable y is changed in the $act2$ of B5, B6. In SMV, for each variable,

Figure 5. Event correspondence of $proc1$ and $proc2$

the next statement describes the transitions when the precondition is satisfied, the specific format is as follows:

```
next(variable):=
case
cond1 : expr1;
cond2 : expr2;
...
TRUE : exprN;
esac;
```

In Figure 5, the equivalent SMV expressions of B4, B5 and B6 of the Event-B model are presented in S3 and S5. If all conditions are not satisfied, the corresponding TRUE expression is executed in SMV. In Event-B, you can describe it directly as an event without any guard, that is, the guard of the when statement in the event definition 2 is null.

The operations of proc2 are the same as that of proc1, as shown in Figure 5, the request event, enter event and release event of proc2 correspond to B7, B8 and B9, respectively.

In addition, there is another expression of conditional choice in SMV:

```
cond_expr ? basic_expr1 : basic_expr2;
which is equivalent to:
case cond1 : expr1; TRUE : expr2; esac.
```

3.4 Basic symbol correspondence

In the basic symbolic expression, there exist intersections between Event-B and SMV. The following are semantically equivalent symbols, some are just different forms, summarized as follows in Table 2.

Table 2 Basic symbol correspondence table

Event-B	SMV
$a \cup b$	a union b
$a \not\subseteq b$	$(a \text{ in } b) \& (a != b) = \text{FALSE}$
$a \not\supseteq b$	$(a \text{ in } b) = \text{FALSE}$
$a \in b$	$b: \{a\}$
$a \Leftrightarrow b$	$a <-> b$
$a \Rightarrow b$	$a \rightarrow b$
$a \wedge b$	$a \& b$
$\forall a$	$A \ a$
$\exists a$	$E \ a$
$a \neq b$	$a != b$
$a \leq b$	$a <= b$
$a \geq b$	$a >= b$
$\subset$	$(a \text{ in } b) \& (a != b)$
$\subseteq$	$a \text{ in } b$
$a..b$	$a..b$
$a+b$	$a+b$
$a-b$	$a-b$
$a*b$	$a*b$
$a \div b$	$a \div b$
$a := b$	$a := b$
$a \notin b$	a is not in the enumeration of b
$a \in \mathbb{N}1$	$a > 0$
$a \in \mathbb{N}$	$a \geq 0$
$\mathbb{Z}$	integer
BOOL	boolean
$a \vee b$	$a b$
$\neg a$	$!a$
T	TRUE
$\perp$	FALSE

4 Experimental comparison

At present, the model checking for Event-B model is implemented by ProB. After the model translation, we can choose nuXmv to make various temporal property verifications. ProB and nuXmv have almost common temporal logic operators of property description. In the LTL, they all support: X, G, F, U, R (V), Y, H, O, S, T, the expression about 'release' has same meaning and different keyword, that is, 'R' in ProB, 'V' in SMV. In addition, SMV has 'Z' expression, and 'Z' is basically the same as 'Y' expression. For the syntax and semantic explanations for these temporal operators, please refer to the nuXmv manual [14] or the ProB manual [15]. In the CTL, ProB supports only the verification for operator AX, EX, AF, EF, AG, EG. In addition to supporting the above CTL formula, nuXmv also supports A [p U q], E [p U q] (p and q are sub-formula). Moreover, important difference between them is that nuXmv also support the verification for real-time property, such as:

EBF m..n p, requires that there exists a path starting from a state, such that property p holds in a future time instant i, with $m \leq i \leq n$;

A [p BU m..n q], requires that for all paths starting from a state, property q holds in a future time instant i, with $m \leq i \leq n$, and property p holds in all future time instants j, with $m \leq j < i$;

Similarly, ABF m..n p, EBG m..n p, ABG m..n p, A [p BU m..n q], etc.

To sum up, the nuXmv expression is more powerful in the temporal logic expression of model checking. We still use the process mutex exclusion as a example, then select randomly 6 groups of LTL and CTL formulas to compare the performance in ProB and nuXmv. In terms of formula expression, there is a slight difference between them. In each group, the former is the formula supported by ProB, and the latter is the equivalent SMV formula.

Formula1: $\text{not}\{\text{proc1} = \text{crit1} \ \& \ \text{proc2} = \text{crit2}\} \Leftrightarrow \text{!(proc1} = \text{crit1} \ \& \ \text{proc2} = \text{crit2})$.

Formula2: $\text{XXXXY}\{\text{proc1} = \text{crit1}\} \Leftrightarrow \text{X(X(X(Y(proc1} = \text{crit1))))}$.

Formula3: $\text{AG}(\{\text{proc1} = \text{wait1}\} \Rightarrow \{\text{proc1} = \text{crit1}\}) \Leftrightarrow \text{AG(proc1} = \text{wait1} \rightarrow \text{proc1} = \text{crit1})$.

Formula4: $\text{G}(\{\text{proc1} = \text{crit1}\} \ \text{R} \ \{\text{proc2} = \text{crit2}\}) \Leftrightarrow \text{G(proc1} = \text{crit1} \ \vee \ \text{proc2} = \text{crit2})$.

Formula5: $\text{F}(\{\text{proc1} = \text{wait1}\} \ \text{U} \ \{\text{proc1} = \text{crit1}\}) \Leftrightarrow \text{F(proc1} = \text{wait1} \ \text{U} \ \text{proc1} = \text{crit1})$.

Formula6: $\text{EF}\{\text{proc1} = \text{crit1} \ \& \ \text{proc2} = \text{crit2} \ \& \ y = 0\} \Leftrightarrow \text{EF(proc1} = \text{crit1} \ \& \ \text{proc2} = \text{crit2} \ \& \ y = 0)$.

The experimental results of ProB and nuXmv are shown respectively in Table 3 and table 4:

Table 3 Model checking results of ProB

Temporal formula	Memory usage(MB)	Total verification time(seconds)
Formula1	1.154	7.216
Formula2	1.796	7.449
Formula3	140.790	7.640
Formula4	1.540	7.373
Formula5	1.540	7.508
Formula6	140.775	7.520

Table 4 Model checking results of nuXmv

Temporal formula	Memory usage(MB)	Total verification time(seconds)
Formula1	8.556	0.074
Formula2	8.556	0.075
Formula3	8.556	0.064
Formula4	8.556	0.080
Formula5	8.556	0.072
Formula6	8.556	0.064

Experimental results show that, in memory consumption, nuXmv's memory occupancy is always maintained at 8.556MB, the reason is easy to find, NuXmv uses BDDs to describe state transitions, then the number of nodes of BDDs remains unchanged, thus, the memory occupancy is maintained in a stable state. ProB's memory consumption is related to formula complexity, which is consistent with the principle of model checking in the CLP environment. In terms of time consumption, nuXmv has an overwhelming advantage over ProB, the time that ProB returns verification results is rather slow. In addition, we have made lots of experimental comparisons by using various examples, then the analysis results are consistent with the above.

5 Conclusion

Model translation of Event-B to SMV combines theorem proving and symbolic model checking technique. The method not only uses the strict mathematical formula of theorem proving to guarantee systems meet the design requirements, but also takes advantage of the automated features of model checking tool and counterexample generation for error specifications. As a result, bugs or errors during the software model designment can be modified as soon as possible. The verified model is also refined and simulated by Event-B method, and then the correct software source code is generated directly, which is of great importance to the design and implementation for safety-critical softwares.

The Event-B model also includes complex functional relation, such as Total injection($\rightarrow$), partial injection ($\rightarrow$), total surjection ($\rightarrow$), etc. Direct conversion is not possible for these functional relations, thereby, for the complex system model described by Event-B, the modeler needs

to modify the transformed SMV model according to the actual situation in the problem, making it more realistic. Our future work is to modify our conversion algorithm based on more instances to maximize the conversion for the two models, then reduce the workload of software design and verification process. While ensuring the reliability, correctness and security of softwares, it also greatly improves the development efficiency.

References

1. K. L. Mcmillan, Symbolic Model Checking, **38**, 161-165(1992).
2. A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, Computer Aided Verification, **1633**, 495-499(1999).
3. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A.Mariotti, Computer Aided Verification, **8559**, 334-342(2014).
4. J. R. Abrial, *The B-book*(1996).
5. J. R. Abrial, *Modelling in Event-B: System and Software Engineering*(2010).
6. V. Pantelic, British Journal of Radiology, **42**, 278-280(2005).
7. K. Arkoudas, S. Khurshi, D. Marinov, M. Rinard, Lecture Notes in Computer Science, **3051**, 21-33(2003).
8. N. Shankar, Proceedings of 11th International Conference on Concurrency Theory, **1877**, 1-16(2000).
9. J. Poll, Conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries(2008).
10. M. Leuschel, M. Butler, International Symposium of Formal Methods Europe, 2805 ,855-874(2003).
11. M. Leuschel, M. Butler, International Journal on Software Tools for Technology Transfer, **10**, 185-203(2008).
12. L. Ladenberger, J. Bendisposto, M. Leuschel, Formal Methods for Industrial Critical Systems, 14th International Workshop (2009).
13. J. R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, International Journal on Software Tools for Technology Transfer, **12**, 447-466(2010).
14. nuXmv 1.1.1 User Manual, R. Cavada(2014).
15. ProB User Manual, M. Leuschel(2017).