

Model Checking Rebeca Code by SMV

M. Sirjani* A. Movaghar* H. Iravanchi** M. Jaghoori** A. Shali**

*Department of Computer Engineering, Sharif University of Technology

**Department of Electrical and Computer Engineering, University of Tehran

Abstract

Object-based modelling is known to be an appropriate approach for representing concurrent and distributed systems. Besides having an appropriate and efficient way for modelling these systems, one needs a formal verification approach for ensuring their correctness. We have developed a tool for translating our actor-based model, Rebeca, to SMV. It enables us to model check Rebeca codes. We also show that how we can reduce state space by using compositional verification.

Keywords: actor model, model checking, compositional verification, SMV, Rebeca.

1 Introduction

Reactive systems are increasingly used in applications where failure is unacceptable. Correct and highly dependable construction of such systems is particularly important and challenging. A very promising and increasingly attractive method for achieving this goal is using the approach of formal verification [3, 6].

Much work has been done on formal verification, with different system modelling languages and different verification approaches. Also, actor model [1, 2, 5] is used in different ways for modelling open, distributed systems. But to the best of our knowledge, little is done on verifying actor languages.

We use an actor-based model, called *Rebeca*¹ [9], for describing reactive systems. A tool is presented which facilitates modelling in Rebeca and then model checking the Rebeca codes using SMV [8].

A major obstacle to the use of automatic verification methods is the problem of state explosion. Specially in our model, because of the encapsulated constructs used, we may have a very large state space in model checking, even for simple systems. We use compositional verification to overcome the state-explosion problem [10, 11].

2 Rebeca

Rebeca model is similar to the actor model in that it has independent active objects, asynchronous message passing, and unlimited buffers for messages. We add class declarations to the syntax; classes act like templates

¹*Reactive Objects Language*

for states, behavior, and interfaces of active objects. We simplify the model by ignoring dynamic changing topology and dynamic creation of active objects.

Our objects are reactive and self-contained called *rebec* (for *reactive object*). Computation takes place by message passing and execution of the corresponding methods of messages. Each message contains a unique method which is invoked when the message is received. Each rebec has an unbounded buffer, called a queue (or inbox), for arriving messages. When a message at the head of a queue of a rebec is serviced, its method is invoked and the message is deleted from the queue. Each rebec is instantiated from a *class* and has a single thread of execution.

Also, we have the notion of a component as a subset of objects in the system. In this case external objects act like environment which nondeterministically sends messages to the component. State and exact behavior of external objects are not modelled. For more information, please see [9].

3 Rebeca to SMV Translator

The SMV system is a tool for checking properties of finite state systems specified in the temporal logic. The input language of SMV is designed to allow the description of finite state systems that range from completely synchronous to completely asynchronous and from the detailed to the abstract. Instead of model checking Rebeca codes directly, we have decided to develop a tool to automatically translate Rebeca codes to SMV.

We used SableCC [4] for generating the parser. SableCC produces shift-reduce parsers for LALR(1) grammars expressed in EBNF format. Parsers generated by SableCC produce abstract syntax tree (AST) of the input code. Rebeca to SMV translator is written in Java. It uses this AST to navigate in the Rebeca source code and build the SMV result code.

There is a module in SMV for each active class in Rebeca, and a process for each rebec. All the methods are translated to next-case statements in SMV. Plenty of problems encountered in translating Rebeca codes to a low level language like SMV. We tried to generate the most efficient code in SMV with the minimum reachable states while not violating Rebeca semantics.

Bounding queues is one of our main problems. We maintain a queue-overflow variable (corresponding to each rebec) in SMV code and check it as a property. Often, in our case studies, we had to increase the length of the queues to have the proper execution.

Compositional verification can be used to overcome the state explosion problem in model checking [7]. In [11] we have shown our compositional approach and proved that a property (specified in ACTL or LTL without next operator) satisfied by a component of a system is true for the system too. We translate components in Rebeca into SMV code in a semi-automatic way. In the next section, we mention some of our results and it can be concluded that in some circumstances our compositional approach can reduce the state space significantly.

4 Case Studies

We translated some Rebeca examples into SMV code. Then, we used SMV code to check safety, deadlock and starvation properties. In all the examples, there were bugs in our code which were found by model checking. Some of the bugs simply were in initializing variables and some were more serious ones, in communication

Model	Reachable states	Total states	CPU time (mm:ss)	Memory usage (KByte)
2 Philosophers and Forks	285	3.28312E+22	00:00	11136
3 Philosophers and Forks	14671	8.79172E+36	00:12	19304
4 Philosophers and Forks	390720	1.8084E+52	06:28	38700
2 Philosophers and 1 Fork with 2 External Forks	4132	1.16095E+21	00:02	14076
2 Trains and the Controller	203	5.16E+13	00:00	8956
1 Train and the Controller and an external Train	231	2.38879e+09	00:00	8612
3 Readers and 1 Writer	3293	2.60919e+23	00:02	18288
External Reader and Writer	180	1.81399e+09	00:00	8664

Table 1: Computing reachable states by SMV

and synchronization between rebecs. The properties that were checked are not mentioned here, but the CPU time and memory used by SMV for computing total and reachable states are summarized in Table 1.

Dining philosophers We modelled the dining philosophers example as a case study and translated it to SMV. Also, we modelled a component including two philosophers and a fork shared by them, plus two external forks. We noticed that we need to slightly change the code of rebecs to operate appropriately under new conditions. Then, we also rewrote the closed version of the problem with these changes, in order to compare the results properly. The results in Table 1 show the efficiency of our abstraction technique. The lengths of message queues are three, for both philosophers and forks.

Trains and the bridge controller In this example there are two trains travelling opposite to each other. There is a bridge in the path, which is not wide enough to accommodate both trains. There is also a bridge controller which has to prevent collisions between the two trains. We checked the queue-overflow condition and found out that queue length of two for the trains and four for the bridge controller is enough for preventing overflow.

Readers and writers This is the typical example of a data buffer that multiple readers can read from it, but only one writer can write into it. Here, we need a message queue of length two for both readers and writers, and four for the data buffer.

5 Future Work

In this paper, we present abstraction techniques used for formal verification of an object-based language, Rebeca. Results of model checking of three case studies, using our Rebeca to SMV translator, are included. We have an ongoing project for extending our tool to automatically generate component-based code as well

as closed system code. Also, nondeterministic statements and inline property specifications are going to be added to Rebeca.

We are working on other abstraction techniques to overcome the problems with message queues. We are also translating Rebeca code to Promela which is the modeling language of Spin model checker [12] and has more high level constructs in it.

References

- [1] Agha G., *Actors: A Model of Concurrent Computation in Distributed Systems*, The MIT Press, 1986.
- [2] Agha G., Mason I., Smith S. and Talcott C., A Foundation for Actor Computation, *Journal of Functional Programming*, No. 7, pp. 1-72, 1997.
- [3] Clarke E.M., Grumberg O. and Peled D.A., *Model Checking*, MIT Press, 1999.
- [4] Gagnon E.M. and Hendren L.J., SableCC, an Object-Oriented Compiler Framework, in *Proceedings of TOOLS'98*, pp. 140-154, 1998.
- [5] Hewitt C., *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot*, PhD Thesis, MIT, 1971.
- [6] Manna Z. and Pnueli A., *Temporal Verification of Reactive Systems (Safety)*, Springer-Verlag, 1995.
- [7] McMillan K., *Compositional Systems and Methods*, in *Verification of Digital and Hybrid Systems*, Springer-Verlag, 2000.
- [8] NuSMV User Manual, <http://nusmv.irst.itc.it/NuSMV/userman/index-v2.html>
- [9] Sirjani M. and Movaghar A., An Actor-Based Model for Formal Modelling of Reactive Systems: Rebeca, Technical Report, Department of Computer Engineering, Sharif University of Technology, <http://mehr.sharif.edu/~msirjani>, 2001.
- [10] Sirjani M, Movaghar A. and Mousavi M.R., Compositional Verification of an Object-Based Reactive System, in the *Proceedings of the Workshop on Automated Verification of Critical Systems (AVoCS'01)*, Oxford University Computing Laboratory, PRG-RR-01-07, April 2001.
- [11] Sirjani M. and Movaghar A., Simulation in Rebeca, in the *Proceedings of PDPTA 2002*, Vol. 2, CSREA Press, pp. 923-926, June 2002. (Also presented at the Workshop on Automated Verification of Critical Systems (AVoCS02), University of Birmingham, April 2002.)
- [12] Spin User Manual, <http://spinroot.com/spin/whatispin.html>