

An Example Demonstrating the Requirement of Fully Formal Verification Method

Pratima K. Shah^{*1} *Pauline N. Kawamoto*^{*2}

^{*1}Graduate School of Science and Technology,
Shinshu University
Nagano ken, Nagano shi, Wakasato 4-17-1, 380-8553, Japan
11st254g@shinshu-u.ac.jp

^{*2}Faculty of Engineering, Department of Information Engineering
Shinshu University
Nagano ken, Nagano shi, Wakasato 4-17-1, 380-8553, Japan
Fax: +81 26-269-5591
pauline@cs.shinshu-u.ac.jp

Abstract

Concurrency and synchronization is an important behavior of hardware and software systems. Finding defects in the late phase of system development process is not acceptable anymore in industrial applications because of its highly capital investment in modern complex systems. This paper demonstrates system verification by two approaches: model checking and proof checker. Here verification of a concurrent running architecture is demonstrated in both approaches to exhibit the requirement of fully formal verification method in order to generate a better hardware and software quality. To support this, we will describe how the proof checker is used for complete verification of a system in terms of any input size and complexity.

Index Terms - concurrency and synchronization, formal methods, proof checker, model checking.

1. Introduction

For many years, researchers have been using formal methods to generate a reliable software and hardware systems. The advantage of using this method is to find system defects as early as possible and eventually cutting the efforts and budgets needed for redesigning any systems. System failure not only results budget loss, but also causes unwanted life loss. The recent example of budget loss is the presence of Knight Capital's computer bugs in an automated trading program cost its proprietor 440 million dollars in 30 minutes which took place on August 2012 [1]. The reason of most system failures is because of a lack of guard against unexpected inputs and sometimes also because of mismatch between model specification and actual software, which could have been prevented by a

lot of attention to strict coding as well as thorough reviews which can be partly done by computer and partly by human. Model checking (MC) is most popular and widely used technique in many industrial applications because its system specification is quite similar to programming which does not require additional expertise to learn and also its verification process is typically fast and fully automatic. Moreover, MC technique depends on the search of the state space of the system transitions. Unfortunately, this may increase with the size of a system inputs and the complexity of the concurrent system description resulting the state space explosion problem. This restricts the scope of model checking techniques for sophisticated concurrent systems. An alternative to this technique is fully formal verification method called proof checker where a system is modeled as a mathematical definition in some formal mathematical rules and logic. The desired properties of a system are then derived as a theorem, that needs to be proved and verified. Here every line of the proof is explicitly justified and hence is fully completed and readable. Thus, according to [2], when a system is said to be correct by fully formal verification method, means all of its behavior is verified and the query of missed behavior become irrelevant. Despite of knowing this fact, this technique is not very popular in many industrial areas because its user needs an extra effort to learn this technique and coding span is also too long. Whereas mathematicians and computing scientists believe that this is the ultimate way of fully verifying any system.

Considering model checking in section 2, this paper demonstrates the specification of pipelined concurrent processing architecture by using LOTOS specification language [3] whose verification is done by CADP Toolbox [4]. Its verification result exhibits how the search of desired properties in the state transition diagram becomes difficult if the number of inputs or the complexity of specification increases. Then in section 3, we will see how fully formal verification method is capable of verifying any complex system and any input size. Here the verification is done by the Mizar System [5].

2. LOTOS and CADP Toolbox

There are Communication Sequential Processes (CSP) as a model of message passing parallel computation [6]. The ISO standard LOTOS abbreviated for Language of Temporal Ordering Specification is a process algebra used to describe CSP and synchronizing by rendezvous on gates. LOTOS was developed as a language by adding abstract data type to CSP. It seems to be suitable for the connection of multiple arithmetic elements which perform the concurrent operation and the description of their behaviors.

2.1 LOTOS specification of Dynamic Reloading Architecture

In this architecture, while the processor is processing the data with one set of coefficients, we can reload another set without halting the core processing. In order to show the increase in state space, first the model is fixed for three

concurrently running processes. The LOTOS code for coefficient reloading is shown in code 1. The pictorial diagram of before and after reloading coefficients is shown in figure 1 and 2. Coefficients W4, W5, W6 in figure 2 denotes the reloaded coefficients.

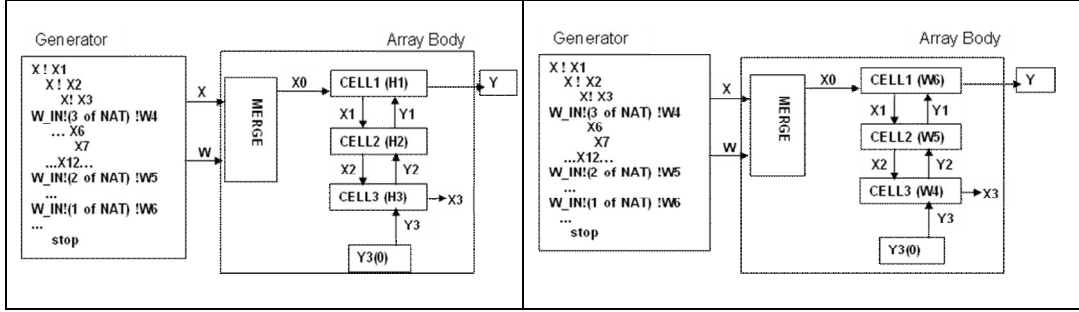


Figure 1. Model before coefficient reloading. Figure 2. Model after new coefficient reloading.

```

process CELL [X_IN, Y_OUT, X_OUT, Y_IN] (W:EXP, K:NAT, NUM:NAT) :noexit :=
.....
X_IN? X: EXP? DIST: NAT? W_IN: EXP;
Y_IN? Y: EXP;
(
  [DIST eq NUM] ->
  (
    X_OUT !X;
    Y_OUT ! (Y+(W_IN *X));
    CELL [X_IN, Y_OUT, X_OUT, Y_IN] (W_IN, 1, NUM)
  )
[]
  [DIST ne NUM] ->
  (
    X_OUT !X !DIST !W_IN;
    Y_OUT ! (Y + (W*X));
    CELL [X_IN, Y_OUT, X_OUT, Y_IN] (W, 1, NUM)
  )
) .....

```

Code 1: LOTOS code for new coefficient reloading

2.2 CADP Tool for verification of Model

Among several CADP's features, Caesar and Aldebaran are used for compiling and generating the labeled transition graph, where we can search the specified properties. The result shown in figure 3 is the reduction of bcg graph of the above code specification. Note this figure is fixed for 3 concurrent running processor, where we can easily see how and where the new coefficient is reloaded, but if we increase the number of processors, say 6 or more, it is comparatively not easy to track the properties because of the exponential increase in computation level. In our case it consumed almost more than hours to even generate its bcg graph for 6 cells, which is not really practical [12]. The graph of figure 3 shows that the new coefficient W4 is reloaded in cell 3 at the

transition from state 6 to state 16 replacing the previous coefficient H3. If we see clearly in the state transition graph, it is still difficult to trace another new reloaded coefficients even in the reduced version of graph for 3 cells, because of large number of X input computation after reloaded W4 coefficient.

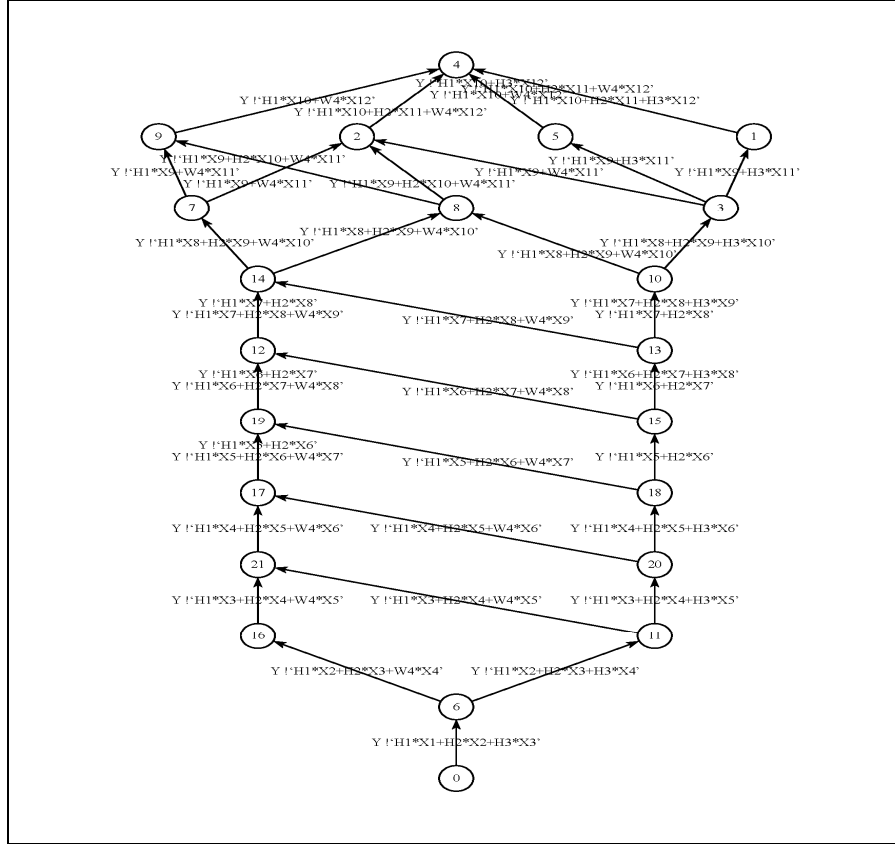


Figure 3. Reduced LTS graph of above LOTOS specification.

3 MIZAR System

This section presents a model verification by the Mizar System. Mizar is one among several proof assistant tools developed by Andrzej Trybulec since 1973 at the university of Bialystok in Poland [7]. Mizar tool consists Mizar language for writing the mathematical definitions of a hardware and software systems in a first order declarative logic style and a program to check files written in Mizar language for complete verification or logical correctness of the system [7], [8]. To demonstrate the verification of a system in Mizar, we considered an event graph which is a restricted version of Petri nets (PNs) [9]. PNs is a graphical and mathematical modeling tool, that provides an environment for designing and formal analysis of any discrete event systems. Event graph has a characteristic that every state in the system, there is exactly one input arc and one output arc as

shown in Figure 4. For detail description of Petri nets and its notations, readers are requested to refer [9]. But for the quick knowledge about Petri nets, it contains 4 elements: places, transitions, arcs, and tokens. Here the places are denoted by circle, transitions by bar or rectangle, and the tokens by a black dotted circle. Places and transitions are connected by arcs with the rule that arcs can only connect places to transitions and vice versa. In event graph, if the starting and the ending elements are same, we called that is directed circuit. Here in Petri net according to rules, tokens will always reside in places and the process of assigning tokens is called marking. The movement of tokens in event graph will only take place if the number of tokens in its only input place is greater than or equal to 1 since the number of input and output arc here is exactly equal to 1 as shown in Figure 4. After transition firing, it removes exactly 1 token from its only input place and adds 1 token to its only output place. Now let us consider a desired property, stating that firing any transitions in the directed circuit preserves the number of tokens. As we can see that the number of tokens remains same, i.e. 2, before and after firing transition t2 as given in figure 4 (a) and (b). For more detail presentation of this, authors online paper is available at [10], which presents every small details needed during formalizing this property. Mizar code of the above desired properties is shown in code 2.

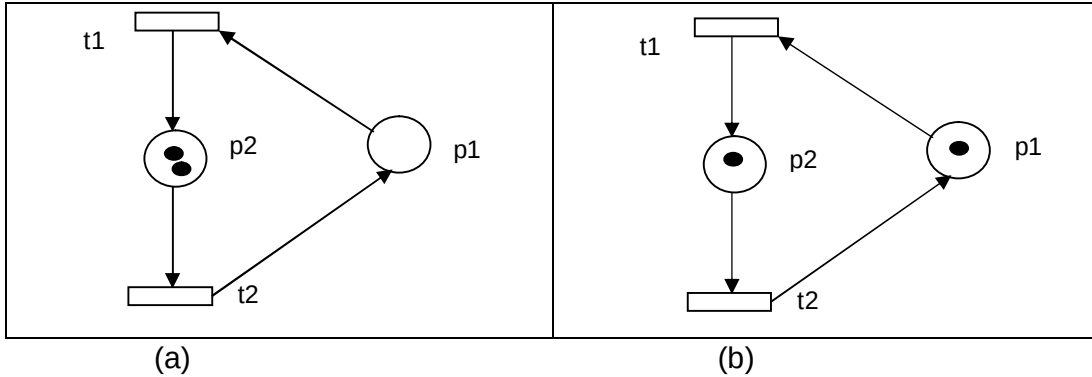


Figure 4. Token changes before and after firing transition t2

```
for Dftn being Decision_free_PT, dct being Circuit_of_places_and_trans of Dftn,
M0 being marking of Dftn, t being transition of Dftn holds num_marks (places_of
dct, M0) = num_marks (places_of dct, Firing(t, M0));
```

Code 2. Mizar code of above described properties

No doubt, it needs intense effort and concentration to verify even this single property [10] in fully formal verification approach. The definitions, pre-theorems, and lemmas of this property are completely verified and accepted to be available in the library of the Mizar system called Mizar Mathematical Library (MML) [11].

The main point of taking this as an example is to show that once the desired properties is fully verified in any proof checker, it holds for any input size and system complexity. In this case the above stated verified property is true for any number of places, transitions, and tokens in any circuit of the event graph PNs.

Conclusion

This paper concludes that if the desired properties of hardware and software system is successfully verified by fully formal verification method, it holds for any number of input size in terms of system complexity. Whereas, since searching the desired properties in the state transition graph of model checking tools is fully automatic and fast, but still it is quite difficult if the system complexity and its computational level increases.

References

- [1] Available. <http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/>
- [2] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled; Model Checking. The MIT Press Cambridge, Massachusetts London, England (1999).
- [3] ISO International Standard 8807:1989, Information processing systems-Open Systems Interconnection –LOTOS – A formal description technique based on the temporal ordering of observational behavior, Geneva, September 1989.
- [4] CADP (Caesar/Aldebaran Development Package), A Software Engineering Toolbox for Protocols and Distributed Systems, INRIA/VASY, France, <http://www.inrialpes.fr/vasy/cadp>.
- [5] Available <http://mizar.org/>
- [6] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
- [7] Trybulec, A., The Mizar-QC/6000 Logic Information Language, ALLC Bulletin, 6 (1978), No 2.
- [8] Trybulec, A., The Mizar Logic Information Language, Studies in Logic, Grammar and Rhetoric, 1 (1980), Bialystok.
- [9] T. Murata. (1989, Apr). Petri nets: Properties, analysis and applications. *Proceeding of the IEEE*, vol. 77, no. 4, pp.541-580. Available: http://embedded.eecs.berkeley.edu/Respep/Research/hsc/class.F03/ee249/discussionpapers/Petri_Nets.pdf
- [10] Shah, Pratima K., Kawamoto, Pauline N., and Giero, Mariusz. The Formalization of Decision Free Petri Net. FORMALIZED MATHEMATICS, Vol. 22, No. 2, pp. 29–35 (2014). Online: http://fm.mizar.org/fm22-1/petri_df.pdf.
- [11] <http://www.mizar.org/library/>
- [12] Shah, Pratima K. 信州大学大学院工学系研究科修士学位論文 (2011) "Automatically Generation of N bit Circuit Components in Extended LOTOS from High-Level Functional Programming Language HDCaml"