A Technique for Parameterized Verification of Cache Coherence Protocols

V.S. Burenkov <burenkov_v@mcst.ru>
JSC MCST, 24 Vavilov str., Moscow, 119334, Russian Federation

Abstract. This paper introduces a technique for scalable functional verification of cache coherence protocols that is based on the verification method, which was previously developed by the author. Scalability means that verification efforts do not depend on the model size (that is, the number of processors in the system under verification). The article presents an approach to the development of formal Promela models of cache coherence protocols and shows examples taken from the Elbrus-4C protocol model. The resulting formal models consist of language constructs that directly reflect the way protocol designers describe their developments. The paper describes the development of the tool, which is written in the C++ language with the Boost.Spirit library as parser generator. The tool automatically performs the syntactical transformations of Promela models. These transformations are part of the verification method. The procedure for refinement of the transformed models is presented. The refinement procedure is supposed to be used to eliminate spurious error messages. Finally, the overall verification technique is described. The technique has been successfully applied to verification of the MOSI protocol implemented in the Elbrus computer systems. Experimental results show that computer memory requirements for parameterized verification are negligible and the amount of manual work needed is acceptable.

Keywords: multicore microprocessors; shared memory multiprocessors; cache coherence protocols; model checking; Spin; Promela.

DOI: 10.15514/ISPRAS-2017-29(4)-15

For citation: Burenkov V.S. A Technique for Parameterized Verification of Cache Coherence Protocols. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 231-246. DOI: 10.15514/ISPRAS-2017-29(4)-15

1. Introduction

Shared memory multiprocessors constitute one of the most common classes of high-performance computer systems. In particular, multicore microprocessors, which combine several processors (cores) on a chip, are widely used [1]. The number of cores is constantly increasing. The presence of cache memories that are local to each core determines the need for ensuring coherent memory state. To satisfy the need, microprocessor developers design and implement in hardware cache coherence protocols [2].

Cache coherence mechanisms are extremely complex. Therefore, both the design and their implementation are error-prone. Being especially critical, protocol bugs should be revealed before implementing the hardware. The widely recognized method for protocol verification is *model checking* [3]. It is fully automated, but suffers from a principal drawback – it is not scalable due to the *state space explosion* problem. Verification of a cache coherence protocol for five or more processors is impossible (at least, highly problematic) with the traditional methods [4].

To overcome the problem and develop scalable verification technologies, researchers focus mostly on verification of *parameterized designs* [3]. Previous articles of the author [5–8] presented a method for parameterized verification of cache coherence protocols. The author successfully applied the method to verification of the cache coherence protocol of the Elbrus-4C computing system. This paper presents an approach to the development of formal Promela models that can be analyzed by the verification method, describes the development of the tool that performs transformations of Promela models according to the method and presents the overall verification technique.

The paper is structured as follows. Section 2 takes a brief look at related work and provide the necessary links. Section 3 considers the question development of Promela models of cache coherence protocols. In Section 4, we describe how to perform parameterized verification of the Promela models in a semi-automatic way. We examine the development of the tool that automates parts of the verification method used. We present a technique for cache coherence protocols verification. Section 5 provides experimental results on using the technique for verifying the Elbrus-4C protocol. Section 6 summarizes the work and defines further research directions.

2. Related Work

This work extends the previous works [5–8] by dealing with the question of practical application of the method for parameterized verification of cache coherence protocols presented in those works.

Article [5] presents a review of related work and gives the motivation for development of a new method. The developed method is based upon works [9–13] that present a method of compositional model checking, which is based on syntactical transformations of models written in the Murφ language and counterexample-guided abstraction refinement.

The method [5–8] is used in the context of the following verification process:

1) Development of formal models of cache coherence protocols.
2) Parameterized verification by means of the method.

3. Development of Formal Models

It is highly desirable to have a modeling language that allows us to conveniently describe cache coherence protocols. To choose or develop such a language, we need to define a mathematical model of cache coherence protocols.

In accordance with the microprocessor system model that is used in work [2] for representation and analysis of cache coherence protocols, I chose to model cache coherence protocols as a set of communicating finite-state machines.

An element of this set may be either a cache controller or the system commutator. Let us define these notions. Each memory device of the microprocessor is operated by a *coherence controller*, which is a finite-state machine. Coherence controllers are coordinated by a special device – the *system commutator* – that is also a finite-state machine. A set of these machines constitutes a distributed system, in which the machines communicate by message passing in order to maintain cache coherence.

Each coherence controller connected with cache memory logically implements a set of independent and identical finite-state machines, one for each cache line. These machines are called *cache controllers*. Due to the independence and identity of cache controllers, it is customary to reflect only one cache line in the models of cache coherence protocols.

The states of cache controllers are divided into two classes: Stable states and transient states. Stable states of cache controllers are often the subset of the common set Modified, Owned, Exclusive, Shared, Invalid [2]. Transitions between these states are not atomic and occur through transient states. Transient states are specific to each microprocessor and their presence is one of the factors that determine high verification complexity.

Conditions that define correctness of cache coherence protocols are formulated as statements about stable states, for example: “Cache line can never be in Modified state in two caches simultaneously” [5]. Such statements belong to the class of invariant properties [14].

Usage of a set of communicating finite-state machines as the model of cache coherence protocols and invariant properties for specification defined the choice of the Promela language for modeling cache coherence protocols:

- In contrast to other languages (for example, Murφ and NuSMV), Promela provides process types and the means of synchronous and asynchronous interprocess communication (channels).
- Promela provides convenient specification language, which is Linear Temporal Logic (LTL).
- Spin – the system that implements Promela – provides different verification algorithms and optimizations, and is a modern and constantly developing tool.

The question of development of formal models of cache coherence protocols is insufficiently covered in the literature. Here, I present an approach to the construction of such models. According to the approach, a formal model of a cache coherence protocol of a system with n cores consists of n Promela processes for cache controllers and one Promela process for the system commutator.

For the considered cache coherence protocols, the following property holds: Only one initial request may be in process at a given point in time. System commutator performs a sequence of steps during the request processing, for example, the reception of the initial request and its analysis, sending of snoop- and other requests according to the results of the analysis, reception of the answers to these requests. Initial requests correspond to the memory access instructions that the processor core is executing. Reception of messages from other devices can only occur at particular steps. Thus, it is convenient to represent the system commutator as a Promela process whose body simply consists of operators that follow each other (Fig. 1).

```
proctype system_commutator() {
again:
 <receive initial request>
 <analyze the initial request>
 <send coherent requests>
 <receive answers to coherent requests or the request completion message>
 <finalize the request processing>
goto again }
```

Fig. 1. Structure of the System Commutator Process.

Cache controllers operate differently. On the one hand, we still may identify a number of steps, for example, sending an initial request, changing state from stable to transient, receiving snoop-requests. On the other hand, the relative order of these steps is often unspecified, and the same messages from other devices may be processed in different states of a cache controller. Thus, it is convenient to represent processes of this kind as infinite do-cycles consisting of the guarded commands (Fig. 2).

```
proctype cache_controller() {
 do
 :: <send initial request from main states>
 :: <receive and process snoop-requests>
 :: <receive answers to coherent requests>
 :: <send the completion message>
od }
```

Fig. 2. Structure of Cache Controller Processes.

See papers [5, 6, 8] for more details on how to organize processes and their communication.

For example, modeling of a situation in which cache controller sends an initial request and the system commutator receives it, may be performed as follows:

```
mtype cache[N] = I; // states of cache line
proctype cache_controller(byte i) {
do
:: atomic {cache[i] == I ->
// send initial request and change state
if :: ini_req_chan ! R, i; cache[i] = WR;
   :: ini_req_chan ! RI, i; cache[i] = WRI;
      ...
fi }
...
od }


proctype system_commutator(byte i) {
message_t message;
again:
// receive initial request
atomic {ini_req_chan ? message;
           curr_command = message.opcode;
           curr_client = message.requester;
}
if :: atomic {
// send snoop-request as a response
// to the initial request
curr_command == R ->
    coh_req_chan[0] ! snR, curr_client; ...
}
...
// receive acknowledgement
final_ack_chan ? message;
goto again; }
```

As another example, reception of a snoop-request by cache controller and generation of the response can be modeled as follows:

```
proctype cache_controller(byte i) {
do
...
:: atomic {nempty(coh_req_chan[i]) ->
    // receive snoop-request
    coh_req_chan[i] ? message;
if ...
   // analyze state...
:: cache[i] == WI_O
  // ... and the snoop-request type
  && message.opcode == snI ->
    // send corresponding answer
    coh_ans_chan ! ack, i;
    cache[i] = WRI;
... fi }
... od
}
```

Developers of cache coherence protocols describe and reason about their protocols in terms of message passing, and, as these examples show, their reasoning can be directly expressed in Promela. Moreover, the proposed organization of Promela processes allows verification engineers to perform quick changes that are needed to reflect the modifications of the cache coherence protocol under verification that occur in the course of its development.

4. Parameterized Verification of Cache Coherence Protocols

The method for parameterized verification of cache coherence protocols presented in works [5, 6, 8] consists of two stages:

1. Performing the syntactical transformations of Promela models.
2. Refining the obtained model in accordance with the proposed procedure.

Model transformations have the following effect:

1. Reduction of the number of processes from n+1 (n cache controller processes and one system commutator process) to 4: two fully functioning cache controller processes, one abstract cache controller process that models the environment of the two processes, and the system commutator process. This transformation is possible due to the symmetry inherent in models of cache coherent protocols (all cache controller processes are identical and interchangeable, they do not have behaviors that depend on a particular process index value) and because the specification of cache coherence protocols only contains properties that regard the state of cache line in two caches.
2. Syntactical transformations of Promela operators constituting the model.

These transformations preserve invariant properties. This means that if such a property is true for the reduced model, then it is true for the initial model. A mathematical proof of the corresponding theorem is presented in articles [5, 6, 8].

4.1 Performing the Syntactical Transformations

The syntactical transformations presented in [5, 6, 8] may be performed manually. However, manual model modification is a very tedious, laborious and error-prone process. Moreover, some of the errors made may go undetected, as they will only lead to incorrect state space reduction and not to counterexamples. Therefore, it is highly desirable to perform the transformations automatically. To achieve that, I have developed a dedicated tool. With this tool, the verification engineer simply provides their Promela model as input to the tool, and the tool generates the transformed Promela model.

To automate the syntactical transformations, I have used a widespread approach to this kind of problems, according to which a tool builds the abstract syntax tree that represents the syntactical structure of the source code and then performs the transformations upon the tree traversal (Fig. 3).

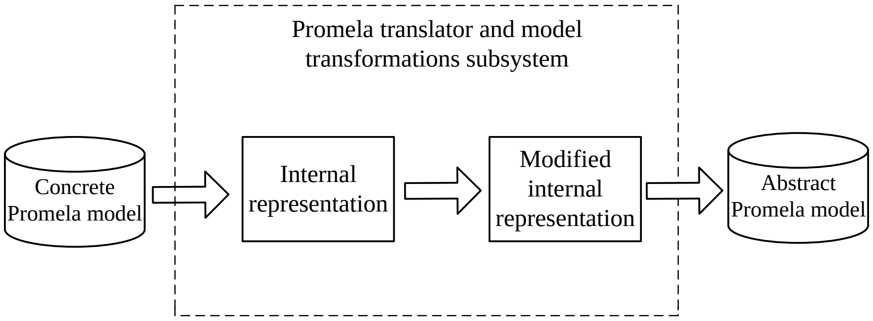
Promela translator and model transformations subsystem
Concrete Promela model
Internal representation
Modified internal representation
Abstract Promela model

Fig. 3. Scheme of Automated Model Transformation.

Abstract syntax trees are usually constructed by parsers. There are two ways of parser implementation: manual and by means of a parser generator tool (for example, Bison, ANTLR, Boost.Spirit). Due to the unnecessary complexity of the first approach, I have chosen the second one.

The Boost.Spirit library was chosen as the parser generator, because:

- Boost.Spirit promotes modern usage of the C++ language that allows us to work with abstractions, which are suitable for a given domain, without performance loss.
- Boost.Spirit eliminates the need for additional tools like Bison or ANTLR: The only tools needed are a C++ compiler and the Boost library.
- The grammars that Boost.Spirit accepts are attributed, which results in a very convenient way of abstract syntax tree generation.

- Boost.Spirit contains a number of built-in parsers.
- The generated parsers are very efficient [15].

The mechanism of synthesized and inherited attributes allows us to simplify the task of abstract syntax tree generation by dividing it into two sequentially performed subtasks:

1. Development of the grammar, testing and debugging of the grammar. During this step, we only need to focus on the question of whether the grammar can correctly determine the syntactical correctness of a Promela model.
2. Development of data structures for the nodes of the abstract syntax tree and definition of the types of attributes of the grammar rules. The attribute mechanism allows Boost.Spirit to generate abstract syntax trees automatically, without any need for the addition of node construction operators to the grammar.

Usage of the abstract syntax tree generated by Boost.Spirit as an intermediate representation of Promela models allowed us to divide the task of performing the syntactical transformations automatically into three subtasks:

1. Development of Promela grammar in the C++ language by means of Boost.Spirit.
2. Development of data structures for abstract syntax tree representation.
3. Development of algorithms for abstract syntax tree traversal and abstract model generation.

Promela grammar is presented in [16]. Its implementation in C++ using Boost.Spirit looks similarly to that description. However, as Boost.Spirit generates recursive descent parsers, I have eliminated left recursion from the grammar.

Data structures for the nodes of abstract syntax tree are developed according to the information that we want the nodes to represent and attribute propagation rules defined in Boost.Spirit's documentation. In the developed tool, data structures that correspond to the synthesized attributes of the Promela grammar rules, contain information about nonterminals that are part of the rules. This is a very straightforward and convenient way of implementation of these data structures. For example, the following rule that describes the nonterminal "module" of the Promela grammar

```
qi::rule<Iter, module(), Skipper> module;
module =
    proctype
    | init
    | ltl
    | utype
    | mtype
    | decl_lst
    | ';' ;
```

has a synthesized attribute of type `module`, which is implemented as follows:

```
using module = boost::variant<
    proctype,
    init,
    ltl,
    utype,
    mtype,
    decl_lst
    >;
```

All the other nonterminals mentioned in this example have synthesized attributes of types implemented in a similar way.

The abstract syntax tree, which is generated automatically by Boost.Spirit based on the grammar and the attribute mechanism, consists of nodes of different types. Traversal of such tree is performed uniformly by means of visitors, as advocated by the Boost.Spirit documentation.

The syntactical transformations are performed during the abstract syntax tree traversal. I classified the transformations, most of which turned out to be in one of the three categories (transformations of assignments, transformations of expressions, transformations of communication actions), and precisely described them. To automatically carry them out, I have developed a number of abstract syntax tree modification algorithms and implemented them as part of the visitation mechanism. Printing out the modified syntax tree gives us the abstract Promela model.

For example, when generating the code for the abstract process, the following piece of Promela code

```
proctype cache_controller(byte i) {
do
...
:: (cache[i] == M_MAU || cache[i] == M_MAU_I)
&& (message.opcode == wb_ready) ->
    final_ack_chan ! data, i;
    cache[i] = I
```

is transformed into

```
proctype cache_controller_abs(byte i) {
do
...
:: true ->
```

```
        final_ack_chan ! data, i;
```

This example demonstrates the transformations of expressions and the assignment operator.

4.2 Abstraction Refinement

Execution of each type of initial requests consists of a particular sequence of events presented in the cache coherence protocol documentation. Considerations about the ordering of the events inspired the following refinement procedure:

1. For each type of initial requests define (according to the documentation) a partially ordered set $(A, \prec)$ of events ($\prec$ is a strict partial order):

   $\forall a_1, a_2 \in A: a_1 \prec a_2$, if action a_1 occurs earlier than action a_2.

2. While there are false counterexamples:

   2.1. Find action a that lead to the appearance of the counterexample. Find set A that contains action a: $a \in A$. In set A find action b such that $b \prec a$.

   2.2. Introduce a logical variable aux_b with the initial value $false$. In the model, replace b with the atomic sequence $b; aux_b \coloneqq true$.

3. By means of the logical AND, add aux_b to the guard of the command that contains action a. Replace a with the atomic sequence $a; aux_b \coloneqq false$.

For example, for one type of initial requests defined for the Elbrus-4C microprocessor, the set $(A, \prec)$ is as follows. Here, cc_i denotes the ith cache controller.

$\{a_0 =$ processing of the previous request from process $cc_i, 1 \le i \le n$ is finished,

$a_1 =$ requester cc_i sends an initial request,

$a_2 = system\_commutator$ receives the initial request,

$a_3 = system\_commutator$ sends snoop-requests to all $cc_j, 1 \le j \le n, j \ne i$,

$a_4 = cc_j$ receives a snoop-request, $1 \le j \le n, j \ne i$,

$a_5 = cc_j$ sends an answer to the snoop-request to the requester,

$a_6 =$ the requester receives the coherent answer from cc_j,

$a_7 =$ the requester sends the operation completion message to $system\_commutator$,

$a_8 = system\_commutator$ receives the operation completion message$\}$.

The relation $\prec$ is defined as follows: $\forall i, j = 0, \ldots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. We identify the auxiliary variables with the elements of the set A.

Refinement of the abstract model of the Elbrus-4C cache coherence protocol required us to introduce two auxiliary variables, because there were two spurious counterexamples. Let us examine the introduction of the first variable.

The analysis of the first counterexample showed that the abstract process had sent the operation completion message to $system\_commutator$ before $system\_commutator$ received a coherent answer. Examination of the set A allows us to conclude that action a_7 happening at the wrong time led to the counterexample. According to the refinement procedure, in the set A we find action a_6 and introduce an auxiliary variable `ack_received` with the initial value $false$. Then we replace the operator that corresponds to a_6 with the atomic sequence consisting of this operator and the operator that assigns $true$ to `ack_received`. After this, we add `ack_received` to the guard of the command of the abstract process that contains a_7 and replace the operator that corresponds to a_7 with the atomic sequence consisting of this operator and the operator that assigns $false$ to `ack_received`. Thus, we guarantee that the behavior of the abstract process that led the false counterexample will no longer be exhibited.

4.3 Verification Technique

According to the results obtained by the author in this and the previous works, the proposed verification technique consists of the following steps (Fig. 4):

1. Development of a concrete Promela model of the cache coherence protocol under verification. Using the proposed approach to model description, verification engineer develops Promela processes that model cache controllers and the system commutator and the necessary infrastructure elements (channel definitions, process creation). Specific actions performed by the processes correspond to the cache coherence protocol documentation.

2. Development of the abstract Promela model of the cache coherence protocol under verification. This step is performed automatically by the developed tool.

3. Verification of the abstract model. This step is the usual verification process of Promela models using the Spin model checker [17].

4. Analysis of the verification report generated by Spin. If there are no errors, then the verification process is finished with the conclusion that the cache coherence protocol is correct. If the report states the presence of an error, then the verification engineer should analyze the corresponding counterexample. If the engineer concludes that the counterexample is spurious because the corresponding sequence of steps is impossible in a real system, then the engineer refines the model in accordance with the proposed procedure and goes to step 3. Otherwise, if the counterexample represents an actual error in the cache coherence protocol, then the error is reported. When the protocol developers fix the error, the verification engineer incorporates the changes into the model and starts the verification process again (goes to step 1).

This sequence of steps is repeated until there are no counterexamples.

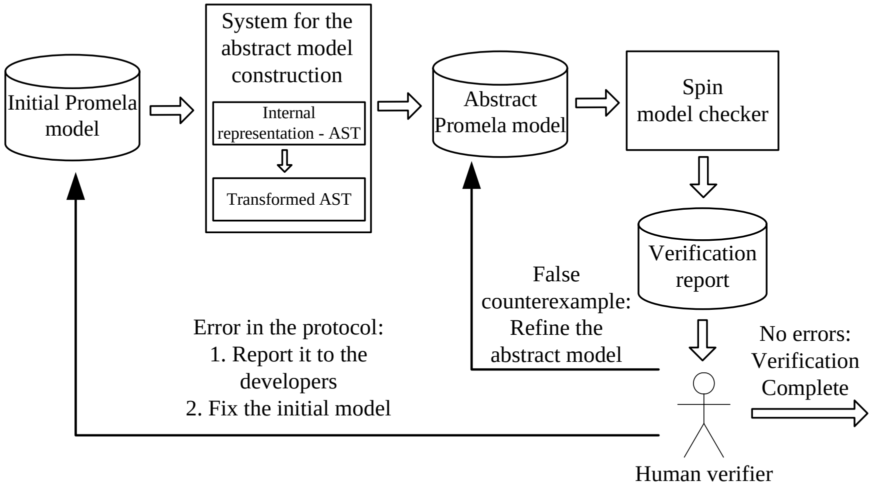
Initial Promela model
System for the abstract model construction
Internal representation - AST
Transformed AST
Abstract Promela model
Spin model checker
Verification report
False counterexample: Refine the abstract model
Error in the protocol: 1. Report it to the developers 2. Fix the initial model
No errors: Verification Complete
Human verifier

Fig. 4 Scheme of the Verification Process.

5. Experimental Results

The proposed method was used to verify the MOSI family cache coherence protocol implemented in the Elbrus-4C computer system. The abstraction refinement step was completed after the introduction of two auxiliary variables.

Table 1 and Table 2 show resources consumed for checking the property

$$\mathbf{G}\{\neg(cache[1] = M \wedge cache[2] = M)\},$$

respectively, on the original and the refined abstract model. Spin's optimization COLLAPSE was used. The experiments were performed on an Intel Xeon E5-2697 machine with a clock rate of 2.6 GHz and 264 Gb of RAM.

Table 1. Required resources — initial model

Number of cores	State space size	Memory consumption	Verification time
3	5.1×10^6	328 Mb	15 s
4	1.3×10^9	81 Gb	1.5 h

Table 2. Required resources — abstract model

Number of cores	State space size	Memory consumption	Verification time
any > 2	2.2×10^6	108 Mb	6.2 s

Tables 1 and 2 show that even for $n = 3$ there is a gain in state space size and memory consumption. The needed amount of manual work is acceptable. Meanwhile, verification of the constructed abstract model means verification of the protocol for any $n \geq 3$. The task has been reduced to checking of $\sim 10^6$ states, which consumes ~ 100 Mb of memory.

6. Conclusion

Many high-performance computers and most multicore microprocessors use shared memory and utilize complicated caching mechanisms. To ensure that multiple copies of data are kept up-to-date, cache coherence protocols are employed. Errors in the protocols and their implementations may cause serious consequences such as data corruption and system hanging. This explains the urgency of the corresponding verification methods.

The main problem when verifying cache coherence protocols (and other systems with a large number of components) by a fully automated method of model checking is state explosion. The article proposes a technique to overcome the problem for cache coherence protocols and make verification scalable. The price paid for scalability is acceptable, because the main ingredient – the verification method – is highly automated by the developed tool. Part of the method that requires manual work, namely, model refinement, can be done with a reasonable amount of effort, as shown by means of the Elbrus-4C protocol verification example. An approach to describing protocol models in Promela, a widely spread language in the verification community, is proposed. This approach lets us reflect the way protocol designers talk about protocols by representing protocols as a set of communicating finite-state machines.

The technique was successfully applied to the verification of the MOSI family cache coherence protocols implemented in the Elbrus-4C computer system.

Directions for future research include:

1. Development of methods and tools for verification of cache coherence protocols that are implemented by multiple levels of cache. The newest microprocessors (for example, Elbrus-8C, which employs the second- and third-level caches to implement cache coherence) define the need for such methods and tools.

2. Development of methods and tools for verification of hardware implementations of cache coherence protocols. In this direction, I have developed a tool that generates assembly code based on Promela models of cache coherence protocols. With this tool, I have found several dozen errors in the implementation of cache coherence in Elbrus microprocessors. Still, further research is needed to increase the level of confidence in design correctness.

References

[1]. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.

[2]. Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.

[3]. Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999. 314 p.

[4]. Burenkov V.S. Analiz primenimosti instrumenta Spin k verifikacii protokolov kogerentnosti pamyati [An analysis of the Spin model checker applicability to cache coherence protocols verification]. Voprosy radioehlektroniki. Ser. EVT. Issues of Radioelectronics, the series EVT], 2014, issue. 3, pp. 126-134 (in Russian).

[5]. Burenkov V.S., Kamkin A.S. Checking Parameterized PROMELA Models of Cache Coherence Protocols. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 57-76. DOI: 10.15514/ISPRAS-2016-28(4)-4

[6]. Burenkov V.S., Kamkin A.S. Metod masshtabiruemoi verifikacii PROMELA-modelei protocolov kogerentnosti kesh-pamyati [A Method for Scalable Verification of PROMELA Models of Cache Coherence Protocols]. Sb. trudov VII Vserossiiskoi nauchno-technicheskoi konferencii "Problemi razrabotki perspectivnih micro- i nanoelektronnih sistem" [Proceedings of the VII All-Russian Scientific and Technical Conference "Problems of Development of Advanced Micro- and Nanoelectronic Systems"]. 2016, part II, pp. 54-60 (in Russian).

[7]. Burenkov V.S., Kamkin A.S. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols. Proc. of IEEE East-West Design & Test Symposium. 2016. pp. 1-4.

[8]. Burenkov V.S., Ivanov S.R. Metod postroeniya abstraktnih modelei, ispolzuemih dlya verifikacii protocolov kogerentnosti kesh-pamyati [A Method for Construction of Abstract Models Used for Verification of Cache Coherence Protocols]. Vestnik MGTU im N.E. Baumana [Herald of the Bauman Moscow State Technical University], 2017, issue 1, pp. 49-66 (in Russian).

[9]. McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. Conference on Correct Hardware Design and Verification Methods, 2001. pp. 179-195.

[10]. Chou C.-T., Mannava P.K., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols. Formal Methods in Computer-Aided Design, 2004. LNCS, Vol. 3312, pp. 382-398.

[11]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.

[12]. Talupur M., Tuttle M.R. Going with the Flow: Parameterized Verification Using Message Flows. Formal Methods in Computer-Aided Design, 2008. pp. 1-8.

[13]. O'Leary J., Talupur M., Tuttle M.R. Protocol Verification Using Flows: An Industrial Experience. Formal Methods in Computer-Aided Design, 2009. pp. 172-179.

[14]. Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 984 p.

[15]. de Guzman, J. Fastest numeric parsers in the world! http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.

[16]. Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html.

[17]. Holzmann, G. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley. 2004. 608 p.

Методика параметризованной верификации протоколов когерентности памяти

В.С. Буренков <burenkov_v@mcst.ru>,
АО «МЦСТ», 119334, Россия, г. Москва, ул. Вавилова, 24

Аннотация. В статье представлена методика масштабируемой функциональной верификации протоколов когерентности памяти, которая основана на методе верификации, который ранее был разработан автором статьи. Масштабируемость при верификации означает независимость работ по верификации от размера модели, то есть от количества процессоров верифицируемой системы. В статье предложен подход к разработке формальных моделей протоколов когерентности памяти на языке Promela. Приведены примеры описаний, взятые из модели протокола когерентности памяти системы Эльбрус-4С. Результирующие формальные модели отражают представление протоколов когерентности памяти, используемое разработчиками таких протоколов – в виде множества взаимодействующих конечных автоматов. Описана разработка программного инструмента, написанного на языке C++ с использованием библиотеки Boost.Spirit в качестве генератора синтаксических анализаторов. Программный инструмент позволяет автоматизировать выполнение синтаксических преобразований Promela-моделей. Выполнение данных синтаксических преобразований происходит в соответствии с методом верификации. В статье представлена процедура уточнения модифицированных моделей, основанная на введении в модель вспомогательных переменных. Использовать эту процедуру предлагается в том случае, когда при верификации возникают ложные сообщения об ошибках, для устранения таких сообщений. Представлена методика верификации, которая состоит из следующих шагов: разработка исходной модели протокола когерентности памяти на языке Promela, автоматизированное преобразование данной модели согласно методу верификации, верификация модифицированной модели с помощью инструментального средства Spin, анализ отчета о верификации, сгенерированного инструментом Spin. Изложенная методика была успешно применена для верификации протокола когерентности памяти семейства MOSI, реализованного в микропроцессорной системе Эльбрус-4С. Экспериментальные результаты показали, что затраты процессорного времени и памяти на проведение параметризованной верификации незначительны, а требуемый объем ручной работы приемлем. Для уточнения модифицированной модели протокола системы Эльбрус-4С понадобилось ввести лишь две вспомогательные переменные.

Ключевые слова: многоядерные микропроцессоры; мультипроцессоры с разделяемой памятью; протоколы когерентности памяти; проверка моделей; Spin; Promela.

DOI: 10.15514/ISPRAS-2017-29(4)-15

Для цитирования: Буренков В.С. Методика параметризованной верификации протоколов когерентности памяти. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 231-246 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-15

Список литературы

[1]. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.

[2]. Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.

[3]. Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999. 314 p.

[4]. Буренков В.С. Анализ применимости инструмента Spin к верификации протоколов когерентности памяти. Вопросы радиоэлектроники. Сер. ЭВТ. 2014. Вып. 3. стр. 126-134.

[5]. Burenkov V.S., Kamkin A.S. Checking Parameterized PROMELA Models of Cache Coherence Protocols. Trudy ISP RAN/Proc. ISP RAS. vol. 28, issue 4, 2016, pp. 57-76. DOI: 10.15514/ISPRAS-2016-28(4)-4

[6]. Буренков В.С., Камкин А.С. Метод масштабируемой верификации PROMELA-моделей протоколов когерентности кэш-памяти. Сб. трудов VII Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем». 2016. Часть II. стр. 54-60.

[7]. Burenkov V.S., Kamkin A.S. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols. Proc. of IEEE East-West Design & Test Symposium. 2016. pp. 1-4.

[8]. Буренков В.С., Иванов С.Р. Метод построения абстрактных моделей, используемых для верификации протоколов когерентности кэш-памяти. Вестник МГТУ им. Н.Э. Баумана. 2017, вып. 1, стр. 49-66.

[9]. McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. Conference on Correct Hardware Design and Verification Methods, 2001. pp. 179-195.

[10]. Chou C.-T., Mannava P.K., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols. Formal Methods in Computer-Aided Design, 2004. LNCS, Vol. 3312, pp. 382-398.

[11]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.

[12]. Talupur M., Tuttle M.R. Going with the Flow: Parameterized Verification Using Message Flows. Formal Methods in Computer-Aided Design, 2008. pp. 1-8.

[13]. O'Leary J., Talupur M., Tuttle M.R. Protocol Verification Using Flows: An Industrial Experience. Formal Methods in Computer-Aided Design, 2009. pp. 172-179.

[14]. Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 984 p.

[15]. de Guzman, J. Fastest numeric parsers in the world! http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.

[16]. Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html.

[17]. Holzmann, G. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley. 2004. 608 p.