On the Implementation of a Formal Method for Verification of Scalable Cache Coherent Systems

Vladimir Burenkov <burenkov_v@mcst.ru>,
Bauman Moscow State Technical University,
105005, Moscow, Russian Federation, 2nd Baumanskaya st., 5
MCST, 119334, Moscow, Russian Federation, Vavilov st, 24

Abstract. This article analyzes existing methods of verification of cache coherence protocols of scalable systems. Based on the research literature, the paper describes a method of formal parameterized verification of safety properties of cache coherence protocols. The paper proposes a design of a verification system for cache coherence protocols. The article analyzes the method in terms of development and examination of the corresponding Promela model of the German cache coherence protocol and discusses extension and automation of the method needed to adapt it to verification challenges of the Elbrus microprocessors.

Keywords: formal verification; model checking; deductive verification; cache coherence protocol; Elbrus

1. Introduction

Modern microprocessor systems are scalable – the number of cores per chip increases and chips are combined into clusters. Each processor of the system has access to the shared address space. However, memory is physically distributed among the processors in order to increase the bandwidth and reduce the latency to local memory. Thus, access to the local memory is faster than access to the remote memory. To decrease the memory bandwidth demands of a processor, processors are equipped with multilevel caches. Caching of shared data introduces the problem of cache coherence.

To solve the problem, computer architects often use hardware mechanisms that implement cache coherence protocols. Concurrent work of many hardware devices (for example, cache and main memory controllers), which exchange information in accordance with a cache coherence protocol, results in a colossal size of the protocol's state space. This, in turn, makes verification of cache coherence protocols an extremely hard task.

To work out the problem, scientists have been conducting research in the direction of formal methods for the past few decades and achieved a level of success. However, scalable verification is still an issue.

Scalability leads to the need for formal verification methods that are capable of adapting to it. As the size of systems increases, the fully automated method of model checking reaches its limits and can no longer be used due to the state space explosion problem.

As a rule, existing formal approaches to verification are either inapplicable to industrial-strength microprocessor systems or require an enormous amount of manual work.

2. Primary Verification Methods

Formal methods provide a mathematical proof of the correspondence between a model of the object under verification and the object's specification, that is, a set of properties it is supposed to satisfy. A mathematical model of reactive systems – and cache coherence protocols are examples of reactive systems – that allows to systematically represent systems components, their coordination and interaction, is a transition system [1].

The main approaches to formal verification are model checking and deductive verification.

The method of *model checking* [2] systematically explores the finite state space of the protocol under verification by means of specific algorithms. The advantages of model checking are full automation and generation of counterexamples that help us find the sources of bugs. The main disadvantage is the state space explosion problem. Modern cache coherence protocols have too many states for an effective state space inspection to be feasible.

Let us consider verification of safety properties, which are described by linear temporal logic (LTL) formula Gp, where p is an assertion – a formula constructed by applying logical connectives to variables of the model. If the assertion is true in each state of the model, then p is an invariant of the model. According to the method of *deductive verification*, in order to prove Gp, it is necessary to develop an auxiliary assertion φ, which is an over-approximation of the state space, and then show that φ implies p (i.e., that φ is stronger than p). The method is based on the following inference rule [1]:

$$\frac{\begin{array}{l}\text{I1. } \varphi \text{ is true in the initial states of the model} \\ \text{I2. All transitions preserve } \varphi \\ \text{I3. } \varphi \rightarrow p\end{array}}{\text{G}p}$$

An assertion φ is called *inductive* if it satisfies the premises I1 and I2. An inductive assertion is always an over-approximation of the set of reachable states. If p is an

invariant of the system under verification, then there always exists an inductive assertion φ stronger than p [1]. The initial assertion p is rarely inductive. As a rule, the verification engineer must develop an auxiliary assertion and check the validity of the premises I1-I3.

Deductive verification allows us to work with systems with infinite number of states. Theorem provers assist in using formal logic for reasoning about mathematical objects. Popular tools are ACL2, PVS, Isabelle. The underlying logics of theorem provers vary substantially. However, all theorem provers support rich and expressive logics. In general, expressiveness of a logic leads to its undecidability. That means that there is no automatic procedure that, given a formula, can always determine if there exists a derivation of the formula in the logic. The use of theorem proving presumes interaction with an expert user and is a complicated creative process. When the theorem prover cannot find the derivation of a formula given a proof outline, it is very hard to find the actual bug in the system under verification.

Reference [3] describes the experience of using the PVS theorem prover for parameterized verification of the FLASH cache coherence protocol. During the proof construction, authors manually looked for candidates for inductive assertions many times. When they failed to prove their inductiveness, they analyzed the reasons for that and devised additional conditions that transformed the assertion into an inductive one. This process is extremely laborious, which is why methods that are solely based on theorem proving can only find a limited usage in verification of cache coherence protocols.

3. Verification Methods for Scalable Systems

Development of verification methods for scalable systems may be carried on in several directions: 1) improvement of methods based on model checking; 2) improvement of methods based on deductive verification; 3) combination of the methods from the first and the second groups.

Methods of verification of cache coherence protocols deployed in industrial-strength microprocessor systems must satisfy a number of requirements: 1) possibility of conducting verification in a reasonable amount of time; 2) high level of automation; 3) ability to provide information about sources of bugs.

Model checking or deductive verification on their own do not meet these needs. Consequently, building a general infrastructure that would combine and further develop methods of model checking and deductive verification seems to be the most promising approach to verification of scalable systems.

4. Abstraction and Compositional Model Checking

The main approaches allowing the application of model checking to verification of scalable systems are abstract model checking and compositional verification [2]. Abstraction methods diminish the number of states of the model under verification and preserve the properties of interest at the same time.

Equivalence relations, which guarantee that the models will have the same behaviors, usually do not decrease the number of states sufficiently. Instead, simulation relations, which relate models to their abstractions, are used. The simulation guarantees that every behavior of a model is a behavior of its abstraction. However, the abstraction might have behaviors that are not possible in the original system.

Abstract state spaces may be obtained by means of under-approximation methods, which remove behaviors, or over-approximation methods, which add new behaviors. Thus, in case of under-approximation, a bug in the abstract model implies a bug in the concrete model, and in case of over-approximation, correctness of the abstract model implies correctness of the concrete model. Further in this article we only consider over-approximations, also known as conservative abstractions.

Developing abstract models involves finding a compromise between two conflicting goals: 1) generation of small abstract models that can be model checked; 2) generation of precise abstract models.

Usually, the smaller the model, the more behaviors it allows. This may lead to spurious counterexamples that are not present in the concrete model. There are at least two ways out: 1) construction of precise abstract models; 2) analysis of counterexamples and modification of the abstract model according to the acquired information (counterexample-guided abstraction refinement).

Methods that create precise abstract models (for example, based on counter abstraction or environment abstraction [4]) lead to models of big size in case of complicated protocols.

The idea of compositional verification [5] is to exploit the natural decomposition of a distributed system into processes. Processes are verified individually (with a generalized environment), then the results are combined, and a verdict about correctness of the initial model is made. A compositional approach must provably lead to simplified models satisfying the properties of the initial model.

5. A Method of Compositional Model Checking

5.1 General Idea

The method described in this paper adapts the method [6] to work with a subset of Promela. The method is based on a combination of model checking and theorem proving. The choice of Spin is motivated by the fact that Spin is a modern and constantly evolving tool that supports many optimizations and verification modes. The Promela language is convenient for description of distributed systems, including cache coherence protocols. Moreover, Spin may be used as the basis for generators of test programs the purpose of which is verification of implementations of cache coherence protocols [7].

The method shows how to build an abstract model that simulates a given concrete model of a cache coherence protocol. The construction is performed by means of syntactic transformations of the concrete Promela model.

5.2 A Mathematical Model of Cache Coherence Protocols

Cache coherence protocols may be seen as asynchronous systems of communicating processes in which a process is a finite automaton. Then a mathematical model of a cache coherence protocol is a system of communicating finite automata.

A Promela model specifies the behavior of a set of asynchronously executing processes in a distributed system. Each Promela process defines an extended finite automaton. Thus, Promela is suitable for describing models of cache coherence protocols.

By simulating the execution of a Promela model we can build a digraph of all reachable states of the model. Each node in the graph represents a state of the model, and each edge represents a single possible execution step by one of the processes. This graph is always finite [8].

Safety properties can be interpreted as statements about the presence or absence of specific types of nodes in the reachability graph.

Let us consider the transition system corresponding to the reachability graph. The following discussion considers a subset of Promela.

A transition system is a triple $TS = (S, S_0, E)$, where S is a finite non-empty set of states, $S_0 \subseteq S$ is a non-empty set of initial states, $E \subseteq S \times S$ is a transition relation on S such that

$$(\forall s \in S)\ (\exists s' \in S):\ (s, s') \in E$$

In order to be able to formally define syntactic transformations of a Promela model, we will represent models by means of a triple $P = (V, \Theta, R)$, where

- V is a set of variables of the model, each variable is of its own type;
- Θ is the initialization predicate;
- R is the set of transition rules represented as guarded commands consisting of a condition and a set of assignments:

$$cond \rightarrow \{v_1 := t_1; \ldots; v_k := t_k\},$$

where *cond* is the condition (predicate), $v_i \in V$ are model variables, each t_i is a term of the same type as v_i; := denotes assignment.

An interpretation of a set of typed variables V is a mapping that assigns to each variable $v_i \in V$ a value in the domain of v_i.

A triple $P = (V, \Theta, R)$ determines a transition system $TS^P = (S, S_0, E)$ in the following way. Each state $s \in S$ is an interpretation of the set V. For every term t we write $s(t)$ for the value of t in the state s. For a predicate φ, we denote $s \models \varphi$ if and only if $s(\varphi) = true$. A predicate φ is an *invariant* of a model *P*, denoted by $P \models \varphi$, if $\forall s \in S : s \models \varphi$. S_0 is the set of states $s \in S$ such that $s \models \Theta$.

There exists a transition $s \rightarrow s'$, which means $(s, s') \in E$, if there exists a transition rule

$$cond \rightarrow \{v_1 := t_1;\ \ldots\ ; v_k := t_k\},$$

such that $s \models cond$ and s' is a state in which

$$(\forall i \in \{1, \ldots, k\})\ (s'(v_i) = s(t_i))$$

and

$$(\forall v_j \in V \setminus \{v_1, \ldots, v_k\})\ (s'(v_j) = s(v_j)).$$

5.3 The Abstract Model

Let $N = \{p_1, \ldots, p_n\}$ be a parameter set, where $p_1, \ldots, p_n$ are constants of the type used to represent processes in the model and n is a natural number defined by the number of cache agents in the system.

Let $P = (V, \Theta, R)$ be a symmetric model [9] and $M = \{p_1, \ldots, p_m\}$ be a subset of the set $N = \{p_1, \ldots, p_n\}$, $m \le n$. Let *abs* be the element that is an abstraction of elements $p_{m+1}, \ldots, p_n$ and $M_{abs} = M \cup \{abs\}$. We define the abstract model $P_{abs} = (V, \Theta_{abs}, R_{abs})$ with the parameter set M_{abs} as follows.

Let *S* be the set of states of the model *P* and S_{abs} be the set of states of the model P_{abs}.

The predicate Θ_{abs} is obtained by the syntactic transformations $Trans_P$.

The transition rules R_{abs} are obtained by syntactic transformations $Trans_R$ that include transformations of conditions $Trans_P$ and transformations $Trans_A$ of the assignments that appear in the rules:

$$Trans_R(cond \rightarrow \{v_1 := t_1; \ldots; v_k := t_k\}) =$$
$$Trans_P(cond) \rightarrow \{Trans_A(v_1 := t_1); \ldots; Trans_A(v_k := t_k)\}$$

The transformations of terms $Trans_T$ are defined in the following way.

$$Trans_T(v) = v \text{ for each } v \in V ,$$

$$Trans_T(p_i) = \begin{cases} p_i \text{ for } i \le m, \\ abs \text{ for } i > m \end{cases} ,$$

$$Trans_T(c) = c \text{ for all other constants } c .$$

This definition is extended inductively to work with composite term expressions. Suppose $\varphi(t_1, \ldots, t_k)$ is a predicate, i.e., a logical combination of $t_1, \ldots, t_k$. Then $Trans_T(\varphi(t_1, \ldots, t_k))$ is the same logical combination of $Trans_T(t_1), \ldots, Trans_T(t_k)$. Define $Trans_P(\varphi)$ to be the same logical combination of $t'_1, \ldots, t'_k$, where

$$t'_i = \begin{cases} t_i, \text{ if } Trans_T(t_i) = t_i, \\ true, \text{ if } Trans_T(t_i) \ne t_i \text{ and } t_i \text{ occurs positively in } \varphi, \\ false, \text{ if } Trans_T(t_i) \ne t_i \text{ and } t_i \text{ occurs negatively in } \varphi. \end{cases}$$

Now let us define the transformations of assignments $Trans_A$. Denote by $\varnothing$ the absence of assignment and let

$$t' = \begin{cases} t, \text{ if } Trans_T(t) = t, \\ \text{any value in the domain of } t, \text{ otherwise} \end{cases} .$$

Table 1 lists the allowed types of assignments and their corresponding transformations. Define *Array* to be a Promela array and $f_2 : N \to M_{abs}$ to be a mapping that maps $p_1, \ldots, p_m$ to themselves and maps $p_{m+1}, \ldots, p_n$ to *abs*.

The abstract set of transitions is defined as follows:

$$R_{abs} = \{Trans_R(r) \mid r \in R\} .$$

Table 1. Syntactic Transformations of Assignments

Type of assignment	**Assignment transformation**
$v := t$	$v := t'$
$Array[p_i] := t$	$\varnothing$, if $i > m$ $Array[p_i] := t'$, if $i \le m$
$Array[t] := p_i$	$Array[t] := f_2(p_i)$

5.4 Justification of the Abstraction Rules

It can be shown [9] that the abstraction map $\alpha : S \to S_{abs}$ preserves transitions, that is

$$\forall s \in S : (s \to s') \Rightarrow (\alpha(s) \to \alpha(s'))$$

Then, safety properties are preserved: If a state is reachable in the concrete model, it is reachable in the abstract model. In other words, the abstraction map is a simulation relation.

5.5 The Method

The verification method is based on two observations. The first one is the fact that the abstraction map is a simulation relation. The second one is the guard strengthening principle [9] that makes the following strategy correct.

Given a model *P* and a predicate φ, in order to prove that $P \models \varphi$: 1) add φ to the conditions of transition rules of *P* by means of conjunction; 2) prove that φ is an invariant of the newly acquired model.

The method consists of the following steps. Input objects are a symmetric model *P* with parameter set $N = \{p_1, \ldots, p_n\}$ and a safety property φ.

1. Construct P_{abs}, using the syntactic transformations from section 5.3. Let $Q = P_{abs}$.
2. If $Q \models \varphi$, the verification is finished: we conclude that $P \models \varphi$.

Otherwise, examine a counterexample provided by Spin, devise an invariant ψ and modify Q as described in [9]. Set $\varphi = \varphi \wedge \psi$. Go to step 2.

6. Design of a Cache Coherence Protocols Verification System

The syntactic transformations described in section 5.3 can be fully automated. Performing them by hand is tedious and impractical, especially in an industrial setting. Therefore, in order to alleviate this problem, a tool may be developed, which would build an internal representation of the concrete Promela model, modify it according to the transformations, and produce the abstract model. An abstract syntax tree may be the internal representation.

The transformations of Promela models are shown in Fig. 1.

The question of automating the refinement transformations is significantly harder. Further research is needed in this direction.

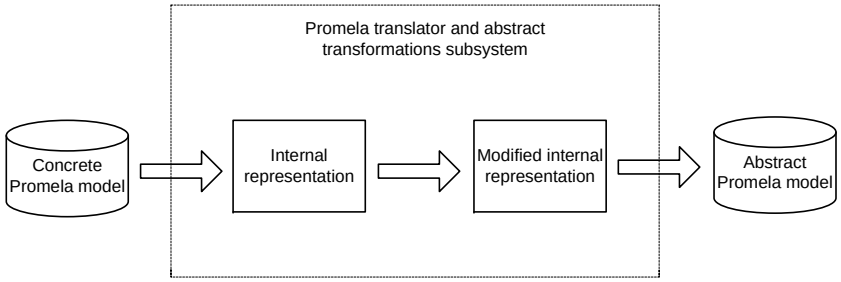

Figure 1. The transformations of Promela models

7. Verification of the German Cache Coherence Protocol

I developed a Promela model of the German protocol. The model is written in the style of [10]. The model implements the algorithm of memory access requests processing shown in Fig. 2.

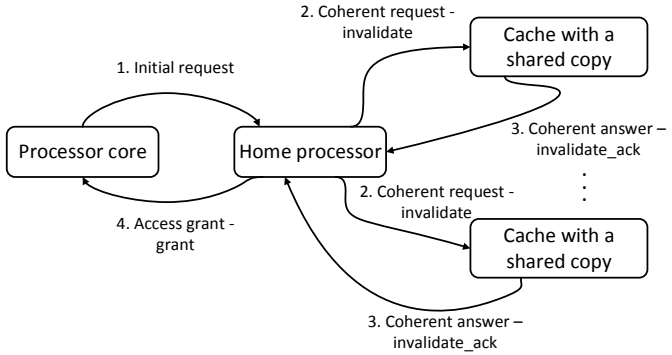

Figure 2. Processing of the read/write requests of the German cache coherence protocols

A processor core and the corresponding cache controller are represented by the Promela process `core` and the home-processor is represented by the process `home`. Thus, the model consists of one process `home` and N processes `core` where N is a natural number. Interaction between the processes is accomplished by means of the three Promela arrays `channel1`, `channel2`, and `channel3` (see Fig. 3).

The array `channel1` is for the initial requests `req_*` sent by a processor to the home processor. The array `channel2` is for the snoop requests `invalidate` sent by the home processor to cache controllers and for grants `grant_*`. The array `channel3` is used for coherence answers sent by cache controllers to the home processor (`invalidate_ack`).

The German protocol uses three main states of a cache line: Invalid, Exclusive, and Shared.

According to the transformations described in section 5.3, I developed the initial version of the abstract model. The abstract model contains one process `home`, two

processes `core`, and one abstract process `home_abs`. One of the most complicated parts of creating the abstract model – the transformation of assignments – is depicted in Table 2. Table 2 shows examples of the corresponding transformations of the German cache coherence protocol Promela model.

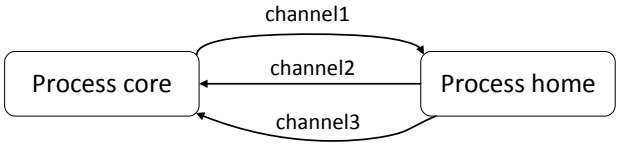

Figure 3. Communication channels between processes in the Promela model of the German cache coherence protocol

Table 2. Examples of the syntactic transformations of the Promela model of the German protocol

Assignment	**Assignment transformation**
`curr_command = req_shared`	`curr_command = req_shared`
`sharer_list[i] = true`	$\varnothing$, if $i > m$ sharer_list[i] = true, if $i \le m$
`curr_client = i`	`curr_client = i` in a concrete process `curr_client = abs` in the abstract process

The verified property stated that it is impossible for a cache line to be in state Exclusive in one cache and in state Shared in some other cache. For example:

```
never { do :: assert( (!(cache[0] == exclusive && cache[1]
== shared)) ) od }
```

This property did not hold on the initial abstract model. According to section 5.5, I performed the refinement process. Two additional invariants were developed and the verification process was finished due to the absence of counterexamples. The refinement process was similar to that described in [6].

For the experimental check of the method's ability to find bugs, I verified two buggy versions of German described in [4]. In the first buggy version, after the home processor grants exclusive access to a cache, it fails to set the `exclusive_granted` variable to true. Thus, when another cache requests shared access, it gets the access even though the first cache holds it in exclusive state. In this case Spin issues a counterexample because the assertion

```
assert( (!(cache[0] == exclusive && cache[1] == shared)) )
```

is violated.

In the second buggy version, the home processor grants a shared request even if `exclusive_granted` variable is true. In this case Spin issued a counterexample because of the violation of one of the invariants found during the abstraction process.

8. Conclusion and Directions for Future Work

Formal methods for verification of cache coherence protocols fall into two groups: methods based on model checking and methods based on deductive verification. Model checking is fully automated but suffers from the state space explosion problem. Deductive verification is scalable but requires a lot of expert's hand work. Combination of the two approaches seems promising because of its potential ability to lead to a scalable method that requires an acceptable amount of hand work.

On the basis of existing literature, a method that is such a combination is described. Although the method can be used for parameterized verification, it has some drawbacks. It supports a very limited subset of Promela constructs and poses unnecessary limitations on the way verification engineers should write their Promela models. The style of the Promela model used in this paper is less intuitive than the style of the model described in [7]. The model from [7] was obtained by a natural decomposition of the Elbrus system-on-chip under verification and organizing process communication through Promela channels. The model was successfully used in verification of several Elbrus systems.

Future work directions include provable extension of the Promela subset that can be dealt with by the verification method, the examination of the impacts of different styles of descriptions of cache coherence protocols, and development of tools that would automate parts of the verification process. The verification process will be applied to Elbrus microprocessors.

References

[1]. Z. Manna, A. Pnueli, "The temporal logic of reactive and concurrent systems: specification," Springer-Verlag, 427 pp., 1992.

[2]. E.M. Clarke, O. Grumberg, D. Peled, "Model checking," MIT Press, 314 pp., 1999.

[3]. S. Park, D. Dill, "Verification of FLASH cache coherence protocol by aggregation of distributed transactions," Proceedings of the 8th annual ACM symposium on parallel algorithms and architectures, pp. 288–296, 1996.

[4]. M. Talupur, "Abstraction Techniques for Parameterized Verification," PhD Thesis, 2006.

[5]. E. Clarke, D. Long, K. McMillan, "Compositional model checking," Proceedings of the fourth IEEE symposium on logic in computer science, 1989.

[6]. C. Chou, P. Mannava, S. Park, "A simple method for parameterized verification of cache coherence protocols," Formal methods in computer-aided design, vol. 3312, pp. 382–398, 2004.

[7]. V. Burenkov, "Generator testov dlya verifikatsii protocola cogerentnosti kesh pamyati [A test generator for cache coherence protocol verification]," Voprosi radioelektroniki, seria EVT, 3, pp. 56–63, 2014.

[8]. G. Holzmann, "The Spin model checker: primer and reference manual," Addison-Wesley Professional, 608 pp., 2003.

[9]. S. Krstic, "Parameterized system verification with guard strengthening and parameter abstraction," Automated verification of infinite state systems, 2005.

[10]. A. Pnueli, S. Ruah, L. Zuck, "Automatic deductive verification with invisible invariants," Tools and algorithms for the construction and analysis of systems, vol. 2031, pp. 82–97, 2001.

О реализации формального метода верификации масштабируемых систем с когерентной памятью

Владимир Буренков <burenkov_v@mcst.ru>,
Московский государственный университет имени Н.Э. Баумана, 105005, Москва, Российская федерация, 2-я Бауманская улица, 5
МЦСТ, 119334, Москва, Российская Федерация, ул. Вавилова, 24

Аннотация. В работе приведен анализ существующих методов верификации протоколов когерентности кэш-памяти масштабируемых систем. На основании литературы изложен формальный метод параметризованной проверки свойств безопасности протоколов когерентности. Предложена архитектура системы верификации протоколов когерентности. Описано применение метода к верификации протокола German, построение и анализ соответствующей Promela-модели. Сформулированы направления по улучшению и автоматизации метода, необходимые для верификации микропроцессоров «Эльбрус».

Keywords: formal verification; model checking; deductive verification; cache coherence protocol; Elbrus

Список литературы

[1]. Z. Manna, A. Pnueli, “The temporal logic of reactive and concurrent systems: specification,” Springer-Verlag, 427 pp., 1992.
[2]. E.M. Clarke, O. Grumberg, D. Peled, “Model checking,” MIT Press, 314 pp., 1999.
[3]. S. Park, D. Dill, “Verification of FLASH cache coherence protocol by aggregation of distributed transactions,” Proceedings of the 8th annual ACM symposium on parallel algorithms and architectures, pp. 288–296, 1996.
[4]. M. Talupur, “Abstraction Techniques for Parameterized Verification,” PhD Thesis, 2006.
[5]. E. Clarke, D. Long, K. McMillan, “Compositional model checking,” Proceedings of the fourth IEEE symposium on logic in computer science, 1989.
[6]. C. Chou, P. Mannava, S. Park, “A simple method for parameterized verification of cache coherence protocols,” Formal methods in computer-aided design, vol. 3312, pp. 382–398, 2004.

[7]. В.С. Буренков. Генератор тестов для верификации протокола когерентности кэш-памяти // Вопросы радиоэлектроники, серия ЭВТ, 2014, выпуск 3, с. 56-63.
[8]. G. Holzmann, “The Spin model checker: primer and reference manual,” Addison-Wesley Professional, 608 pp., 2003.
[9]. S. Krstic, “Parameterized system verification with guard strengthening and parameter abstraction,” Automated verification of infinite state systems, 2005.
[10]. A. Pnueli, S. Ruah, L. Zuck, “Automatic deductive verification with invisible invariants,” Tools and algorithms for the construction and analysis of systems, vol. 2031, pp. 82–97, 2001.