Checking Parameterized PROMELA Models of Cache Coherence Protocols

[1] *V.S. Burenkov <burenkov_v@mcst.ru>*
[2] *A.S. Kamkin <kamkin@ispras.ru>*
[1] *JSC MCST,*
24 Vavilov str., Moscow, 119334, Russian Federation
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25 Alexander Solzhenitsyn str., Moscow, 109004, Russian Federation

Abstract. This paper introduces a method for scalable verification of cache coherence protocols described in the PROMELA language. Scalability means that resources spent on verification (first of all, machine time and memory) do not depend on the number of processors in the system under verification. The method is comprised of three main steps. First, a PROMELA model written for a certain configuration of the system is generalized to the model being parameterized with the number of processors. To do it, some assumptions on the protocol are used as well as simple induction rules. Second, the parameterized model is abstracted from the number of processors. It is done by syntactical transformations of the model assignments, expressions, and communication actions. Finally, the abstract model is verified with the SPIN model checker in a usual way. The method description is accompanied by the proof of its correctness. It is stated that the suggested abstraction is conservative in a sense that every invariant (a property that is true in all reachable states) of the abstract model is an invariant of the original model (invariant properties are the properties of interest during verification of cache coherence protocols). The method has been automated by a tool prototype that, given a PROMELA model, parses the code, builds the abstract syntax tree, transforms it according to the rules, and maps it back to PROMELA. The tool (and the method in general) has been successfully applied to verification of the MOSI protocols implemented in the Elbrus computer systems.

Keywords: multicore microprocessors, shared memory multiprocessors, cache coherence protocols, model checking, SPIN, PROMELA.

DOI: 10.15514/ISPRAS-2016-28(4)-4

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

1. Introduction

Shared memory multiprocessors (*SMP*) constitute one of the most common classes of high-performance computer systems. In particular, it includes multicore

microprocessors, which combine several processors (cores) on a single chip [1]. Nowadays, 8- and 16-core microprocessors are in mass production; hardware vendors have announced forthcoming 48-, 80-, and even 100-core designs. Multicore microprocessors and SMP systems are also designed by Russian companies such as MCST and INEUM, e.g., Elbrus-4C (4 cores, 2014) and Elbrus-8C (8 cores, 2015) [2].

The main problem arising in the development of SMP systems is ensuring *memory coherency*. As each processor contains a local cache, multiple copies of the same data may exist in the system: one copy is in the main memory, and several copies are in the processors' caches. Modification of a copy should cause either the invalidation of the other copies or their consistent modification. This is supported by so-called *cache controllers*, i.e. memory devices connected into a network and cooperating in accordance with a special protocol, so-called *cache coherence protocol* (*CCP*) [3].

Development of cache coherence mechanisms includes two stages: first, design of a CCP; second, its implementation in hardware. The both stages are error-prone; accordingly, methods for protocol verification and methods for hardware verification are in use [4]. Protocol bugs are especially critical and should be revealed before implementing the hardware. The widely recognized method for protocol verification is *model checking* [5]. It is fully automated, but suffers from a principal drawback – it is not scalable due to the *state space explosion* problem. Using the traditional methods for verifying a CCP of a system with four and more processors is impossible (at least, highly problematic) [6].

To overcome the issue and develop scalable verification technologies, researchers utilize *parameterized model checking* [7]. The idea is to construct abstract models that are independent of the number of processors and may be verified with the existing tools. Correctness of the abstract model guarantees correctness of the original one (checking, however, may produce wrong error messages, so-called *false positives*). The proposed approach is also of that type. In contrast to the existing ones, it supports the PROMELA language used in the SPIN model checker [8] and the message passing primitives. The method was successfully used for verifying the CCPs implemented in the Elbrus computer systems [2].

The paper is structured as follows. In Section 2, we analyze the existing approaches to CCP verification. In Section 3, we propose a method for constructing an abstract model out of a PROMELA protocol model. In Section 4, we describe theoretical foundations of the suggested method. In Section 5, we provide a case study on using the method for verifying a MOSI protocol. In Section 6, we summarize our work and outline directions of further research.

2. Related work

As it has been said, classical model checking is inapplicable to CCPs with an arbitrary number of processors. There exists an alternative approach, called *deductive verification*; however, it is hardly automated due to the need of so-called *inductive invariants* [9] and does not provide any diagnostic information if there are errors.

Parameterized model checking seems to be a more promising approach. It is worth mentioning two directions.

First, verification of a parameterized model (in essence, a family of models) can be reduced to the verification of a single model of the family. Corresponding methods are aimed at finding such number N that verification of the model for N components (processors, cache controllers, etc.) is sufficient for proving correctness in general. In [7], such kind of method is presented, and it is reported that $N = 7$ is enough for the protocols having been examined. However, that value is too big to make the method applicable to industrial SMP systems [6].

Second, a model (parameterized model) can be abstracted so as to reduce the state space size (make it independent of the number of components). In [10], a method for abstracting a model from the exact number of *replicated identical components* (e.g., caches in which the cache line is in a given state) is introduced. The technique significantly reduces the state space size; however, the use of a modified version of the Murφ tool complicates its real-life application. A similar idea, called $(0,1,\infty)$-*counter abstraction*, is employed in [11]-[13]. Though the technique seems to be powerful, it often leads to overly detailed abstract models, which makes the approach inapplicable to complex protocols.

In [14], a general method for *compositional verification* is proposed. The idea is to replace a subset of identical components with an abstract one, called *environment*. Such replacement usually leads to false positives, and considerable efforts are required to eliminate them. In [15]-[18], the approach has been adapted to CCPs. The suggested method is based on *syntactical transformations* of Murφ models and *counterexample-guided abstraction refinement* (*CEGAR*). The main drawbacks are as follows:

- Murφ does not support the message passing primitives, which complicates CCP description;
- restrictions on Murφ models of CCPs are not clearly defined;
- the tools are not in open access.

3. Suggested method

The problem to be solved is as follows. Given a PROMELA model of a CCP for some configuration of an SMP system (i.e. a model with a fixed number $n > 2$ of processors), it is required to check the CCP correctness for an arbitrary configuration of the system (i.e. for any $N \geq n$).

Models considered in this paper satisfy the following conditions (obtained from the verification practice and shown to be sufficient for specifying CCPs). The allowed statements are **if**, **do**, **goto**, = (*assignment*), ! (*send*), and ? (*receive*). Each guarded action is placed in an **atomic** block and therefore is executed with no interruption; **else** alternatives are absent. Assignments' right-hand sides contain only primary expressions, i.e. variables and constants; left-hand sides are variables and array elements (an array index is a primary expression). Atomic logic formulae are of the

form $x == c$ or $B(ch)$, where x is a variable (or an array element), c is a constant, ch is a channel, and B is a predicate: **empty**, **full**, etc.

3.1 Model parameterization

From the conceptual point of view, a CCP model consists of an unbounded number of replicated identical processes, so-called *basic processes*, and a fixed number of *auxiliary processes*. Without loss of generality we will assume that there is only one auxiliary process. All processes are enumerated from 0 to N, where N is a parameter: 0 is the identifier of the auxiliary process, while $1, \dots, N$ are the identifiers of the basic processes. All arrays used in the model (arrays of variables and arrays of channels) are of length N and indexed with the identifiers of the basic processes.

To generalize the original model to a parameterized one, the following induction rules are used:

- each condition containing an array is either a conjunction or a disjunction of similar conditions on all array elements:
  - $\varphi\{i/1\} \wedge \dots \wedge \varphi\{i/n\}$ is interpreted as $\forall i \in \{1, \dots, N\}: \varphi$;
  - $\varphi\{i/1\} \vee \dots \vee \varphi\{i/n\}$ is interpreted as $\exists i \in \{1, \dots, N\}: \varphi$;
- each sequence of statements $\alpha\{i/1\}; \dots; \alpha\{i/n\}$ is interpreted as a loop **for** $(i: 1 \,..\, N)$ $\{\alpha\}$.

Here, φ (α) is a formula (statement) containing an index i as a free variable, and $\varphi\{i/t\}$ ($\alpha\{i/t\}$) denotes the result of substitution of t for all occurrences of i in φ (α).

3.2 Assumptions

Let us consider a CCP where request processing is coordinated by a *system commutator* of the *home processor* (the processor that owns the requested data). Accordingly, the PROMELA model contains two process types: $proc$ is a cache controller (a basic process), and $home$ is a home processor's commutator (an auxiliary process). As usual, the CCP model deals with a single cache line.

Broadly speaking, the CCP works as follows. Each $proc$ instance may initiate an operation on the cache line by sending a primary request to the $home$ process. Upon its reception and analysis, $home$ sends *snoop requests* to all processes except for the sender. After snoop reception, a $proc$ sends a response to the sender (data or an acknowledgement that it has completed an action on the cache line). Having collected all of the answers, the sender informs $home$ on the completion of the operation. As soon as the completion message is received, $home$ can accept the next primary request (see Fig. 1).

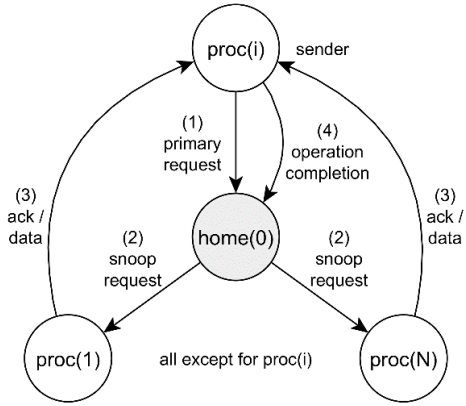

Fig. 1. Generalized scheme of a CCP

It is worth emphasizing that at most one primary request is being processed at each moment of time. It is assumed that values of global variables (e.g., a current sender identifier) are set by $home$ upon reception of a primary request and do not change during its processing.

Each channel can be read by a single process; however, multiple processes are allowed to write into it. A channel is called *simple* if there is only one sender; otherwise, it is called *multiplexed*. Let $C_{S\to r}$ be the set of channels with the reader r and senders from the set S. Channels are divided into three groups (hereinafter, singletons are written without brackets, e.g., $0 \to j$ stands for $\{0\} \to j$):

- $C_* = \bigcup_{j=0}^{N} C_{\{1,\dots,N\}\to j}$ is the set of multiplexed channels of capacity N used by $home$ and $proc$ to receive messages from the basic processes (e.g., a channel over which $home$ receives primary requests, and channels over which processes receive responses);
- $C_{h\to p} = \bigcup_{j=1}^{N} C_{0\to j}$ is the set of simple channels of positive capacity (which is defined by the CCP, but independent of N) used by the basic processes to receive messages from $home$ (e.g., channels over which $home$ transmits snoop requests);
- $C_{p\to h} = \bigcup_{i=1}^{N} C_{i\to 0}$ is the set of simple channels of capacity 1 used by $home$ to receive messages from the basic processes (e.g., channels over which a sender informs $home$ on operation completion).

Messages transmitted via channels are ordered pairs of the form (opc, i), where opc is an operation code, and i is an identifier of the message sender.

A verified CCP property looks as follows:

$$\mathbf{G}\{\forall k, l \in \{1, \dots N\}: (k \neq l) \to \varphi\{i/k, j/l\}\},$$

where **G** is an operator that requires its argument to be true in all reachable states of the model [5]; φ is a formula with two free indices (i and j) that characterizes cache coherency in the corresponding caches. For MOSI protocols [3], φ is as follows:

$$\begin{cases}\neg(cache[i] = M \wedge cache[j] \neq I);\\ \neg(cache[i] = O \wedge cache[j] = O);\end{cases}$$

where $cache$ is an array that stores the cache line states.

3.3 Informal description

The core of the proposed method is syntactical transformation of PROMELA code. The transformations change the process types and retain four processes of $N + 1$: a modified $home$ process ($home_{abs}$), two modified $proc$ processes ($proc_{abs}$), and an environment process representing the rest of the processes ($proc_{env}$). Accordingly, the initialization process of the abstract model is as follows (ABS is a constant distinct from 0, 1, and 2):

```
init {
 atomic {
  run home_abs(0);
  run proc_abs(1);
  run proc_abs(2);
  run proc_env(ABS);
 }
}
```

The length of all arrays is changed from N to 2 (recall that arrays are indexed with the identifiers of the $proc$ processes). Each array access is supplied with the guard $i \leq 2$, where i is the index of the element being accessed.

- On read (in a condition), the atomic formula containing the array access, is replaced with $undef$ (an undefined value) if the index is rejected by the guard:

  $$B(x[i], \dots) \Longrightarrow (i \leq 2 \rightarrow B(x[i], \dots) : undef).$$

  In PROMELA, a formula of the kind $(B \rightarrow t_1 : t_2)$ corresponds to the conditional construct **if** B **then** t_1 **else** t_2 **fi**.

- On write (in an assignment), the assignment to the array is placed inside the selection statement:

  $$x[i] = t \Longrightarrow \textbf{if} :: \textbf{atomic}\,\{i \leq 2 \rightarrow x[i] = t\} :: \textbf{else fi}.$$

  Assignments to the global variables as well as conditions on the global variables remain unchanged.

Channels of the set $C_{h \rightarrow p}$ are represented as an array (let us denote it as ch). Similarly to other arrays, it is truncated to length 2. Each atomic formula over $ch[i]$, where $i > 2$, is replaced with $undef$, while each operation on such a channel is removed. Channels of the sets C_* and $C_{p \rightarrow h}$ are represented by individual variables, not arrays.

Send statements are either unchanged or removed. A statement $ch!m$ in a process type P is removed only in the following cases:

- $ch \in C_{h\to e}$ and $P = home_{abs}$, where $C_{h\to e} = \bigcup_{j=3}^{N} C_{0\to j}$;
  e.g., $home_{abs}$ does not send snoop requests to $proc_{env}$;
- $ch \in C_*$ и $P = proc_{env}$;
  e.g., $proc_{env}$ does not send primary requests / snoop responses.

Receive statements may be left unchanged, modified, or removed. A statement $ch?m$ in a process type P is removed only in the following case:

- $ch \in C_{h\to e}$ and $P = proc_{env}$;
  e.g., $proc_{env}$ does not receive snoop requests.

Modification of $ch?m$ takes place solely in the following case:

- $ch \in C_*$ and $P \in \{home_{abs}, proc_{abs}\}$.

The corresponding transformation replaces a guarded action of the kind **atomic** $\{B \to ch?m\}$ with the following selection statement:

if
:: **atomic** $\{B' \to ch?m\}$
:: **atomic** $\{m.opc = opc_1;\ m.i = ABS\}$
...
:: **atomic** $\{m.opc = opc_k;\ m.i = ABS\}$
fi

where B' is the result of B transformation, and $opc_1, \ldots, opc_k$ are all possible operation codes that may be sent along the channel ch.

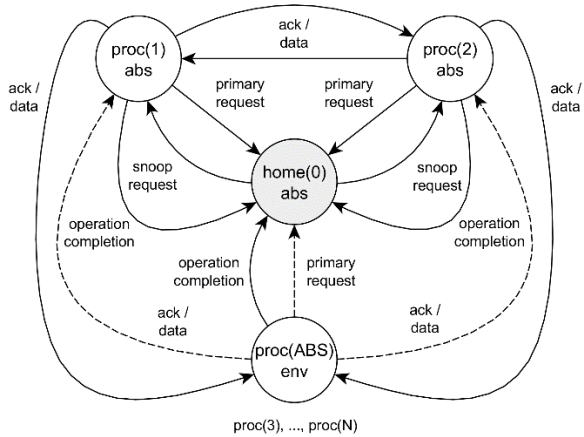

Fig. 2. Abstraction of a CCP model

Fig. 2 provides a simplified view on CCP model abstraction. All processes except for $home(0)$, $proc(1)$, and $proc(2)$ are merged into the environment process

$proc_{env}(ABS)$. Solid arrows represent the unmodified send / receive statements. Dashed arrows correspond to the removed sends / modified receives.

Having performed the above transformations, all logical formulae containing $undef$ (in essence, formulae of Kleene's strong three-valued logic) are transformed into classic logic formulae such that $undef$ in the outer scope is interpreted as $true$. This is achieved by the obvious transformation F:

- $F(\varphi) \Longrightarrow G(\varphi, true)$;
- $G(undef, T) \Longrightarrow T$;
- $G(B, T) \Longrightarrow B$, where B is an atom distinct from $undef$;
- $G(\neg\varphi, T) \Longrightarrow \neg G(\varphi, \neg T)$;
- $G(\varphi \circ \psi, T) \Longrightarrow G(\varphi, T) \circ G(\psi, T)$, where $\circ \in \{\wedge, \vee\}$.

When transforming the PROMELA model, the following optimizations are applied:

- constant propagation and folding;
- dead code elimination.

Here are some simple examples:

- $(i \leq 2) \Longrightarrow true$ in $home_{abs}$ and $proc_{abs}$;
- $(true \wedge B) \Longrightarrow B$ and $(false \wedge B) \Longrightarrow false$;
- **atomic** $\{true \rightarrow \alpha\} \Longrightarrow \alpha$.

It should be said that in general case the abstraction procedure transforms $N + 1$ processes to the $k + 2$ ones, where $k \in \{2, \ldots N - 1\}$: $proc_{abs}$ (in the number k), $home_{abs}$, and $proc_{env}$.

4. Theoretical foundations

4.1 Basic definitions

Let Var be a *set of variables* and $Chan$ be a *set of channels*. $Data = Var \cup Chan$ is referred to as the *set of data*. For each $c \in Chan$, a value $|c| > 0$, called *capacity*, is defined. A *data state* (or *state* for short) is a *valuation* of data, i.e. a mapping s that maps each variable v to the *value* $s(v) \in \mathbb{N}$ and each channel c to the sequence of *messages* $s(c) \in \mathbb{M}^*$ such that $|s(c)| \leq |c|$. The set of all states is denoted by S. A designated state $s_0 \in S$ is called *initial*.

Let us assume that there is a language over the data that includes logic formulae and statements, such as $x = t$ (*assignment*), $c\,!\,m$ (*send*), and $c\,?\,m$ (*read*).

A *guard* is a formula; an *action* is a sequence of statements; a *guarded action* is a pair $\gamma \rightarrow \alpha$, where γ is a guard, and α is an action. The guarded action $true \rightarrow \epsilon$, where ϵ is the empty sequence of statements, is called *empty* and designated as ε. The set of all guarded actions is denoted by Act. A guarded action $\gamma \rightarrow \alpha$ is called *executable* in $s \in S$ iff (if and only if) $s \vDash \gamma$.

A *process graph* (or *process* for short) is a triple $\langle V, v_0, E \rangle$, where V is a *set of vertices*, $v_0 \in V$ is an *initial vertex*, and $E \subseteq V \times Act \times V$ is a *set of edges*.

Process structure is defined by the control statements: **if** (*selection*), **do** (*repetition*), and **goto** (*jump*). Correspondence between code and processes is straightforward and not described here.

A *system* is a set of processes, i.e. $\{\langle V_i, v_{0_i}, E_i\rangle\}_{i=0}^{N}$. Hereinafter, P_i is considered to be a shortcut for $\langle V_i, v_{0_i}, E_i\rangle$. A *configuration* of $\{P_i\}_{i=0}^{N}$ is a pair $\langle l, s\rangle$, where $l: \{0, \dots, N\} \to \bigcup_{i=0}^{N} V_i$ such that $l(i) \in V_i$ for all $i \in \{0, \dots, N\}$, so-called the *control state*, and $s \in S$. The configuration $\langle l_0, s_0\rangle$, where $l_0(i) = v_{0_i}$ for all $i \in \{0, \dots, N\}$, is called *initial*.

The *state space* of a system $\{P_i\}_{i=0}^{N}$ is a triple $\langle C, c_0, T\rangle$, where C is the set of all configurations of the system, c_0 is the initial configuration, and $T \subseteq C \times \left(\{0, \dots, N\} \times (\bigcup_{i=0}^{N} E_i)\right) \times C$ is a *transition relation* such that the following property holds: $\left(\langle l, s\rangle, \left(i, (v, \gamma \to \alpha, v')\right), \langle l', s'\rangle\right) \in T$ iff:

- $l(i) = v$;
- $(v, \gamma \to \alpha, v') \in E_i$;
- $s \vDash \gamma$;
- $l' = (l \setminus \{i \mapsto v\}) \cup \{i \mapsto v'\}$;
- $s' = [\![\alpha]\!](s)$, where $[\![\alpha]\!]: S \to S$ is the semantics of α (actions are assumed to be deterministic).

It is worth mentioning that the restrictions on the transition relation conform to the notion of *asynchronous parallelism*.

A configuration c is called *reachable* in a state space $\langle C, T, c_0\rangle$ iff there is a path in T from c_0 to c. A state s is called *reachable* iff a configuration $\langle l, s\rangle$, for some l, is reachable.

4.2 System abstraction

A *process transformation* (or *transformation* for short) is a function that maps one process to another.

Let $Data_S = (Var_S \cup Chan_S) \subseteq Data$ be a *set of significant data*. States s and s' are called *equivalent* (it is designated as $s \sim s'$) iff $s_{|Data_S} = s'_{|Data_S}$.

A guarded action $\gamma' \to \alpha'$ is referred to as an *abstraction* of a guarded action $\gamma \to \alpha$ in $s \in S$ iff:

- the truth of γ' is determined only by the significant data: for all $s' \in S$ such that $s' \sim s$, $s' \vDash \gamma'$ iff $s \vDash \gamma'$;
- the effect of α' is determined only by the significant data: for all $s' \in S$ such that $s' \sim s$, there holds $[\![\alpha']\!](s') \sim [\![\alpha']\!](s)$;
- γ' is weaker than γ: $s \vDash \gamma \to \gamma'$;
- α' acts similar to α: $[\![\alpha']\!](s) \sim [\![\alpha]\!](s)$.

A set of guarded actions $\{\gamma_i' \to \alpha_i'\}_{i=1}^m$ is referred to as an *abstraction* of a guarded action $\gamma \to \alpha$ in $s \in S$ iff there exists $i \in \{1, \dots, m\}$ such that $\gamma_i' \to \alpha_i'$ is an abstraction of $\gamma \to \alpha$ in s.

A guarded action $\gamma' \to \alpha'$ (a set $\{\gamma_i' \to \alpha_i'\}_{i=1}^m$) is referred to as an *abstraction* of $\gamma \to \alpha$ iff $\gamma' \to \alpha'$ ($\{\gamma_i' \to \alpha_i'\}_{i=1}^m$) is an abstraction of $\gamma \to \alpha$ in all states.

An *abstraction function* is a mapping $f: Act \to 2^{Act}$ such that for all $\gamma \to \alpha \in Act$, $f(\gamma \to \alpha)$ is an abstraction of $\gamma \to \alpha$. The abstraction function $I(\gamma \to \alpha) \equiv \{\gamma \to \alpha\}$ is called *trivial*.

It should be emphasized that this view to abstraction is a bit simplified. An abstraction function should take into account context of a guarded action (the process edge, the process, and the model). Thus, it is assumed that each guarded action contains the context information.

Let $P = \langle V, v_0, E \rangle$ be a process, f be an abstraction function, V' be some set, and $R: V \to V'$ be a mapping. An *abstraction* of P induced by f and R is the process $f(P, R) = \langle V', R(v_0), E' \rangle$, where E' is defined as follows:

- if $(v, \gamma \to \alpha, u) \in E$ and $f(\gamma \to \alpha) = \{\gamma_i' \to \alpha_i'\}_{i=1}^m$, then $\{(R(v), \gamma_i' \to \alpha_i', R(u))\}_{i=1}^m \subseteq E'$;
- no other edges belong to E'.

An abstraction $f(P, R)$, where R is a bijection, is referred to as a *bijective abstraction*.

Besides transforming individual processes, there are of interest transformations that merges several processes into one. Let us consider a particular kind of such transformations, where processes to be merged are identical.

Given a system $\{P_i\}_{i=0}^N$, the following denotations can be introduced ($i \in \{0, \dots, N\}$):

- Use_i is the set of variables read by P_i;
- Def_i is the set of variables assigned by P_i;
- $Var_i = Use_i \cup Def_i$ is the set of variables of P_i;
- Var_{L_i} is the set of *local variables* of P_i (we do not define the set Var_{L_i} assuming that it is provided);
- $Var_G = Var \setminus (\cup_{i=0}^N Var_{L_i})$ is the set of *global variables*.

Similarly, the following sets of channels (including the *sets of local channels* and the *set of global channels*) can be defined: In_i, Out_i, $Chan_i$, $Chan_{L_i}$, and $Chan_G$. In addition,

- $Data_i = Var_i \cup Chan_i$ is the set of data of P_i;
- $Data_{L_i} = Var_{L_i} \cup Chan_{L_i}$ is the set of *local data* of P_i;
- $Data_G = Var_G \cup Chan_G$ is the set of *global data*.

Processes are called *identical* if they can be transformed one another by renaming their local data. More formally, processes P_i and P_j are called identical if there are a bijection $R: V_i \to V_j$ and a bijection $r: Data_{L_i} \to Data_{L_j}$ such that $R(v_{0_i}) = v_{0_j}$ and

$(v, \gamma \to \alpha, u) \in E_i$ iff $\big(R(v), r(\gamma \to \alpha), R(u)\big) \in E_j$, where $r(\gamma \to \alpha)$ is the result of renaming the local data in $\gamma \to \alpha$ in accordance with r.

Let $\{P_i\}_{i=k_1}^{k_2}$ be a system of identical processes, $Data_S \cap \big(\bigcup_{i=k_1}^{k_2} Data_{L_i}\big) = \emptyset$ (the processes' local data are insignificant), g be an abstraction function, V' be some set, and $R: V_{k_1} \to V'$ be a mapping. The process $g\big(P_{k_1}, \dots, P_{k_2}; R\big) = g\big(P_{k_1}, R\big)$ is called a *unifying abstraction* of $\{P_i\}_{i=1}^{k}$ induced by g and R.

The definition needs to be clarified. Provided that the processes $\{P_i\}_{i=k_1}^{k_2}$ operate simultaneously, there are control states that cannot be represented by a single vertex of the abstraction $g\big(P_{k_1}, \dots, P_{k_2}; R\big)$. Thus, a unifying abstraction may appear to be inadequate. Let us assume that each process can be either *active* or *passive*, and it is prohibited two or more processes to be active simultaneously. Besides, the passive mode is organized as the following loop:

- a request is received;
- the local data are updated;
- a response is sent;
- the control is returned to the initial vertex.

Let $V(E')$ be the set of all vertices of the edges from E'.

A process $P = \langle V, v_0, E_A \cup E_P \rangle$ is referred to as a *bimodal process* with the set of *active edges* E_A and the set of *passive edges* E_P iff $E_A \cap E_P = \emptyset$ and the graph $\langle V(E_P), E_P \rangle$ is strongly connected.

Given a bimodal process $P = \langle V, v_0, E_A \cup E_P \rangle$, the following denotation can be introduced: $V_A = V(E_A)$ and $V_P = V(E_P)$ (generally speaking, $V_A \cap V_P \neq \emptyset$).

The process $g(P, R) = \langle V', v_0', E' \rangle$, where g is an abstraction function, and $R: V \to V'$ is a mapping, is called a *serializing abstraction* of P iff R satisfies the following properties:

- $R(v) = v_0'$ for all $v \in V_P \setminus V_A$;
- $R: V_A \to V'$ is a bijection;

and E' is defined as follows:

- if $(v, \gamma \to \alpha, u) \in E_A$ and $g(\gamma \to \alpha) = \{\gamma_i' \to \alpha_i'\}_{i=1}^{m}$, then $\{(R(v), \gamma_i' \to \alpha_i', R(u))\}_{i=1}^{m} \subseteq E'$;
- $(v_0', \varepsilon, v_0') \in E'$ (so-called *ε-self loop*);
- no other edges belong to E';

and for every $(v, \gamma \to \alpha, u) \in E_P$, the empty guarded action ε is an abstraction of $\gamma \to \alpha$, i.e. α depends on and affects solely insignificant data.

The nature of serializing abstraction is removing all passive edges and replacing them with the ε-self loop $(v_0', \varepsilon, v_0')$. Being applied to identical bimodal processes, such abstraction makes them unimodal and serializable (at most one process is operating, i.e. being in a non-initial state, at each moment of time) and allows constructing an adequate unifying abstraction.

Let $M = \{P_i\}_{i=0}^{N}$ be a system where all processes, except maybe $\{P_i\}_{i=0}^{k}$, for some $k \in \{0, \dots, N\}$, are identical and bimodal; $Data_S$ be significant data; V_i', where $i \in \{0, \dots, k+1\}$, be some sets; $R_i: V_i \to V_i'$ be some mappings; f_i, where $i \in \{0, \dots, k\}$, and g be abstraction functions; at that, $f_i(P_i, R_i)$ are bijective abstractions, while $g(P_{k+1}, \dots, P_N; R_{k+1})$ is a serializing abstraction. Then, the system

$$M' = \{f_i(P_i, R_i)\}_{i=0}^{k} \cup \{g(P_{k+1}, \dots, P_N; R_{k+1})\}$$

is called an *abstraction* of M. A process $f_i(P_i; R_i)$, where $i \in \{0, \dots, k\}$, is called an *abstraction of the process* P_i. The process $g(P_{k+1}, \dots, P_N; R_{k+1})$ is called an *abstraction of the environment*.

Statement. Let $M = \{P_i\}_{i=0}^{N}$ and $M' = \{P_i'\}_{i=0}^{k+1}$ be, respectively, a system and its abstraction. Given an arbitrary state s, if s is reachable in the state space of M, then there is a state s' reachable in the state space of M' such that $s' \sim s$.

Proof. Let $ABS = k + 1$. This denotation is introduced to emphasize that the abstraction of the environment, the process $P_{ABS}' = P_{k+1}'$, generalizes not only the process P_{k+1}, but also the processes $P_{k+2}, \dots, P_N$.

A configuration $\langle l', s' \rangle$ of M' is said to *conform* to a configuration $\langle l, s \rangle$ of M iff the following conditions are satisfied:

- $l'(i) = R_i\big(l(i)\big)$ for all $i \in \{0, \dots, k\}$;
- if $l'(ABS) = R_{ABS}(v_{0_{ABS}})$, then $l(i) = v_{0_i}$ for all $i \in \{k+1, \dots, N\}$;
- if $l'(ABS) \neq R_{ABS}(v_{0_{ABS}})$, then there is only one index $i \in \{k+1, \dots, N\}$ such that $l'(ABS) = R_i\big(l(i)\big)$;
- $s' \sim s$.

Let us consider a path in the state space of M starting with $\langle l_0, s_0 \rangle$:

$$\pi = \left\{\Big(\langle l_j, s_j \rangle, \Big(i_j, \big(v_j, \gamma_j \to \alpha_j, v_{j+1}\big)\Big), \langle l_{j+1}, s_{j+1} \rangle\Big)\right\}_{j=0}^{m-1}.$$

Here, $i_j \in \{0, \dots, N\}$ is a process index; $v_j = l_j\big(i_j\big) \in V_{i_j}$ and $v_{j+1} = l_{j+1}\big(i_j\big) \in V_{i_j}$ are the process's vertices connected with the edge labelled by $\gamma_j \to \alpha_j$; $s_j \vDash \gamma_j$ and $s_{j+1} = [\![\alpha_j]\!](s_j)$ for all $j \in \{0, \dots, m-1\}$.

Our goal is to show that, in the state space of M', there is a path π' of the same length as π such that each configuration of π' conforms to the corresponding configuration of π:

$$\pi' = \left\{\Big(\langle l_j', s_j' \rangle, (i_j', (v_j', \gamma_j' \to \alpha_j', v_{j+1}')), \langle l_{j+1}', s_{j+1}' \rangle\Big)\right\}_{j=0}^{m-1}.$$

Obviously, existence of such a path implies that there is a state s_m' reachable in the state space of M' such that $s_m' \sim s_m$. Let us consider how to construct π'.

Induction basis. The initial configuration $\langle l_0', s_0' \rangle$ certainly conforms to $\langle l_0, s_0 \rangle$: $v_{0_i}' = l'(i) = R_i\big(l(i)\big) = R_i(v_{0_i})$ for all $i \in \{0, \dots, N\}$.

Inductive step. Given an arbitrary index $q \in \{0, \dots, m-1\}$, we will show that if the configuration $\langle l'_q, s'_q \rangle$ conforms to $\langle l_q, s_q \rangle$, then there are a process of M' (let us denote its index as i'_q) and an edge $\left(v'_q, \gamma'_q \to \alpha'_q, v'_{q+1}\right)$ of that process such that $\langle l'_{q+1}, s'_{q+1} \rangle = \langle \left(l'_q \setminus \{i'_q \mapsto v'_q\}\right) \cup \left\{i'_q \mapsto v'_{q+1}\right\}, [\![\alpha']\!](s'_q) \rangle$ (see the definition of the state space) conforms to $\langle l_{q+1}, s_{q+1} \rangle$. There are two cases:

- $i_q \in \{0, \dots, k\}$;
- $i_q \in \{k+1, \dots, N\}$.

Case 1. If $i_q \in \{0, \dots, k\}$, let $i'_q = i_q$: the transition is executed by the process $P'_{i_q} = f_{i_q}(P_{i_q}, R_{i_q})$.

The edge $\left(v_q, \gamma_q \to \alpha_q, v_{q+1}\right)$ of the process P_{i_q} is abstracted to the set of edges $\left\{\left(R_{i_q}(v_q), \gamma_q^{(i)} \to \alpha_q^{(i)}, R_{i_q}(v_{q+1})\right)\right\}_{i=1}^{t}$, where $f_{i_q}\left(\gamma_q \to \alpha_q\right) = \left\{\gamma_q^{(i)} \to \alpha_q^{(i)}\right\}_{i=1}^{t}$. Among them, there is selected an edge whose label, $\gamma'_q \to \alpha'_q$, is an abstraction of $\gamma_q \to \alpha_q$ in s_q. Such an edge always exists (see the definition of the process abstraction). We need to proof that the chosen edge belongs to the state space of M' and the configuration $\langle l'_{q+1}, s'_{q+1} \rangle$ conforms to $\langle l_{q+1}, s_{q+1} \rangle$. It is sufficient to proof the following statements:

- $s'_q \vDash \gamma'_q$;
- $[\![\alpha']\!](s'_q) \sim [\![\alpha]\!](s_q)$.

The first of them can be deduced from the facts that $s_q \vDash \gamma_q$ (otherwise, the state space of M would not include the transition under consideration), $\gamma'_q \to \alpha'_q$ is an abstraction of $\gamma_q \to \alpha_q$ in s_q, and $s'_q \sim s_q$ (the induction assumption). Obviously, $s_q \vDash \gamma_q$ and $s_q \vDash \gamma_q \to \gamma'_q$ lead to $s_q \vDash \gamma'_q$, which, in couple with $s'_q \sim s_q$, leads to $s'_q \vDash \gamma'_q$. The second statement is an implication of the facts that $\gamma'_q \to \alpha'_q$ is an abstraction of $\gamma_q \to \alpha_q$ in s_q and $s'_q \sim s_q$.

Case 2. If $i_q \in \{k+1, \dots, N\}$, let $i'_q = ABS$: the transition is executed by the process $P'_{ABS} = g(P_{k+1}, \dots, P_N; R_{ABS})$. There are two subcases:

- the edge $\left(v_q, \gamma_q \to \alpha_q, v_{q+1}\right)$ is active;
- the edge $\left(v_q, \gamma_q \to \alpha_q, v_{q+1}\right)$ is passive.

Subcase 2.1. If the edge is active, then, by definition of configuration conformance, $l'(ABS) = R_{i_q}(v_q)$. In P'_{ABS}, there is selected an edge between $R_{i_q}(v_q)$ and $R_{i_q}(v_{q+1})$ whose label is an abstraction of $\gamma_q \to \alpha_q$ in s_q. Such an edge always exists (active edges are abstracted in a usual way). The further proof is similar to that in Case 1.

Subcase 2.2. If the edge is passive, then $R_{i_q}(v_q) = R_{i_q}(v_{q+1}) = R_{i_q}\left(v_{0_{i_q}}\right) = v'_{0_{ABS}}$. In P'_{ABS}, there is selected an edge $(v'_{0_{ABS}}, \varepsilon, v'_{0_{ABS}})$. Conformance of the configuration follows from the facts that passive edges do not depend on sufficient data and do not affect them.

Conclusion. Given an arbitrary path π in the state space of M, there is a path π' in the state space of M' such that the ending state of π' is equivalent to the ending state of π.

Q.E.D.

Corollary. Let $M = \{P_i\}_{i=0}^{N}$ and $M' = \{P_i'\}_{i=0}^{k+1}$ be, respectively, a system and its abstraction. Given an arbitrary formula φ over significant data, if φ is true (false) in all states reachable in the state space of M', then φ is true (false) in all states reachable in the state space of M.

4.3 Model transformation

This section defines abstraction functions used for protocol model transformation. The description is not quite formal: rigorous definition requires, first, formalization of the PROMELA semantics and, seconds, usage of formalisms for describing code transformations. Nevertheless, we believe that the explanations below are sufficient for formalizing and automating the abstraction procedure.

Let $M = \{P_i\}_{i=0}^{N}$ and $M' = \{P_i'\}_{i=0}^{k+1}$ be, respectively, a system (referred to as an *original model*) and its abstraction (referred to as an *abstract model*).

Let us recall that each message circulating in the model includes the sender's identifier. A state of a channel being written by $\{P_i\}_{i=k+1}^{N}$, as well as messages being read from the channel may contain identifiers from the set $\{k+1, \dots, N\}$. In the abstract model, there are no such identifiers: they are mapped to ABS (usually, $ABS = k+1$). The definition of state equivalence should be modified so as not to distinguish between i and ABS if $i \in \{k+1, \dots, N\}$.

Another issue is as follows. State of a channel's buffer is not of importance until a message is read. The idea is to ignore some messages (in particular, messages written by $\{P_i\}_{i=k+1}^{N}$). In this case, a send statement can be replaced with ε. To preserve the abstraction properties, each read from the channel should be supplied (as alternative behavior) with the assignments of all possible values that could be sent via the channel by the removed statement to the message variable.

To be more precise, the definition of state equivalence should take into account the following considerations:

- given a channel $c \in C_*$, an abstract state s' is (quasi) equivalent to a state s (state is a sequence of messages) iff s' is produced from s by removing all messages with identifiers from $\{k+1, \dots, N\}$;
- the channels from $C_{h\to e} = \bigcup_{j=k+1}^{N} C_{0\to j}$ are insignificant (every two states of a channel are equivalent);
- an abstract state s' of the channels $C_{e\to h} = \bigcup_{i=k+1}^{N} C_{i\to 0}$ (as a whole) is equivalent to a state s iff there is $i \in \{k+1, \dots, N\}$ such that for each $c \in C_{i\to 0}$, the state $s'(c')$, where c' is a channel that corresponds to c in P_{ABS}, is produced from $s(c)$ by replacing i with ABS while the remaining channels

are empty in both states.

The suggested approach implies the following restrictions on the input model:

- $Data_S = Data \setminus \left(\bigcup_{i=k+1}^{N} Data_{L_i}\right)$;
- for each $i \in \{1, \dots, N\}$, there holds $Chan_i = Chan_{A_i} \cup Chan_{P_i}$, where $Chan_{A_i}$ and $Chan_{P_i}$ are the sets of channels used, respectively, in the active and passive modes, and:
  - $Chan_{A_i} \cap Chan_{P_i} = \emptyset$;
  - $Chan_{A_i} \subseteq Chan_G$ $(Chan_{A_i} = C_{\{1,\dots,N\}\to 0} \cup C_{i\to 0})$;
  - $Chan_{P_i} \subseteq Chan_{L_i}$ $(Chan_{P_i} = C_{0\to i} \cup \left(\bigcup_{j=1}^{N} C_{\{1,\dots,N\}\to j}\right))$;
- the only channel predicate in use is **empty** (behavior does not depend on the number of messages in the channels' buffers);
- there are no dependencies via variables between the processes $\{P_i\}_{i=1}^{N}$ (all dependencies are via messages);
- each guarded action is closed under data dependencies via variables;
- there are no data dependencies from the local data (control dependencies from the local data are allowed).

$M' = \{P_i'\}_{i=0}^{k+1} = \{f_i(P_i, R_i)\}_{i=0}^{k} \cup \{g(P_{k+1}, \dots, P_N; R_{k+1})\}$, the abstract model, is constructed as follows (the description below can be viewed as a definition of the mappings R_i and the abstraction functions f_i and g). Initially, each process P_i', where $i \in \{0, \dots, k+1\}$, is isomorphic to P_i: $P_i' = I\left(P_i, R_{0_i}\right)$, where I is the trivial abstraction function, while $R_{0_i}: V_i \to V_i'$ is a bijection. Then, the following transformations are applied to $P'_{ABS} = P'_{k+1}$ and the rest of the processes:

- all passive edges of P'_{ABS} are removed and replaced with the ε-self loops;
- when removing a passive edge whose action contains a read from some channel c (a write to some channel c):
  - in $\{P_i'\}_{i=0}^{k}$, for all $j \in \{k+1, \dots, N\}$, all writes to c_j (all reads from c_j), where c_j is a channel of P_j that corresponds to c (the processes are identical), are removed;
  - when removing a read of a message m:
    - in the guards dependent on m, the minimal subformulae dependent on m are replaced with $undef$;
- the active edges of P'_{ABS} are processed as follows:
  - all assignments to the local variables are removed;
  - when removing an assignment to a local variable x:
    - in the guards dependent on x, the minimal subformulae dependent on x are replaced with $undef$;
  - each read from a global channel c is not modified:
    - in $\{P_i'\}_{i=0}^{k}$, writes to c are not modified;

- each write to a global channel c is removed:
  - in $\{P'_i\}_{i=0}^{k}$, each read $c\,?\,m$ is supplemented with the alternatives $\{m = v_j\}_{j=1}^{t}$, where $\{v_j\}_{j=1}^{t}$ contains all possible values that P'_{ABS} can send via c.

Statement. The processes $\{f_i(P_i, R_i)\}_{i=0}^{k}$ (constructed as it is described above) are bijective abstractions, while the process $g(P_{k+1}, \dots, P_N; R_{k+1})$ is a serializing abstraction. Thus, M' is an abstraction of M.

As the description is informal, the statement is given without a proof. It should be noticed that the abovementioned method has been implemented in a tool prototype. Given a PROMELA model, the tool parses the code, builds the abstract syntax tree, transforms it according to the rules, and maps it back to PROMELA.

5. Case study

The tool and the underlying method were used to verify the MOSI family CCPs implemented in the Elbrus computer systems. The developed PROMELA model supports memory accesses of the types Write Back, Write Through, and Write Combined. The experiments were performed on Intel Core i7-4771 with a clock rate of 3.5 GHz. The verified properties are as follows:

- $\mathbf{G}\{\neg(cache[1] = M \wedge cache[2] = M)\}$;
- $\mathbf{G}\{\neg(cache[1] = O \wedge cache[2] = O)\}$;
- $\mathbf{G}\{\neg(cache[1] = M \wedge cache[2] \in \{O, S\})\}$.

Table 1 and Table 2 show time and memory resources consumed for checking the property (1), respectively, on the original model ($n = 3$) and on the abstract one. Note that in the case $n = 3$ abstraction preserves the number of processes: $home(0)$, $proc(1)$, and $proc(2)$ are replaced with their abstract counterparts, while *proc*(3) is replaced with $proc_{env}(ABS)$.

Table 1. Resources required for checking the original model

SPIN optimization	State space size	Memory consumption	Verification time
$Absent$	5.1×10^6	682 Mb	9 s
$COLLAPSE$	5.1×10^6	328 Mb	15 s

Table 2. Resources required for checking the abstract model

SPIN optimization	State space size	Memory consumption	Verification time
$Absent$	2.2×10^6	256 Mb	3.7 s
$COLLAPSE$	2.2×10^6	108 Mb	6.2 s

The tables show that even for $n = 3$ there is a gain in state space size and memory consumption. Meanwhile, correctness of the abstract model implies correctness of the

original one for any $n \geq 3$. It is shown that the suggested approach reduces verification of the parameterized CCP model to visiting and testing $\sim 10^6$ states, which requires ~ 100 Mb of memory.

6. Conclusion

SMP computer systems utilize complicated caching mechanisms. To ensure that multiple copies of the same data are kept up-to-date, CCPs are employed. Errors in the CCPs and their implementations may cause data corruption and system hanging. This explains why CCP verification methods are of high value and importance.

The main problem arising in CCP verification is state explosion. In this paper, we have proposed an approach to overcome the issue and make verification scalable. The method having been described is aimed at transforming a CCP PROMELA model so as the result is independent of the number of processors and can be verified by the SPIN model checker on a regular basis. The approach was successfully applied to the MOSI family CCPs implemented in the Elbrus computer systems.

In the future, we are planning to extend the method with CEGAR, to develop an open-source tool for syntactical transformations of PROMELA models (a prototype is already available), and to create a unified model-based technology for checking CCPs and verifying memory management units.

References

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

[2]. Kim A.K., Perekatov V.I., Ermakov S.G. Microprocessors and computer systems of the Elbrus familty. SPb.: Piter, 2013. 272 p. (in Russian).

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

[4]. Kamkin A.S., Petrochenkov M.V. A system to support formal methods-based verification of coherence protocol implementations. Voprosy radioehlektroniki. Ser. EVT. [Issues of radio electronics], 2014, issue 3, pp. 27-38 (in Russian).

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

[6]. Burenkov V.S. An analysis of the SPIN model checker applicability to cache coherence protocols verification. Voprosy radioehlektroniki. Ser. EVT [Issues of radio electronics], 2014, issue 3, pp. 126-134 (in Russian).

[7]. Emerson E.A., Kahlon V. Exact and Efficient Verification of Parameterized Cache Coherence Protocols. Correct Hardware Design and Verification Methods, IFIP WG 10.5 Advanced Research Working Conference, 2003, pp. 247-262.

[8]. Holzmann, G.J. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 2003, 608 p.

[9]. Park S., Dill D.L. Verification of FLASH Cache Coherence Protocol by Aggregation of Distributed Transactions. Annual ACM Symposium on Parallel Algorithms and Architectures, 1996, pp. 288-296.

[10]. Ip C.N., Dill D.L. Verifying Systems with Replicated Components in Murphi. International Conference on Computer Aided Verification, 1996, pp. 147-158.

[11]. Pnueli A., Xu J., Zuck L. Liveness with $(0, 1, \infty)$-Counter Abstraction. International Conference on Computer Aided Verification, 2002, pp. 107-122.

[12]. Clarke E., Talupur M., Veith H. Environment Abstraction for Parameterized Verification. Verification, Model Checking, and Abstract Interpretation, 2006. LNCS, vol. 3855, pp. 126-141.
[13]. Clarke E., Talupur M., Veith H. Proving Ptolemy Right: The Environment Abstraction Framework for Model Checking Concurrent Systems. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 33-47.
[14]. 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.
[15]. 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.
[16]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.
[17]. Talupur M., Tuttle M.R. Going with the Flow: , pp. 1-8.
[18]. 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.

Проверка параметризованных PROMELA-моделей протоколов когерентности памяти

[1] *В.С. Буренков <burenkov_v@mcst.ru>*
[2] *А.С. Камкин <kamkin@ispras.ru>*
[1] *АО «МЦСТ»*
119334, Россия, г. Москва, ул. Вавилова, 24.
[2] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, 25

Аннотация. В статье представлен метод масштабируемой верификации PROMELA-моделей протоколов обеспечения когерентности памяти. Под масштабируемостью понимается независимость затрат на верификацию (прежде всего, машинного времени и памяти) от числа процессоров в системе. Метод состоит из трех основных шагов. На первом шаге в модель протокола, созданную для определенной конфигурации системы (для конкретного числа процессоров), вводится параметр, представляющий число процессоров в системе. Для этого используются простые индуктивные правила, что возможно только при определенных допущениях на вид протокола. На втором шаге построенная параметризованная модель абстрагируется от числа процессоров. Для этого над присваиваниями, выражениями и коммуникационными действиями модели совершается ряд синтаксических преобразований. На третьем шаге полученная абстрактная модель верифицируется с помощью инструмента SPIN обычным образом. Помимо описания метода, в статье приводится доказательство его корректности:

утверждается, что предложенная схема абстракции является консервативной в том смысле, что любой инвариант (свойство истинное во всех достижимых состояниях) абстрактной модели является инвариантом исходной модели (свойства-инварианты — это именно те свойства, которые представляют интерес при верификации протоколов обеспечения когерентности памяти). Предложенный метод был воплощен в прототипе инструмента, который разбирает код на языке PROMELA, строит дерево абстрактного синтаксиса, преобразует его по заданным правилам и отображает обратно в PROMELA код. Инструмент (и метод в целом) был успешно использован при верификации протоколов семейства MOSI, разработанных в АО «МЦСТ» и реализованных в вычислительных комплексах «Эльбрус».

Ключевые слова: многоядерные микропроцессоры, мультипроцессоры с разделяемой памятью, протоколы когерентности памяти, проверка моделей, SPIN, PROMELA.

DOI: 10.15514/ISPRAS-2016-28(4)-4

Для цитирования: Буренков В.С., Камкин А.С. Проверка параметризованных PROMELA-моделей протоколов когерентности памяти. Труды ИСП РАН, том 28, вып. 4, 2016 г. стр. 57-76 (на английском). DOI: 10.15514/ISPRAS-2016-28(4)-4

Список литературы

[1]. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.
[2]. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». Спб.: Питер, 2013. 272 с.
[3]. Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011, 195 p.
[4]. Камкин А.С., Петроченков М.В. Система поддержки верификации реализаций протоколов когерентности с использованием формальных методов. Вопросы радиоэлектроники. Серия ЭВТ, 2014, вып. 3, стр. 27-38.
[5]. Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999, 314 p.
[6]. Буренков В.С. Анализ применимости инструмента SPIN к верификации протоколов когерентности памяти. Вопросы радиоэлектроники. Серия ЭВТ, 2014. вып. 3, стр. 126-134.
[7]. Emerson E.A., Kahlon V. Exact and Efficient Verification of Parameterized Cache Coherence Protocols. Correct Hardware Design and Verification Methods, IFIP WG 10.5 Advanced Research Working Conference, 2003, pp. 247-262.
[8]. Holzmann, G.J. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 2003, 608 p.
[9]. Park S., Dill D.L. Verification of FLASH Cache Coherence Protocol by Aggregation of Distributed Transactions. Annual ACM Symposium on Parallel Algorithms and Architectures, 1996, pp. 288-296.
[10]. Ip C.N., Dill D.L. Verifying Systems with Replicated Components in Murphi. International Conference on Computer Aided Verification, 1996, pp. 147-158.
[11]. Pnueli A., Xu J., Zuck L. Liveness with (0, 1, ∞)-Counter Abstraction. International Conference on Computer Aided Verification, 2002, pp. 107-122.

[12]. Clarke E., Talupur M., Veith H. Environment Abstraction for Parameterized Verification. Verification, Model Checking, and Abstract Interpretation, 2006. LNCS, vol. 3855, pp. 126-141.
[13]. Clarke E., Talupur M., Veith H. Proving Ptolemy Right: The Environment Abstraction Framework for Model Checking Concurrent Systems. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 33-47.
[14]. 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.
[15]. 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.
[16]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.
[17]. Talupur M., Tuttle M.R. Going with the Flow: , pp. 1-8.
[18]. 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.