Computing Invariants for Parameter Abstraction*

Yi Lv, Huimin Lin, Hong Pan
State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, P.O.Box 8718, Beijing 100080, China
Email:{lvyi, lhm, ph}@ios.ac.cn

Abstract

A new approach to combining invariants computing and guard strengthening methods is presented in the context of parameter abstraction for parameterized model checking of cache coherence protocols. The approach uses a small instance of a parameterized protocol as a "reference model" to compute candidate invariants. References to a specific node in these candidate invariants are then abstracted away, and the resulting formulas are used to strengthen guards of the transition rules in the abstract node. The correctness of the approach is guaranteed by symmetry which exists in many parameterized systems. A number of case studies have been carried out to illustrate the effectiveness of the approach. During the process a data consistency error was identified and fixed in the German 2004 cache coherence protocol.

1 Introduction

Verification of parameterized concurrent systems has attracted considerable interests from both model checking and theorem proving communities, due to practical importance of such systems. Let $\mathcal{P}$ be a parameterized system and $\mathcal{P}(N)$ denotes its instance with the cardinality N.[1] $\mathcal{P}(N)$ usually consists of a small number (may be none) of heterogeneous processes, plus a set of homogeneous processes $\{ P_i \mid 0 \leq i \leq N \}$. Parameterized systems exist in many importance application areas: cache coherence protocols, security protocols, communication protocols and network protocols, to name just a few. The challenge posed by parameterized systems is that one can only verify the correctness of such a system for a fixed (usually very small) number of instances, which does not imply the correctness of the system with arbitrary sizes.

Although the problem of parameterized systems verification is in general undecidable [1], many techniques, either automatic or interactive, have been developed and successfully applied to specific subsets of parameterized system. *Parameter abstraction and guard strengthening*, proposed in [7] with a rigorous proof presented in [13], is one of such techniques. It is designed for verifying safety properties of parameterized systems by means of model checking. The work-flow of the method goes as follows: Given a parameterized system instance $\mathcal{P}(N) = \{ P_i \mid 0 \leq i \leq N \}$ and a safety property, expressed as an invariant ϕ, to verify, one first decides on a small number m, and merge all the nodes $P_i,\ m < i \leq N$ into an *abstract node* P^* using an *abstraction function*. The abstract system consists of $m + 1$ nodes $\{P_1, \ldots, P_m, P^*\}$ with m normal nodes and one abstract node. The abstract system normally does not satisfy the invariant ϕ. Nevertheless it is still submitted to a model checker for verification. When a counter-example is produced, the human verifier carefully analyzes it and comes up with an auxiliary invariant ψ, then uses it to strengthen the guards of some transition rules of the abstract node. The "strengthened" system is then subject to model checking again. This process may iterate several times, until the refined abstract system is, if lucky enough, eventually found to satisfy the original invariant ϕ, *as well as* all the auxiliary invariants supplied by the verifier. It was demonstrated in [7] that this method is powerful enough to handle complex cache coherence protocols such as FLASH effectively.

It can be seen that the most challenging part of this method lies in finding appropriate auxiliary invariants for guard strengthening, a task that calls for human intelligence. In this paper we shall report our on-going research aiming at replacing a large part of invariant-finding with *invariant-computing*, and using abstracted auxiliary invariants to guard strengthening the abstract system, both can be carried out mechanically. Our approach is partly inspired by the work of "invisible invariants" [19].

In our approach a small instance of the parameterized system is used as a reference model. In the case of the ab-

*Supported by research grants from Intel Strategic CAD Labs and Natural Science Foundation of China (60421001).

[1] In general there may be several such sets but for simplicity we shall focus on the case of only one set in this paper, as many interesting applications fall into this category.

stract system $\{P_1, \ldots, P_m, P^*\}$ described above, the reference model will consist of $m + 1$ normal nodes among which the $(m + 1)$-th node is used as the "reference" for the abstract node P^* in the abstract system. Since the reference model is concrete and small, its strongest invariant denoting the set of reachable states can be computed using a suitable model checker. For each transition rule of the abstract node, candidates for the auxiliary invariants are generated by computing both correspondent transition rule of reference node and reachable states set in the reference model. These candidates may contain references to the reference node, hence are not suitable for the abstract system. Such references can be abstracted away using a technique, which will be detailed in Section 3, to create permutable invariants. The abstracted versions of the auxiliary invariants are employed to strengthen the relevant guards in the abstract model. The modified abstract system is then model checked for the safety properties *as well as* these auxiliary invariants. Our approach is sound, in the sense that if the abstract system satisfies the safety properties and the auxiliary invariants are indeed invariants of the abstract system, then the original parameterized system also satisfies the safety properties. In general, the invariant-computing and abstraction process may be repeated for a few times, and some auxiliary variables may need to be introduced.

We have implemented the approach using the TLV tool [18]. Any verification tools that accept CMU SMV language, such as Cadence SMV, TLV and NuSMV, can be used to model check the abstract systems generated by our approach. The method has been applied to German 2000, German 2004 and FLASH cache coherence protocols. While working on the German 2004 protocol, a counterexample was reported. By analyzing the counterexample a data-consistence error was revealed. The error was reported to the designer of the protocol and got fixed.

The rest of this paper is organized as follows. The remaining part of this section is devoted to discussions on related work. Section 2 provides a brief overview of the parameter abstraction and guard strengthening method. The approach for calculating auxiliary invariants from a reference model is presented in Section 3. Section 4 reports case studies. The paper is concluded with Section 5 where directions for future work is also outlined.

Related Work Automatic discovery of invariants is not a new idea. A large amount of research has been conducted in this direction [4, 6, 20]. However, so far no method has been shown uniformly good across a spectrum of examples. Recently, a heuristic-based method for discovering invariants for parameterized verification of safety properties was reported in [17]. The heuristics are created by syntactic analysis of the counterexamples generated during verification, combined with simple static analysis of predicates involved in the counterexamples. These heuristics are then used to construct and refine invariants. The method is implemented using the UCLID tool with EUF decision procedures.

The "invisible invariants" method, proposed in [19], is another automatic technique for parameterized verification. In this method, auxiliary invariants are computed in a finite system instance. Their idea of computing invariants in a finite instance is a source of inspiration for our work. This method has been extended so that complex protocols such that FLASH and German 2004 protocols can be handled [2]. However, the cut-off in [2] is rather large. For example, the cut-off size of finite instance of German 2004 is 14, beyond the capability of any brute-force model checking tool. In [5] a new finite-state symbolic model checking algorithm is presented for safety property verification that reduces a conjunctively guarded protocol to a broadcast protocol. Their method can verify German 2000 automatically. But it is not clear whether it works for FLASH and German 2004.

The literature in the area of cache coherence protocol verification are abundant. Here we only list a few. In [10] German 2000 protocol have been verified by reducing it into a snoopy protocol, for which the cut-off is known to be 7. Safety properties of FLASH was proved using the PVS theorem prover in [16]. In a different context, safety properties of German 2000 and FLASH were proved using Murφ tool in [7]. Both methods are based on the notion of simulation, which is an over-approximation [8]. In [15] safety and liveness properties of FLASH were verified in Cadence SMV with built-in abstraction and symmetry reductions, using compositional model checking method. In all the three methods mentioned above, auxiliary invariants have to be supplied manually by the verifiers, which requires considerable human insights and skills in proving conjectures and using the tool. Predicate abstraction based methods were applied to verify German 2000 in [3], and FLASH in [9]. Users need to provide plausible properties in predicate abstraction and automated predicate discovery techniques to find large predicates. So verifying large protocols like FLASH using predicate abstraction is difficult. In [14] a method for automatically discovering indexed predicates is developed which can be applied to construct inductive invariants. With this technique a version of German 2000 protocol with unbounded FIFO queues was verified.

2 Preliminaries

We assume a set of *basic types* including boolean B and a finite *parameter set* $N = \{1, 2, \ldots, n\}$ for a fixed n, with $1, 2, \ldots, n$ being the only constants of N. We can easily relax the data type restriction to allow arbitrary finite types instead of just boolean. Types are generated from basic types using standard product $\times$, disjoint union $+$ and function space $\Rightarrow$ constructors. Arrays are functions from basic

types to basic types. *Terms* are constructed from constants and variables using typed operators in the usual way. *Predicates*, also called *formulas*, are terms of type B.

Valuations, ranged over by s, are total, type-respecting mappings from V to the domain of all typed values. We shall write $s(e)$ for the value of the term e evaluated in s, and use $s \models f$ to mean $s(f) = true$.

A *system presentation* is a triple $P = (V, \Theta, \Delta)$ where

- V is the set of system variables.
- $\Theta(V)$ is the *initial predicate*.
- $\Delta(V, V')$ is the set of *transition rules*, each rule $\delta \in \Delta(V, V')$ being of the form $\rho(V) \rightsquigarrow a(V, V')$, where $\rho(V)$ is the *guard predicate* and $a(V, V')$ is the *transition relation* of the rule.

The precondition $\rho(i)_\delta$ of each rule δ with index i, j of parameter type are of the following special form:

$\varrho(i)_\delta \wedge (\bigwedge_{j=1}^{N} \tau_1(i,j)_\delta \wedge \ldots \wedge \bigwedge_{j=1}^{N} \tau_k(i,j)_\delta) \wedge$
$(\bigvee_{j=1}^{N} \varsigma_1(i,j)_\delta \wedge \ldots \wedge \bigvee_{j=1}^{N} \varsigma_l(i,j)_\delta)$

where $\varrho(i)_\delta$, $\tau(i,j)_\delta$ and $\varsigma(i,j)_\delta$ are quantifier-free formulas.

As an illustration, in the mutual exclusion protocol example shown in Figure 1 (written in the Murφ language), we have two system variables `x` and `n`. The `StartState "Init"` block defines the initial predicate of the system. There are four transition rules named `Try`, `Crit`, `Exit` and `Idle`. The precondition of the rule `Crit` is `n[i].state = T & x = true`, while its transition relation is given implicitely by the assignment statements `n[i].state := C; x := false`.

A *state transition system* is a triple (S, I, R) where S is the set of *states*, $I \subseteq S$ is the set of *initial states*, and $R \subseteq S \times S$ is the *transition relation*. We shall write $s \longrightarrow s'$ to mean $s, s' \in R$. A state s is *reachable* if there is a finite sequence of transitions starting from some initial state and ending at s.

A presentation $P = (V, \Theta, \Delta)$ induces a state transition system $|P| = (S, I, R)$ in the following way: S is the set of all valuations for V; I is the subset of S that satisfies Θ; For any $s, s' \in S$, $(s, s') \in R$ if there exists some $\rho(V) \rightsquigarrow a(V, V') \in \Delta(V, V')$ such that $s \models \rho(V)$ and $a(V, V')$ evaluates to $true$ when each $v \in V$ is assigned the value $s(v)$ and each $v' \in V'$ is assigned the value $s'(v')$. If $s \models \phi$ holds for all reachable states in S then we shall call ϕ an *invariant* of P, denoted $P \models \phi$.

Let $\pi : N \Rightarrow N$ be a permutation on the parameter set N. It induces a transformation on terms, also denoted π, as follows: for any term e, $\pi(e)$ is the term obtained by replacing in e every occurrence of every constant $k \in N$ with $\pi(k)$. A formula f is *symmetric* if $\vdash f \iff \pi(f)$ for every permutation π on N. The conjunction $f_{sym} =$

```
NODE : 100;
VAR
  x : boolean;
  state : enum {I, T, C, E};
  n : array [NODE] of state;

StartState "Init"
  x := true;
  for i : NODE do n[i].state := I;
end; end;

ruleset i : NODE do rule "Try"
  n[i].state = I ==> n[i].state := T;
end;

ruleset i : NODE do rule "Crit"
  n[i].state = T & x = true ==>
  n[i].state := C; x := false;
end;

ruleset i : NODE do rule "Exit"
  n[i].state = C ==> n[i].state := E;
end;

ruleset i : NODE do rule "Idle"
  n[i].state = E ==> n[i].state := I;
  x := true;
end;

Invariant "Mutual Exclusion"
forall i : NODE do forall j : NODE do
(i!=j) -> !(n[i].state=C & n[j].state=C)
end end;
```

Figure 1. Mutual Exclusion Model

$\bigwedge_\pi \pi(f)$ taken over all permutations on N is the weakest symmetric formula which implies f.

For example, applying the permutation $\pi : 1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 1$ to formula $\phi = \neg(n[1].state{=}C \wedge n[2].state{=}C)$ yields the formula $\pi(\phi) = \neg(n[2].state{=}C \wedge n[3].state{=}C)$. Furthermore, $\phi_{sym} = \forall i, j : (i \neq j) \rightarrow \neg(n[i].state = C \wedge n[j].state = C)$ is the weakest symmetric formula of ϕ.

A presentation $P = (V, \Theta, \Delta)$ is symmetric when

- Θ is a symmetric formula;
- For every permutation π and rule $\rho \rightsquigarrow a \in \Delta$, there exists a rule $\rho' \rightsquigarrow a' \in \Delta$ which satisfies $\rho' = \pi(\rho)$ and $a' = \pi(a)$.

Let P be a symmetric presentation on N, $M = \{1, \ldots, m\}$ a subset of N, and $M_* = M + \{*\}$. For each type T of P we define an abstract type $[T]$ as follows: $[N] = M_*$ and $[T] = T$ for other basic types, $[B \Rightarrow B] = B \Rightarrow B$, $[B \Rightarrow N] = B \Rightarrow M_*$, $[N \Rightarrow B] = M \Rightarrow B$, and $[N \Rightarrow N] = M \Rightarrow M_*$. We then define a syntactic transformation H_E on terms, by letting $H_E(i) =$ if $i \leq m$ then i else $*$, $H_E(u) = u$ for other constant or variable u, and $H_E(op(e_1, \ldots, e_k)) = op(H_E(e_1), \ldots, H_E(e_k))$.

For formulas we further define a total function H_F as follows. Let f be a formula which is a boolean combination of its atom formulas $f_1, \ldots, f_k$. Then $H_F(f)$ is the same boolean combination of the atom formulas $H_F(f_1), \ldots, H_F(f_k)$, where

$$H_F(f_i) = \begin{cases} f_i & \text{if } H_E(f_i) = f_i \\ true & \text{if } H_E(f_i) \neq f_i, \text{where } f_i \text{ is positive in } f \\ false & \text{if } H_E(f_i) \neq f_i, \text{where } f_i \text{ is negative in } f \end{cases}$$

Now we define the *abstraction* of a symmetric presentation $P = (V, \Theta, \Delta)$ to be $P_m = (V, \Theta_m, \Delta_m)$, where $\Theta_m = H_F(\Theta)$, and for every rule $\rho \rightsquigarrow a \in \Delta$, there exists a rule $H_F(\rho) \rightsquigarrow H_F(a) \in \Delta_m$. Although P_m and P have the same set of system variables V but the variables of P_m are of abstract types.

As an example, let us consider the mutual exclusion protocol in Figure 1. We choose $m = 2$ to construct the abstract model. In the abstract model, all occurrences of `i`, for `i > 2`, are abstracted away. For instance, the abstract versions of the guard and the transition of the `Crit` rule are `x = true` (abbreviated as `x` in Murφ) and `x := false`, respectively.

We quote the following theorem from [13]:

Theorem 2.1 *Let $P = (V, \Theta, \Delta)$ be a symmetric presentation and P_m its abstraction. Suppose that only the first m elements of N may occur in ϕ. Let Q be the presentation obtained from P_m by replacing each rule $H_F(\rho) \rightsquigarrow H_F(a)$ in P_m with $H_F(\rho^\#) \rightsquigarrow H_F(a)$, where $\vdash \rho \wedge \phi_{sym} \Longrightarrow \rho^\#$. Then $Q \models \phi$ implies $P \models \phi_{sym}$.*

```
NODE : 2;
ABS_NODE: NODE + 1;
VAR
  x : boolean;
  state : enum {I, T, C, E};
  n : array [NODE] of state;

StartState "Init"
  x := true;
  for i : NODE do n[i].state := I;
end; end;

ruleset i : NODE do rule "Try"
  n[i].state = I ==> n[i].state := T;
end;

ruleset i : NODE do rule "Crit"
  n[i].state = T & x = true ==>
  n[i].state := C; x := false;
end;

ruleset i : NODE do rule "Exit"
  n[i].state = C ==> n[i].state := E;
end;

ruleset i : NODE do rule "Idle"
  n[i].state = E ==> n[i].state := I;
  x := true;
end;

rule "ABS_Skip" end;

rule "ABS_Crit"
  x ==>
  x := false;
end;

rule "ABS_Idle"
  true ==>
  x := true;
end;

Invariant "Mutual Exclusion"
forall i : NODE do forall j : NODE do
(i!=j) -> !(n[i].state=C & n[j].state=C)
end end;
```

Figure 2. Abstract Mutual Exclusion Model

This theorem justifies the "guard strengthening" method, proposed in [7], which works as follows: Given a symmetric presentation P and a symmetric property $\phi_0 = \phi_{sym}$ to verify, first construct an abstract system $Q_0 = P_m$ from P and model check Q_0 against ϕ. If this fails then locate a rule in Q_0 and devise a formula ψ, called a "non-interference lemma", to strengthen the guard of the rule, and model check the modified system against both ϕ and ψ. If this fails too then repeat the process, until the resulting system satisfies ϕ and all the non-interference lemmas.

For more details on parameter abstraction and guard strengthening we refer the readers to [7, 13].

3 Computing Auxiliary Invariants

In this section we discuss how to compute the auxiliary invariants for parameter abstraction and guard strengthening. Let us fix a symmetric system presentation parameterized on N, a symmetric property $\phi_0 = \phi_{sym}$ to be verified on P, and a number m which is far less than N. Given a system we fix an m according to the maximal number of different variables of parameter type in the transition rules and the property specification of the system.

After an abstract system P_m has been created from P by merging all but the first m nodes into an abstract node, the abstract protocol still permits all possible behaviors of the m preserved nodes, referred to as the basic component, but has no information about the internal behavior of the abstract node. In P the m nodes interact with other nodes, thus only when the m nodes are in certain states the corresponding state transitions of the other nodes can take place. After abstraction, such information gets lost. As a consequence, some unexpected behavior of the abstract node may happen, which could have undesirable impact on the behavior of the m preserved nodes. The key issue in parameter abstraction is to devise suitable "auxiliary invariants" to constrain the behavior of the abstract node so that the preserved nodes, when "interacting" with the abstract node in the abstract system, can simulate the transitions they can engage in when they are in the original system. In the method proposed in [7] such auxiliary invariants are provided by the human verifier in a sequence of "try-and-error" efforts, through careful analysis of the diagnosis messages generated from a model checker.

We would like to automate and simplify such a process as much as possible. From the above analysis we can see that, among the transition rules of the abstract node, only those which can change the values of some global variables and local variables of preserved nodes are of interest. Let us call such rules "significant". We wish to constrain the guard of a significant rule δ in such a way that the rule can only be "fired" if the global variables and local variables of preserved nodes (the variables of basic component) whose values to be changed by the rule are in a similar state as in P when some of the rules "merged" into δ can be fired. Such information can be extracted from the formula characterizing all the reachable states of P, denoted by *reach*. Let P' be the systems obtained by conjugating *reach* with the precondition ρ of every rule in P. We can observe that the behavior of P' is the same as P. In P' the precondition of each rule δ in a node i has the form $reach \wedge \rho(i)_\delta$ where $\rho(i)_\delta$ is the guard predicate of δ. Intuitively, $reach \wedge \rho(i)_\delta$ denotes the set of states in which node i can perform the transition specified in δ. This formula is the strongest global constraint as guard for δ. Besides node i, the formula also contains the state information of the other nodes. By symmetry, we can chose any node j which is different from i as a representative. The state information of node j can be extracted out by projecting away all other nodes. This state information is still a constraint for guard of δ. In other words, we try to find the local constraint relationship between two nodes.

Although *reach* is computable, for large P it is infeasible to compute. Our solution is to use a small model $P(m+1)$ as a reference and compute the reachable state set of $P(m+1)$ as an approximation. Let *reach* be the formula for the reachable states of $P(m+1)$, and node $m+1$ in $P(m+1)$ serves as the "reference node" for the abstract node in P_m. Given rule δ in a node i and representative node j, we can compute the the local constraint relationship between two nodes. Technically, this can be done using BDD operations to compute $\varphi(i,j)_\delta = \exists V(1), \ldots, V(j-1), V(j+1), \ldots, V(m+1) : (reach \wedge \rho(i)_\delta)$ in system $P(m+1)$, where $i \in \{1, \ldots, m+1\}$ and $j \in \{1, \ldots, m\}$. Let $\psi(i,j)_\delta = \rho(i)_\delta \rightarrow \varphi(i,j)_\delta$.

Among the $m+1$ nodes in $P(m+1)$, node $m+1$ is of special interest as it is the reference node for the abstract node in P_m. In particular, $H_F(\rho(m+1)_\delta \wedge \bigwedge_{j=1}^{m}(\rho(m+1)_\delta \rightarrow \varphi(m+1,j)_\delta)) = H_F(\rho(m+1)_\delta) \wedge \bigwedge_{j=1}^{m} \varphi(m+1,j)_\delta$ will be used to strengthen the guard of the rule in the abstract node of P_m which corresponds to δ in node $m+1$ of $P(m+1)$. To justify this, we appeal to Theorem 2.1. First we note that, for any j,

$$\begin{array}{l} \vdash \rho(m+1)_\delta \wedge (\psi(m+1,j)_\delta)_{sym} \Longrightarrow \\ \rho(m+1)_\delta \wedge \bigwedge_{j=1}^{m}(\rho(m+1)_\delta \rightarrow \varphi(m+1,j)_\delta) \end{array} \quad (1)$$

Besides this, the Theorem 2.1 also requires that $\psi(m+1,j)_\delta$ should not contain $m+1$, which is not true. But, since $(\psi(m+1,j)_\delta)_{sym} = \pi(\psi(m+1,j)_\delta)_{sym}$ for any permutation π, in most cases we can choose a suitable π such that $\pi(\psi(m+1,j)_\delta)_{sym} = (\psi(i,j)_\delta)_{sym}$ and $m+1$ does not occur in $\psi(i,j)_\delta$, where $1 \leq i,j \leq m$. Thus we use $\pi(\psi(m+1,j)_\delta)_{sym}$ to replace $(\psi(m+1,j)_\delta)_{sym}$ in (1). Note that $\pi(\psi(m+1,j)_\delta)$ can be computed from $P(m+1)$ in the same way as $\psi(m+1,j)_\delta$.

In some situations special transformations are needed: (a) The formula computed from $P(m+1)$ may contain

sub-formulas of the form $v = m+1$. In this case we can substitute $\bigwedge_{i=1}^{m} \neg(v = i)$ and $\bigvee_{i=1}^{m} v = i$ for $v = m+1$ and $\neg(v = m+1)$, respectively. (b) The formula may contain sub-formulas of the form $\pi(\varrho(m+1)_\delta) \wedge \bigwedge_{j=1}^{m+1} \tau(j)_\delta \rightarrow \pi(\varphi(m+1,j)_\delta)$, which can be written as $\varrho(i)_\delta \wedge \bigwedge_{j=1}^{m+1} \tau(j)_\delta \rightarrow \varphi(i,j)_\delta$, where $1 \leq i,j \leq m$. It's sub-formula $\tau(m+1)_\delta$ may contain $m+1$. In this case we simply use $\varrho(i)_\delta \wedge \bigwedge_{j=1}^{m} \tau(j)_\delta \rightarrow \varphi(i,j)_\delta$ as approximation. (c) When the sub-formula is of the form $\pi(\varrho(m+1)_\delta) \wedge \bigvee_{j=1}^{m+1} \varsigma(j)_\delta \rightarrow \pi(\varphi(m+1,j)_\delta)$, we use $\pi(\varrho(m+1)_\delta) \rightarrow \pi(\varphi(m+1,j)_\delta)$ as approximation. For many parameterized cache coherence protocols, including those analyzed in the case studies reported in the following section, the guards of transition rules are in the form as required. For such protocols the approximation is sound as shown by the following proposition.

Proposition 3.1 *Let* $\rho(m+1)_\delta = \varrho(m+1)_\delta \wedge (\bigwedge_{j=1}^{N} \tau_1(m+1,j)_\delta \wedge \ldots \wedge \bigwedge_{j=1}^{N} \tau_k(m+1,j)_\delta) \wedge (\bigvee_{j=1}^{N} \varsigma_1(m+1,j)_\delta \wedge \ldots \wedge \bigvee_{j=1}^{N} \varsigma_l(m+1,j)_\delta)$ *be a guard of* δ *in node* $m+1$ *of* P *which corresponds to* δ *in the abstract node of* P_m*, and* $\psi = (\varrho(i)_\delta \wedge (\bigwedge_{j=1}^{m} \tau_1(i,j)_\delta \wedge \ldots \wedge \bigwedge_{j=1}^{m} \tau_k(i,j)_\delta) \rightarrow \varphi(i,h)_\delta$ *be an auxiliary invariant generated from* P_m*, where* $1 \leq i,h \leq m$ *and* $i \neq h$*. Then* $\varrho(m+1)_\delta \wedge (\bigwedge_{j=1}^{N} \tau_1(m+1,j)_\delta \wedge \ldots \wedge \bigwedge_{j=1}^{N} \tau_k(m+1,j)_\delta) \wedge \bigwedge_{j=1}^{m} \varphi(m+1,j)_\delta$ *is a logical consequence of* $\rho(m+1)_\delta \wedge \psi_{sym}$.

When constructing the abstract model we mark each transition rule of the abstract node that can change the state of the basic component, and indicate which rule in the reference model it corresponds to, so the candidate invariants can be computed from the guards of the marked rules and the reference model, in the way as stated above.

Continue with the mutual exclusion protocol. As we have chosen $m = 2$ to construct the abstract model, both the abstract and the reference models will consist of 3 nodes. We take $\phi = \neg(n[1].state{=}C \ \wedge n[2].state{=}C)$ as mutual exclusion property. The two rules of the abstract node which need guard strengthening are marked by p_1 and p_2, as shown in Figure 3. The guards of their corresponding rules in the reference model are $\rho_1 \equiv n[3].state = T \wedge x = true$ and $\rho_2 \equiv n[3].state = E$, respectively. The two "markers" p_1 and p_2 will be replaced by auxiliary invariants once they have been computed. First, the candidate auxiliary invariants can be calculated as follows:

```
NODE : 2;
ABS_NODE: NODE + 1;
-- Include the original mutual exclusion
-- protocol here.

rule "ABS_Skip" end;

rule "ABS_Crit"
  p1 & x ==>
  x := false;
end;

rule "ABS_Idle"
  p2 ==>
  x := true;
end;

Invariant "Mutual Exclusion"
forall i : NODE do forall j : NODE do
(i!=j) -> !(n[i].state=C & n[j].state=C)
end end;
```

Figure 3. Marked Abstract Model

$$
\begin{aligned}
\varphi(3,1)_1 &= \exists n[2].state, \exists n[3].state : reach \wedge \\
&\quad (n[3].state = T \wedge x = true) \\
&= x = true \wedge n[1].state \in \{T, I\} \\
\varphi(3,1)_2 &= \exists n[2].state, \exists n[3].state : reach \wedge \\
&\quad (n[3].state = E) \\
&= x = false \wedge n[1].state \in \{T, I\} \\
\psi(3,1)_1 &= (n[3].state = T \wedge x = true) \rightarrow \\
&\quad (x = true \wedge n[1].state \in \{T, I\}) \\
\psi(3,1)_2 &= (n[3].state = E) \rightarrow (x = false \wedge \\
&\quad n[1].state \in \{T, I\})
\end{aligned}
$$

According to Theorem 2.1, we can use the abstract forms of $p_1 = \psi(3,1)_1 \wedge \psi(3,2)_1$ and $p_2 = \psi(3,1)_2 \wedge \psi(3,2)_2$ to strengthen the guards in the abstract model. However both $\psi(3,1)_1$ and $\psi(3,1)_2$ contain $m+1$, hence can not be verified in abstract model. Thus we choose $\pi : 2 \leftrightarrow 3$ and compute the permutations $\pi(\psi(3,1)_1) = \psi(2,1)_1$ and $\pi(\psi(3,1)_2) = \psi(2,1)_2$. In addition to the mutual exclusion property ϕ, we also check that the abstract model satisfy $\psi(2,1)_1$ and $\psi(2,1)_2$, see Figure 4. For this example, all checks give positive results. Hence the mutual exclusion property is verified in only one step of abstraction.

For more complicated systems, we may not be so lucky. Parameter abstraction is an over-approximation technique. Some information which is crucial to the correctness of the protocol may still be lost even after guard strengthening. As a consequence, the abstract system may not get model

```
NODE : 2;
ABS_NODE: NODE + 1;
-- Include the original mutual exclusion
-- protocol here.

rule "ABS_Skip" end;

rule "ABS_Crit"
  (n[1].state=T | n[1].state=I) &
  (n[2].state=T | n[2].state=I) & x ==>
  x := false;
end;

rule "ABS_Idle"
  (n[1].state=T | n[1].state=I) &
  (n[2].state=T | n[2].state=I) & !x ==>
  x := true;
end;

Invariant "Mutual Exclusion"
forall i : NODE do forall j : NODE do
(i!=j) -> !(n[i].state=C & n[j].state=C)
end end;

Invariant "Auxiliary Invariants"
(n[2].state=T & x ->
  x & (n[1].state=T | n[1].state=I))
& (n[2].state=E ->
  !x & (n[1].state=T | n[1].state=I))
```

Figure 4. Guard Strengthen Abstract Model

checked. One way to uncover such lost information is to introduce some auxiliary variables into the system which do not constraint nor change the behavior of the system. That is, these variables are used as "observers" of the system. In practice, we analyze counterexamples to figure out suitable auxiliary variables and introduce them into the system in such a way that they only occur in the left hand-side of any assignment statement or in the guard predicate part of a rule. The correctness of the procedure of introducing auxiliary variables is guaranteed by the following proposition.

Let $P = (V, \Theta, \Delta)$ and $P' = (V', \Theta', \Delta')$ are two system presentations. $|P| = (S, I, R)$ and $|P'| = (S', I', R')$ are two transition systems generated from P and P'. P and P' are *bisimilar* if there exists a *bisimulation relation* $E \subseteq S \times S'$ s.t. $(P, P') \in E$. Formally, E is a bisimulation if $I \times I' \subseteq E$, and for all $(s, s') \in E$ the following conditions are satisfied:

- if $s \longrightarrow s_1$ then there exists some state $s'_1 \in S'$ such that $s' \longrightarrow s'_1$ and $(s_1, s'_1) \in E$.
- if $s' \longrightarrow s'_1$ then there exists some state $s_1 \in S$ such that $s \longrightarrow s_1$ and $(s_1, s'_1) \in E$.

Proposition 3.2 *Suppose $P = (V, \Theta, \Delta)$ is a system presentation, and $P^+ = (V \cup \{aux\}, \Theta^+, \Delta^+)$ is obtained from P by adding boolean variable aux to P in such a way that it only occurs in the left hand-side of any assignment statement or in the guard predicate part of any rule in Θ^+ and Δ^+. P^+ and P are bisimilar if*

- *$P^+ \models aux \leftrightarrow pred(V)$ for some predicate $pred(V)$ not containing aux,*
- *$P^+ \models \rho^+ \leftrightarrow \rho$, where ρ is any guard predicate of P and ρ^+ is its corresponding guard predicate in P^+.*

It is well-known that bisimilar systems verify the same temporal properties. Therefore to verify $P \models \phi$ it is sufficient to check if $P^+ \models \phi$, for any ϕ. The work flow of our approach is as follows:

Verification process

Given a system P parameterized on N, $\phi_0 = \phi_{sym}$ a symmetric safety property to be verified such that only the first m elements of N occur in ϕ.

1. Create an abstract presentation P_m from P.
2. Mark each transition rule of an abstract node in P_m which can change the state of the basic component for guard strengthening.
3. For every marked rule δ of the abstract node, compute auxiliary invariant $\psi(m+1, j)_\delta = \rho(m+1)_\delta \rightarrow \varphi(m+1, j)_\delta$, where $j \in \{1, \ldots, m\}$, and its corresponding invariant $\pi(\psi(m+1, j)_\delta)$ from $P(m+1)$ where π is a chosen permutation such that $m+1$ does not occur in $\pi(\psi(m+1, j)_\delta)$.
4. Create Q by replacing the guard $\rho(*)_\delta$ of every rule δ in the abstract node with $H_F(\rho(m+1)_\delta) \wedge \bigwedge_{j=1}^{m} \varphi(m+1, j)_\delta$.
5. Check $Q \models \phi \wedge \bigwedge_\delta \pi(\psi(m+1, j)_\delta)$, where δ ranges over all marked abstract rules.
6. If the results of step 5 are confirmatory, stop. Otherwise, analyze the counterexamples generated in the step. If it is a real counterexample(which can be determined by model checking the reference model, for instance), then stop. Otherwise introduce auxiliary variables into P and modify the abstract model if necessary, go to step 2.

4 Case Studies

4.1 German 2000 Protocol

The German 2000 directory-based protocol was posted as a challenge to the formal verification community by Steven German in 2000. We use a SMV model which be translated from Murφ model of Chou etc[7] with data paths added to allow checking data consistency.

The safety property we need to verify are

$$\begin{aligned}&\forall i,j:(i\neq j)\rightarrow\\&\quad(\neg(n[i].state=E\wedge n[j].state=E)\wedge\\&\quad(\neg(n[i].state=E\wedge n[j].state=S)))\\&\forall i:((exgntd=0\rightarrow memory=auxdata)\wedge\\&\quad(n[i].state\neq I\rightarrow n[i].data=auxdata)\end{aligned}$$

Some rules containing sub-formulas $\rho \wedge v' = n[3].d$ in German 2000 and other protocols deserves attention during the abstraction procedure. After abstraction, any information of node $n[3]$ in the reference model should be abstracted away. As a result, we do not know the value of v'. Therefore we need to consider all possible values for v'. For example, in case $n[3].d$ is a boolean variable, we can instantiate v' with 0 and 1, resulting in two rules: $(\rho \wedge n[3].d = 0) \wedge v' = 0$ and $(\rho \wedge n[3].d = 1) \wedge v' = 1$. Consequently, we need to compute two auxiliary invariants for these two new rules.

We construct an abstract model including 2 normal nodes an one abstract node. A reference system of 3 nodes is used to compute auxiliary invariants. TLV generated 9 auxiliary invariants to verify these properties. All the experiments were carried out on a 3.2 GHz Pentium4 PC with 2 GB memory running Linux. The total time used by TLV and Cadence SMV was 17 seconds.

4.2 German 2004 Protocol

German 2004 is a considerable extension of German 2000[11]. The SMV German 2004 model was translated from Murφ model. We assume that the model has only one memory address for simplicity. We construct an abstract model with three normal nodes plus one abstract node, and one of the three normal nodes is regarded as home node.

A total of 49 auxiliary invariants were generated to verify cache coherence property. Cadence SMV reported a counterexample. By analyzing the counterexample, we observed that the variables $invalidate\_list[*]$ and $directory[*]$ of home nodes that characterize the states of abstract node were eliminated after the abstraction procedure, so that all auxiliary invariants generated did not have enough information to prevent the wrong transition of the abstract node. So we added three auxiliary global boolean variables $exclusive\_granted$, $shared\_valid$ and $invalidate\_valid$ to retain the information. This counterexample guided auxiliary variables adding procedure is done by hand. The coherence property is verified successfully in the modified model.

Then we complemented the data path specification and verified data consistency property:

$$\forall i : n[i].state \neq I \rightarrow n[i].data = auxdata$$

TLV generated 51 auxiliary invariants to verify this property. We expect the data consistency to be verified in the modified abstract model. However, a counterexample was reported. Since abstraction may cause false negative, to make sure that this counterexample is a true counterexample, we checked an instance of the protocol with four nodes and a two element data domain in Cadence SMV. The counterexample was also reported. Analyzing both counterexamples we found that the original German 2004 protocol has a bug. The bug is described as follows:

In the protocol a node requiring read access to a given address should sends a read_shared request to the home node. If the directory of the address in home node shows that the cache shared locally, the home node reads data from home cache, updates directory, and sends grant. The bug concerns a rare scenario when the home node is in a state where it has received invalidate acknowledgment from other exclusive state node and has updated its memory data received from the invalidate_ack message and directory entry, but has not sent read grant message to itself yet, so the home cache's state may have not changed from invalid to shared. If a read_shared require arrives at this point, reading from the home cache directly will possibly gets stale data, violating data consistency requirement.

After discussion with the designer of the protocol, the bug is fixed[12]. We verified the modified model again, this time all auxiliary invariants and data consistency property are verified successfully. The total memory TLV uses is about 562 MB. The total time used by TLV and Cadence SMV was about 110 minutes.

4.3 FLASH Protocol

FLASH protocol is much more complicated than German 2004 protocol. Until now we can only verify the cache coherence property of reduced model of FLASH without data path.

The number of nodes in the abstract model is 4 (3 normal nodes + 1 abstract node), as in [7]. We choose $n[0]$ as home node($home = 0$) and $n[3]$ as abstract node. In FLASH, the guard of a rule δ may involve two nodes, for instance $\rho(3)_\delta = n[3].cmd = get \wedge n[3].proc = 2 \wedge n[2].state = E$.

To abstract a rule with such guard, we need to consider four cases: two nodes i, j are both normal nodes, only node

i is normal node, only node j is normal node, and two nodes are both abstract nodes. In the last case, we assume $n[2]$ and $n[3]$ are both abstract nodes. After generating the auxiliary invariants $\varphi(3,0)_\delta = \exists V(1), \exists V(2), \exists V(3) : (reach \wedge \rho(3)_\delta)$ in system $P(4)$, we replace all occurrence number 2 in both $\varphi(3,0)_\delta$ and $\rho(3)_\delta$ with 3 that characterizing the abstract node, obtaining $\varphi(*,0)_\delta$ and $\psi(*,0)_\delta$.

We first verified control coherence:

$$\forall i,j : i \neq j \rightarrow \neg(n[i].state = E \wedge n[j].state = E)$$

We use 56 auxiliary invariants to get the proof to work for cache coherence properties. The total memory TLV uses is about 541 MB. The total time used by TLV and Cadence SMV was about 95 minutes.

When we try to verify data consistency property, we need to compute reachable state set from the full description FLASH reference model. The procedure failed to complete within the limit of 4GB of memory that TLV can deal with. We now have two choices. We could adjust the reference model by eliminating some irrelevant variables manually and choose an optimal BDD variable order that reduce the memory consumption of TLV, or we hack the TLV for 64bit version. Both choices need some time to work.

5 Discussion and Future Work

We have presented a systematic approach to parameterized verification of safety properties of concurrent systems by model-checking. Our approach combines "guard strengthening" and "automatically computed invisible invariants" methods in a novel way. In this approach, a "raw" invariant is first computed in a small reference model (of the same size as the abstract model), using a BDD-based model checker; it is then "cooked", by exploiting symmetry in parameterized systems, to generate auxiliary invariants for guard strengthening. Thus the most human-intelligent demanding task of invariants discovery is partly automated. As case studies we have applied the approach to verify the safety properties of German 2000, German 2004 and FLASH cache coherence protocols. During this process a data consistency error was identified and fixed in German 2004 protocol. These studies show the effectiveness of our approach.

Our approach also has its limitations. In the presence of data part, even though the reference models are quite small (usually consisting of only $3 \sim 4$ nodes), invariant computation can be space-demanding. This is already evident in our case study on the FLASH protocol. Data aspects cannot be ignored if data consistency is to be checked – in fact the data consistency error in German 2004 would not be revealed if all possible data values were collapsed into one. Currently we are looking for effective data abstraction techniques which can be incorporated with parameter abstraction to reduce the size of search space. Another problem is that the generated invariants could be very large (to give an idea, the invariants for German 2004 protocol consists of about 20300 lines). We are considering some filtering heuristics to eliminate irrelevant variables [17], so that space consumption can be reduced and more compact invariants can be generated.

Acknowledgements The authors are grateful to Steven German and Ching-Tsun Chou for providing sources of the case-study protocols, to Sava Krstic for answering several questions on his paper, to Ittai Balaban for his help with TLV system, to Jin Yang and Yongjian Li for helpful discussions and comments.

References

[1] K.R. Apt and D.C. Kozen(1986), Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, **22(6)**, pages 307-309.

[2] T. Arons, A. Pnueli, S. Ruah, J. Xu, L. Zuck(2001), Parameterized Verification with Automatically Computed Inductive Assertions. In *CAV'01*, volume 2102 of LNCS, pages 221-234. Springer-Verlag.

[3] K. Baukus, Y. Lakhnech, and K. Stahl(2002), Parameterized verification of a cache coherence protocol: safety and liveness. In *VMCAI'02*, volume 2294 of LNCS, pages 317-330. Springer-Verlag.

[4] S. Bensalem, Y. Lakhnech, and H. Saidi(1996), Powerful techniques for the automatic generation of invariants. In *CAV'96*, volume 1102 of LNCS, pages 323-335. Springer-Verlag.

[5] J. D. Bingham and A. J. Hu(2005), Empirically efficient verification for a class of infinitestate systems. In *TACAS'05*, volume 3440 of LNCS, pages 77-92. Springer-Verlag.

[6] N.Bjorner, A. Browne, and Z. Manna(1997), Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, **173(1)**, pages 49-87.

[7] C. T. Chou, P. K. Mannava, and S. Park(2004), A simple method for parameterized verification of cache coherence protocols. In *FMCAD'04*, volume 3312 of LNCS, pages 382-398. Springer-Verlag.

[8] E. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith(2003), Counterexample-guided abstraction refinement for smybolic model checking. *Journal of the ACM*, **50**, pages 752-794.

[9] S. Das, D. Dill, and S. Park(1999), Experience with predicate abstraction. In *CAV'99*, volume 1633 of LNCS, pages 160-171. Springer-Verlag.

[10] E. A. Emerson and V. Lahon(2003), Exact and efficient verification of parameterized cache coherence protocols. In*CHARME'03*, volume 2860 of LNCS, pages 247-262. Springer-Verlag.

[11] S. German, G. Janssen(2004), Tutorial on verification of distributed cache memory protocols. In *FM-CAD'04*.

[12] S. German(2006), Personal communication.

[13] S. Krstic(2006), Parametrized system verification with guard strengthening and parameter abstraction. In *AVIS'05*, ENTCS (to appear).

[14] S. K. Lahiri and R. E. Bryant(2004), Indexed predicate discovery for unbounded system verification. In *CAV'04*, volume 3114 of LNCS, pages 135-147. Springer-Verlag.

[15] K. McMillan(2001), Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *CHARME'01*, volume 2144 of LNCS, pages 179-195. Springer-Verlag.

[16] S. Park and D. L. Dill(1996), Verification of FLASH cache coherence protocol by aggregation of distributed transactions. In *SPAA'96*, pages 288-296. ACM Press.

[17] S. Pandav, K. Slind, and G. Gopalakrishnan(2005), Counterexample guided invariant discovery for parameterized cache coherence verification. In *CHARME'05*, volume 3725 of LNCS, pages 317-331. Springer-Verlag.

[18] A. Pnueli and E. Shahar(1996), A platform for combining deductive with algorithmic verification. In CAV'96, volume 1102 of LNCS, pages 184-195. Springer-Verlag.

[19] A. Pnueli, S. Ruah, and L. Zuck(2001), Automatic deductive verification with invisible invariants. In *TACAS'01*, volume 2031 of LNCS, pages 82-97. Springer-Verlag.

[20] A. Tiwari, H. Rueb, H. Saidi, and N. Shankar(2001), A technique for invariant generation. In *TACAS'01*, volume 2031 of LNCS, pages 113-127. Springer-Verlag.