

A Novel Approach to Parameterized Verification of Cache Coherence Protocols

Yongjian Li¹ Kaiqiang Duan¹ Yi Lv¹ Jun Pang²
Shaowei Cai¹

State Key Laboratory of Computer Science, China

Computer Science and Communications, University of Luxembourg, Luxembourg

October 1, 2016

Problem of Parameterized Verification

Consider a protocol P , a property Inv

- $P(N) \models Inv$ for any N
- Not just for a single protocol instance $P(c) \models Inv$, where c is a fixed number
- Model-checking can only check $P(c) \models Inv$
- Our opinion: A theorem proving problem is a complete solution to parameterized verification

- CMP : parameter abstraction and guard strengthening
- Proposed, by McMillan, elaborated by Chou, Mannava, and Park (CMP) , and formalized by Krstic
- construction of an abstract instance which can simulate any protocol instance
- *Not automatic*: human provides auxiliary invariants (non-interference lemmas)

State of Arts–Invisible invariants

- Proposed by Amir Pnueli, Sitvanit Ruah, and Lenore Zuck
- auxiliary invariants are computed from reachable state set in a finite protocol instance $P(c)$
- “Invisible invariants” are raw formulas translated from BDD
- *Problem*: the required reachable state set can’t be enumerated for complex protocol, e.g., the FLASH protocol

Two central and difficult problems

- searching auxiliary invariants is not automatic

Two central and difficult problems

- searching auxiliary invariants is not automatic
- soundness problem: Neither is the theoretical foundation mechanized, nor is there a formal proof for all the things

Our Motivation

- automatically searching auxiliary invariants

Our Motivation

- automatically searching auxiliary invariants
- Formally proving all the things: both the theoretical foundation and case studies
- A formal proof script as a formal verification product

An Overview of Our Approach

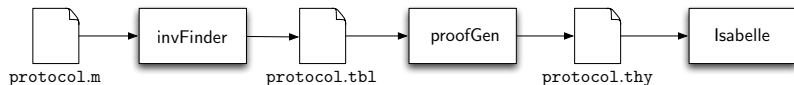


Figure: The workflow of paraVerifier.

Some Explanations

- $\text{paraVerifier} = \text{invFinder} + \text{proofGen}$
- `protoocl.m`: a small reference instance of the protocol in Murphi
- `invFinder` searches auxiliary invariants automatically
- `protocol.tbl`: stores the set of ground invariants and a causal relation table in a concrete form
- `proofGen`: creates an Isabelle proof script `protocol.thy` which models the protocol
- runs Isabelle script to automatically proof-check `protocol.thy`

A protocol is formalized as a pair $(ini, rules)$, where

- ini is a formula to specify initial states
- $rules$ is a set of transition rules. Each rule $r \in rules$ is defined as $g \triangleright S$, where S is a parallel assignment $S = \{x_i := e_i \mid i > 0\}$.

We write $\text{pre } r = g$, and $\text{act } r = S$ if $r = g \triangleright S$.

We define the following relations

- ① $\text{invHoldForRule}_1 s f r \equiv s \models \text{pre } r \longrightarrow s \models \text{preCond } f (\text{act } r)$,
where $\text{preCond } S f = f[x_i := e_i]$, which substitutes each occurrence of x_i by e_i ;
- ② $\text{invHoldForRule}_2 s f r \equiv s \models f \longleftrightarrow s \models \text{preCond } f (\text{act } r)$;
- ③ $\text{invHoldForRule}_3 s f r F \equiv \exists f' \in F \text{ s.t.}$
 $s \models (f' \wedge (\text{pre } r)) \longrightarrow s \models \text{preCond } f (\text{act } r)$;
- ④ $\text{invHoldForRule } s f r F$ represents a disjunction of invHoldForRule_1 , invHoldForRule_2 and invHoldForRule_3 .

Theoretical Foundation - Consistency Relation)

A consistency relation, i.e., consistent $invs$ ini $rules$, that holds between a protocol $(ini, rules)$ and a set of invariants $invs = \{inv_1, \dots, inv_n\}$, is defined as:

- For any invariant $inv \in invs$ and state s , ini implies inv
- For any $inv \in invs$, and $r \in rules$, and any state s , $inv \text{HoldForRule } s \text{ } inv \text{ } r \text{ } invs$.

Theoretical Foundation - Consistent Lemma

$$[|\text{consistent } invs \text{ ini rules}; s \in \text{reachableSet } ini \text{ rules}|] \implies \forall inv \in invs. s \models inv$$

- $\text{reachableSet } ini \text{ rules}$ is the set of reachable states of the protocol.
- $s \models inv$ means inv holds at the state s

Key Algorithm of invFinder

Input: Initially given invariants F , a protocol $\mathcal{P} = \langle I, R \rangle$

Output: A set of tuples which represent causal relations between concrete rules and invariants:

```
1  $A \leftarrow F$ ;  $tuples \leftarrow []$ ;  $newInvs \leftarrow F$ ;  
2 while  $newInvs$  is not empty do  
3    $f \leftarrow newInvs.dequeue$ ;  
4   for  $r \in R$  do  
5      $paras \leftarrow \text{Policy}(r, f)$ ;  
6     for  $para \in paras$  do  
7        $cr \leftarrow \text{apply}(r, para)$ ;  
7        $newInvOpt, rel \leftarrow \text{coreFinder}(cr, f, A)$ ;  
8        $tuples @ [ \langle r, para, f, rel \rangle ]$ ;  
9       if  $newInvOpt \neq \text{NONE}$  then  
10         $newInv \leftarrow \text{get}(newInvOpt)$ ;  
11         $newInvs.enqueue(newInv)$ ;  $A \leftarrow A \cup \{newInv\}$ ;  
12 return  $tuples$ ;
```

- trying to construct a consistency relation that guides the tool invFinder to find auxiliary invariants
- works in a semi-proving and semi-searching way, the result of causal relation can be regarded as key information to construct a concrete proof
- After fetching an unchecked formula f from *newInvs*, for any rule r , $\text{Policy}(r, f)$ generates groups of parameters *paras* according to r and f .
- For each parameter *para* in *paras*, it is applied to instantiate r into a concrete rule cr

- trying to construct a consistency relation that guides the tool invFinder to find auxiliary invariants
- works in a semi-proving and semi-searching way, the result of causal relation can be regarded as key information to construct a concrete proof
- After fetching an unchecked formula f from *newInvs*, for any rule r , $\text{Policy}(r, f)$ generates groups of parameters *paras* according to r and f .
- For each parameter *para* in *paras*, it is applied to instantiate r into a concrete rule cr
- $\text{coreFinder}(cr, f, A)$ is called to check whether a causal relation exists between cr and f .

Intuition behind Policy(r, f)

- Question: How many groups of rule parameters are needed to instantiate r into concrete rules?
- The answer will determine how to compute the enough auxiliary invariants and causal relations between these concrete rules and f for generating a proof.
- For instance, [1], [2], and [3] are three groups which are enough to instantiate r into crit(1), crit(2), and crit(3). But [4] is not needed.
- Here the generated groups of parameters should cover the typical cases which are needed in case analysis in a proof of generalized protocol instance.
- avoiding choosing redundant group of parameters which are similar (or equivalent) to each other.

Formalizing Policy(r, f)

Let m and n be two natural numbers, where $n \leq m$, L and L' are two n -permutations of m ,

- ① $L \sim_m^n L' \equiv (|L| = |L'| = n) \wedge (\forall i. i < |L| \wedge L_{[i]} \leq m - n \longrightarrow L_{[i]} = L'_{[i]}).$
- ② $L \simeq_m^n L' \equiv L \sim_m^n L' \wedge L' \sim_m^n L.$
- ③ $\text{semiP}(m, n, S) \equiv (\forall L \in \text{perms}_m^n \exists L' \in S. L \simeq_m^n L') \wedge (\forall L \in S. \forall L' \in S. L \neq L' \longrightarrow \neg(L \simeq_m^n L')).$
- ④ A set S is called a quotient of the set perms_m^n under the relation \simeq_m^n if $\text{semiP}(m, n, S).$

Consider a parameterized rule r with nR -parameters, and a concrete invariant formula with nI - actual parameters. We only need choose the elements of $\text{semiP}(nR + nI, nR, S)$ to instantiate a rule r .

Formalizing Policy(r, f)

Algorithm 1: Computing a quotient of perms_m^n : *cmpSemiperm*

Input: m, n

Output: A list of permutations L

```
1  $L_0 \leftarrow \text{perms}_m^n; L \leftarrow [];$ 
2 while  $L_0 \neq []$  do
3    $para \leftarrow \text{hd}(L_0); L_0 \leftarrow \text{tl}(L_0);$ 
4   if  $\forall para' \in \text{set}(L). para' \not\sim_m^n para$  then
5      $L \leftarrow L @ [para];$ 
6 return  $L;$ 
```

Policy(r, f) is simply calling *cmpSemiperm*($nR + nI, nR$)

Core Searching Algorithm: *coreFinder*

Input: $r, inv, invs$

Output: A formula option f , a new causal relation rel

```
1  $g \leftarrow$  the guard of  $r$ ,  $S \leftarrow$  the statement of  $r$ ,  $inv' \leftarrow \text{preCond}(inv, S)$ ;  
2 if  $inv = inv'$  then  
3    $relItem \leftarrow (r, inv, invRule_2, -)$ ;  
4   return (NONE,  $relItem$ );  
5 else if  $\text{tautChk}(g \rightarrow inv') = \text{true}$  then  
6    $relItem \leftarrow (r, inv, invRule_1, -)$ ;  
7   return (NONE,  $relItem$ );  
8 else  
9    $candidates \leftarrow \text{subsets}(\text{decompose}(\text{dualNeg}(inv') \wedge g))$ ;  
10   $newInv \leftarrow \text{choose}(chk, candidates)$ ;  
11   $relItem \leftarrow (r, inv, invRule_3, newInv)$ ;  
12  if  $\text{isNew}(newInv, invs)$  then  
13     $newInv \leftarrow \text{normalize}(newInv)$ ;  
14    return (SOME( $newInv$ ),  $relItem$ );  
15  else  
16    return (NONE,  $relItem$ );
```

Core Searching Algorithm: *coreFinder*

After computing the pre-condition inv' (line 1) then:

- If $inv = inv'$, meaning that statement S does not change inv , then no new invariant is created, and new causal relation item marked with tag `invHoldRule2` is recorded between r and inv .
- If `tautChk` verifies that $g \dashv\rightarrow inv'$ is a tautology, then no new invariant is created, and the new causal relation item marked with tag `invHoldRule1` is recorded between r and inv .
- If neither of the above two cases holds, then a new auxiliary invariant $newInv$ will be constructed, which will make the causal relation `invHoldRule3` to hold.

A fragment of output of invFinder

rule	ruleParas	inv	causal relation	f'
..
crit	[1]	mutualInv 1 2	invHoldForRule3	invOnX ₁ 2
crit	[2]	mutualInv 1 2	invHoldForRule3	invOnX ₁ 1
crit	[3]	mutualInv 1 2	invHoldForRule2	
..
crit	[1]	invOnX ₁ 1	invHoldForRule1	-
crit	[2]	invOnX ₁ 1	invHoldForRule1	-

- invariants and causal relations are in concrete form
- we need parameterized form (or symbolic form)

There are two main kinds of generalization in our work:

- generalization of a normalized invariant into a symbolic one.
For instance, $\neg(x \doteq \text{true} \wedge n[1] \doteq C)$ is generalized into $\neg(x \doteq \text{true} \wedge n[i\text{Inv}_1] \doteq C)$.
- The generalization of concrete causal relations into parameterized causal relations, which consists of two phases.
 - Phase I: groups of rule parameters such as $[[1],[2],[3]]$ will be generalized into a list of symbolic formulas such as $[iR_1 = i\text{Inv}_1, iR_1 = i\text{Inv}_2, (iR_1 \neq i\text{Inv}_1) \wedge (iR_1 \neq i\text{Inv}_2)]$
 - Phase II: the formula field accompanied with a relation of kind invHoldRule_3 is also generalized

-From groups of parameters to symbolic formulas

Let LR and LI be two permutations which represent rule parameters and invariant parameters, we define:

- symbolic comparison condition generalized from comparing $LR_{[i]}$ and $LI_{[j]}$: $\text{symbCmp}(LR, LI, i, j) \equiv$

$$\begin{cases} iR_i = i\text{Inv}_j & \text{if } LR_{[i]} = LI_{[j]} \end{cases} \quad (1)$$

$$\begin{cases} iR_i \neq i\text{Inv}_j & \text{otherwise} \end{cases} \quad (2)$$

- symbolic comparison condition generalized from comparing $LR_{[i]}$ and with all $LI_{[j]}$: $\text{symbCase1}(LR, LI, i) \equiv$

$$\begin{cases} \text{symbCmp}(LR, LI, i, j) & \text{if } \exists! j. LR_{[i]} = LI_{[j]} \end{cases} \quad (3)$$

$$\begin{cases} \text{forallForm}(|LI|, pf) & \text{otherwise} \end{cases} \quad (4)$$

where $pf(j) = \text{symbCmp}(LR, LI, i, j)$, and $\exists! j.P$ is a qualifier denoting there exists an unique j s.t. property P ;

- symbolic case generalized from comparing LR with LI :
 $\text{symbCase}(LR, LI) \equiv \text{forallForm}(|LR|, pf)$, where
 $pf(i) = \text{symbCase1}(LR, LI, i)$;

The result of generalizing lines of Table

rule	inv	case	causal relation	f'
r	f	$iR_1 = iInv_1$	invHoldRule3	$invOnXC(iInv_2)$
r	f	$iR_1 = iInv_2$	invHoldRule3	$invOnXC(iInv_1)$
r	f	$(iR_1 \neq iInv_1) \wedge$ $(iR_1 \neq iInv_2)$	invHoldRule2	

Automatic Generation of Isabelle Proof

- Building formal model and properties for a protocol in a theorem prover
 - Building formal model and properties *automatically*
 - Murphi model and computed invariants \longrightarrow Isabelle model

Automatic Generation of Isabelle Proof

- Building formal model and properties for a protocol in a theorem prover
 - Building formal model and properties *automatically*
 - Murphi model and computed invariants \rightarrow Isabelle model
- Proving that properties hold in the formal model
 - Instead of working interactively, *we construct our proof automatically*
 - lines of symbolic causal table \rightarrow a proof doing case analysis

A Fragment of Isabelle model

- Definition of a formula of an invariant:

definition inv_1::"nat \Rightarrow nat \Rightarrow formula" where [simp]:

"inv_1 p_Inv3 p_Inv4 \equiv

(neg (andForm (eqn (IVar (Para (Ident "n") p_Inv4)) (Const C)) (eqn (IVar (Para (Ident "n") p_Inv3)) (Const C))))"

- Definition of a lemma:

lemma critVsinv1:

assumes a1: $\exists iR1. iR1 \leq N \wedge r=crit\ iR1$ and

a2: $\exists ilnv1\ ilnv2. ilnv1 \leq N \wedge ilnv2 \leq N \wedge ilnv1 \neq ilnv2 \wedge$
f=inv1 ilnv1 ilnv2

shows invHoldForRule s f r (invariants N)

A Fragment of Isabelle proof of Lemma critVsinv1

```
1 lemma critVsinv1:
2   assumes a1:  $\exists iR1. iR1 \leq N \wedge r = \text{crit } iR1$  and
3   a2:  $\exists iInv1\ iInv2. iInv1 \leq N \wedge iInv2 \leq N \wedge iInv1 \neq iInv2 \wedge f = \text{inv1 } iInv1\ iInv2$ 
4   shows  $\text{invHoldRule } s\ f\ r\ (\text{invariants } N)$ 
5   proof -
6     from a1 obtain iR1 where a1:  $iR1 \leq N \wedge r = \text{crit } iR1$ 
7     by blast
8     from a2 obtain iInv1 iInv2 where a2:  $iInv1 \leq N$ 
9      $\wedge iInv2 \leq N \wedge iInv1 \neq iInv2 \wedge f = \text{inv1 } iInv1\ iInv2$ 
10    by blast
11    have  $iR1 = iInv1 \vee iR1 = iInv2 \vee (iR1 \neq iInv1 \wedge iR1 \neq iInv2)$ 
12    by auto
13    moreover {
14      assume b1:  $iR1 = iInv1$ 
15      have  $\text{invHoldRule3 } s\ f\ r\ (\text{invariants } N)$ 
16      proof (cut_tac a1 a2 b1, simp,
17        rule_tac x =  $\neg (x = \text{true} \wedge n[iInv2] = C)$  in exI, auto) qed
18      8 then have  $\text{invHoldRule } s\ f\ r\ (\text{invariants } N)$  by auto
19    }
20    moreover {
21      assume b1:  $iR1 = iInv2$ 
22      have  $\text{invHoldRule3 } s\ f\ r\ (\text{invariants } N)$ 
23      proof (cut_tac a1 a2 b1, simp,
24        rule_tac x =  $\neg (x = \text{true} \wedge n[iInv1] = C)$  in exI, auto) qed
25      11 then have  $\text{invHoldRule } s\ f\ r\ (\text{invariants } N)$  by auto
26    }
27    moreover {
28      assume b1:  $(iR1 \neq iInv1 \wedge iR1 \neq iInv2)$ 
29      12 have  $\text{invHoldRule2 } s\ f\ r$ 
30      proof (cut_tac a1 a2 b1, auto) qed
31      14 then have  $\text{invHoldRule } s\ f\ r\ (\text{invariants } N)$  by auto
32    }
33    ultimately show  $\text{invHoldRule } s\ f\ r\ (\text{invariants } N)$  by blast
34  qed
```

Experiments

Protocols	#rules	#invariants	time (sec.)	Memory (MB)
mutualEx	4	5	3.25	7.3
MESI	4	3	2.47	11.5
MOESI	5	3	2.49	13.5
German	13	52	38.67	14
FLASH_nodata	60	152	280	26
FLASH_data	62	162	510	26

Among them, the former three ones are small, German is medium, FLASH is industrial-scale.

Conclusions

- invFinder *generates automatically* invariants
- proofGen *generates automatically* proofs to prove invariants
- Our verification output: a formally readable proof.
- Our verification goal: verifying the protocol in both an automatic and rigorous way