A Novel Approach to Parameterized Verification of Cache Coherence Protocols

Yongjian Li[1] Kaiqiang Duan[1] Yi Lv[1] Jun Pang[2] Shaowei Cai[1]

State Key Laboratory of Computer Science, China

Computer Science and Communications, University of Luxembourg, Luxembourg

September 30, 2016

Problem of Parameterized Verification

Consdier a protocol P, a property Inv

- $P(N) \models Inv$ for any N
- not just for a single protocol instance $P(c) \models Inv$
- Our opinion: parameterized verification is a theorem proving problem

State of Arts–CMP method

- CMP : parameter abstraction and parameter abstraction
- 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
- 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 $P(c)$
- raw formula translated from BDD
- the reachable state set can't be enumerated, e.g., the FLASH protocol

Two central and difficult problems

- searching auxiliary invariants is not automatic
- soundness problem: the theoretical foundation is not mechanized, and there is no a formal proof

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

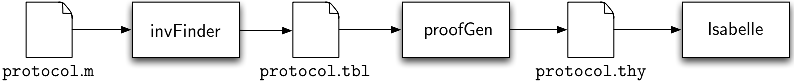

Figure: The workflow of paraVerifier.

Some Explanations

- paraVerifier=invFinder + proofGen
- protoocl.fl: a small reference instance of the protocol
- invFinder searches auxiliary invariants automatically
- protocol.tbl: stores the set of ground invariants and a causal relation table
- proofGen: create an Isabelle proof script protocol.thy which models and verifies the protocol
- run Isabelle script to automatically proof-check protocol.thy

Theoretical Foundation-Protocol

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

- *ini* is an initialization formula; and
- *rules* is a set of transition rules. Each rule $r \in rules$ is defined as $g \triangleright S$, where g is a predicate, and S is a parallel assignment to distinct variables v_i with expressions e_i, where S is a parallel assignment $S = \{x_i := e_i | i > 0\}$.

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

Theoretical Foundation-Causal Relation

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. $\text{invHoldForRule}_3\ s\ f\ r\ F \equiv \exists f' \in F$ 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 $invs = \{inv_1, \ldots, inv_n\}$, is defined as:

- For any invariant $inv \in invs$ and state s, if *ini* is evaluated as true at state s (i.e., formEval $ini\ s = true$), then *inv* is also evaluated as true at the state s.
- For any $inv \in invs$, and $r \in rules$, and any state s, invHoldForRule *inv r invs*.

Theoretical Foundation - Consistent Lemma

For a protocol $(ini, rules)$, we use reachableSet $ini\ rules$ to denote the set of reachable states of the protocol. Given a set of invariants $invs$, we have $[|\text{consistent } invs\ ini\ rules; s \in \text{reachableSet } ini\ rules|] \Longrightarrow \forall inv \in invs.\text{formEval } inv\ s$

Key Algorithm of invFinder

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

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

$A \leftarrow F$; $tuples \leftarrow []$; $newInvs \leftarrow F$;
while *newInvs is not empty* **do**
  $f \leftarrow newInvs.dequeue$;
  for $r \in R$ **do**
    $paras \leftarrow \text{Policy}(r, f)$;
    for $para \in paras$ **do**
      $cr \leftarrow \text{apply}(r, para)$;
      $newInvOpt, rel \leftarrow \text{coreFinder}(cr, f, A)$;
      $tuples \leftarrow tuples@[< r, para, f, rel >]$;
      if $newInvOpt \neq NONE$ **then**
        $newInv \leftarrow \text{get}(newInvOpt)$;
        newInvs.enqueue(newInv); $A \leftarrow A \cup \{newInv\}$;
return *tuples*;

More on invFinder

- 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, 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*, $coreFinder(cr, f, A)$ 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 $n \le m$, L and L' are two n-permutations of m,

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

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

$L_0 \leftarrow \mathrm{perms}_m^n$;$L \leftarrow []$;
while $L_0 \neq []$ **do**
    $para \leftarrow \mathrm{hd}(L_0)$;$L_0 \leftarrow \mathrm{tl}(L_0)$;
    if $\forall para' \in \mathrm{set}(L).para' \not\simeq_m^n para$ **then**
        $L \leftarrow L@[para]$;
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

$g \leftarrow$the guard of r, $S \leftarrow$the statement of r, $inv' \leftarrow$ preCond(inv, S);
if $inv = inv'$ **then**
    $relItem \leftarrow (r, inv, invRule_2, -)$;
    return (NONE, $relItem$);
else if tautChk($g \rightarrow inv'$) $= true$ **then**
    $relItem \leftarrow (r, inv, invRule_1, -)$;
    return ($NONE$, $relItem$);
else
    $candidates \leftarrow$ subsets(decompose(dualNeg(inv') $\wedge$ g));
    $newInv \leftarrow choose(chk, candidates)$;
    $relItem \leftarrow (r, inv, invRule_3, newInv)$;
    if $isNew(newInv, invs)$ **then**
        $newInv \leftarrow normalize(newInv)$;
        return (SOME($newInv$), $relItem$);
    else
        return (NONE, $relItem$);

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 $\texttt{invHoldRule}_2$ is recorded between r and inv.
- If tautChk verifies that $g \dashrightarrow inv'$ is a tautology, then no new invariant is created, and the new causal relation item marked with tag invHoldRule_1 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 invHoldRule_3 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)

Generalization

There are two main kinds of generalization in our work:

- generalization of a normalized invariant into a symbolic one. For instance, $\neg(\mathsf{x} \doteq \mathsf{true} \wedge \mathsf{n}[1] \doteq \mathsf{C})$ is generalized into $\neg(\mathsf{x} \doteq \mathsf{true} \wedge \mathsf{n}[\mathtt{iInv}_1] \doteq \mathsf{C})$.
- 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 $[\mathtt{iR}_1 = \mathtt{iInv}_1, \mathtt{iR}_1 = \mathtt{iInv}_2, (\mathtt{iR}_1 \neq \mathtt{iInv}_1) \wedge (\mathtt{iR}_1 \neq \mathtt{iInv}_2)]$
  - Phase II: the formula field accompanied with a relation of kind $\mathsf{invHoldRule}_3$ is also generalized

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 $LI_{[j]}$: $\text{symbCmp}(LR, LI, i, j) \equiv$

$$\begin{cases} \mathtt{iR_i} = \mathtt{iInv_j} & \text{if } LR_{[i]} = LI_{[j]} \quad (1) \\ \mathtt{iR_i} \neq \mathtt{iInv_j} & \text{otherwise} \quad (2) \end{cases}$$

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

$$\begin{cases} symbCmp(LR, LI, i, j) & \text{if } \exists! j. LR_{[i]} = LI_{[j]} \quad (3) \\ forallForm(|LI|, pf) & \text{otherwise} \quad (4) \end{cases}$$

  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 $pf(i) = \text{symbCaseI}(LR, LI, i)$;

The result of generalizing lines of Table

rule	inv	case	causal relation	f'
r	f	$\texttt{iR}_1 = \texttt{iInv}_1$	invHoldRule3	invOnXC($iInv_2$)
r	f	$\texttt{iR}_1 = \texttt{iInv}_2$	invHoldRule3	invOnXC($iInv_1$)
r	f	$(\texttt{iR}_1 \neq \texttt{iInv}_1) \wedge$ $(\texttt{iR}_1 \neq \texttt{iInv}_2)$	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
- Proving that properties hold in the formal model
  - Instead of working interactively, *we construct our proof automatically*
  - lines of symbolic causal table $\longrightarrow$ a proof doing case analysis

A Fragment of Isabelle model

- Definition of a formula of an invariant:
  definition inv__1::"nat ⇒ nat ⇒ formula" where [simp]:
  "inv__1 p__Inv3 p__Inv4 ≡
  (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: $\exists$ *iInv1 iInv2. iInv1* $\leq N \wedge$ *iInv2* $\leq N \wedge$ *iInv1* $\neq$ *iInv2* $\wedge$
  f=inv1 iInv1 iInv2
  shows invHoldForRule s f r (invariants N)

A Fragment of Isabelle proof of Lemma critVsinv1

```
1lemma critVsinv1:
2 assumes a1:  ∃ iR1.  iR1 ≤ N ∧ r=crit iR1 and
a2:  ∃iInv1 iInv2.  iInv1≤N ∧ iInv2 ≤ N ∧ iInv1≠iInv2 ∧ f=inv1 iInv1 iInv2
3 shows invHoldRule s f r (invariants N)
4 proof -
from a1 obtain iR1 where a1:iR1 ≤ N ∧ r=crit iR1
  by blast
from a2 obtain iInv1 iInv2 where a2:  iInv1 ≤ N
∧ iInv2 ≤ N ∧ iInv1 ≠ iInv2 ∧ f=inv1 iInv1 iInv2
  by blast
5 have iR1=iInv1 ∨ iR1=iInv2 ∨ (iR1 ≠ iInv1 ∧ iR1 ≠ iInv2)
by auto
6 moreover{assume b1:iR1=iInv1
7   have invHoldRule3 s f r (invariants N)
    proof(cut_tac a1 a2 b1, simp,
rule_tac x=¬ (x=true ∧ n[iInv2]=C) in exI,auto)qed
8   then have invHoldRule s f r (invariants N) by auto}
9 moreover{assume b1:iR1=iInv2
10   have invHoldRule3 s f r (invariants N)
    proof(cut_tac a1 a2 b1, simp,
rule_tac x=¬ (x=true ∧ n[iInv1]=C in exI,auto)qed
11   then have invHoldRule s f r (invariants N) by auto}
12 moreover{assume b1:(iR1 ≠ iInv1 ∧ iR1 ≠ iInv2)
13   have invHoldRule2 s f r
    proof(cut_tac a1 a2 b1, auto) qed
14   then have invHoldRule s f r (invariants N) by auto}
15ultimately show invHoldRule s f r (invariants N) by blast
16qed
```

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