paraVerifier: An Automatic Framework for Proving Parameterized Cache Coherence Protocols

Yongjian Li[1,3] Jun Pang[2] Yi Lv[1] Dongrui Fan[4]
Shen Cao[1] Kaiqiang Duan[1]

State Key Laboratory of Computer Science, China

Computer Science and Communications, University of Luxembourg, Luxembourg

College of Information Engineering, Capital Normal University, Beijing, China

Institute of Computing Technology, Chinese Academy of Sciences, China

2016 年 9 月 30 日

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 opnion: a theorem proving problem becasue we cann't enumerate all protocol instance $P(N)$

State of Arts

- 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-non-interference lemmas)

State of Arts

- invisible invariants,
- auxiliary invariants are computed from reachable state set in a finite protocol instance $P(c)$
- raw formula translated from BDD
- the reachable state set cann't be enumerated, e.g., the FLASH protocol

Two central and diffcult problems

- searching auxilairy invariants is not automatical
- soundness problem: the theoretical foundation is not mechanized, and a gap between existing approaches and a formal proof

Our Motivation

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

paraVerifier: An Automatic Framework for Parameterized Verification

2015 年 12 月 29 日

Some Explanations

- paraVerifier=invFinder + proofGen
- protoocl.fl: a small cache coherence protocol instance
- invFinder searches auxiliary invariants automatically
- Some advanced theories: Kepler guess, Four-coloured problems
- 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 to automatically proof-check protocol.thy

Theoretical Foundation-Protocol

A cache coherence protocol is formalized as a pair $(ini, rules)$, where (1) ini is an initialization formula; and (2) $rules$ 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. We write pre $r = g$, and act $r = S$ if $r = g \triangleright S$, where S is a parallel assignment $S = \{x_i := e_i | i > 0\}$.

Theoretical Foundation-Causal Relation

Definition

We define the following relations

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

Theoretical Foundation - Consistency Relation)

Definition

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 $r \in rules$, invHoldForRule $inv\ r\ invs$.

Theoretical Foundation - Consistent Lemma

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

```
let findInvsFromRule chk choose tautChk isNew rule inv newInvs invs casRel=
val (g ▷ S)=rule in
let inv'=preCond S inv in
if inv=inv' then
  let relItem=(rule, inv,invHoldForRule2 inv r) in
  (newInvs, relItem:casRel)
else if tautChk (g ⟶inv') then
  let relItem=(rule, inv, invHoldForRule1 inv r) in
  (newInvs, relItem:casRel)
else let candidates= subsets (decompose ((dualNeg inv') ∧ g )) in
   let newinv = choose chk candidates in
   let relItem=(rule, inv, invHoldForRule3 inv newInv ) in
   if ((isNew newInv (newInvs@invs)) then
   (newInvs@[normalize newInv], relItem#casRel)
   else (newInvs, relItem#casRel)
else error ``no new invariant";
```

More on invFinder

- trying to construct a consistency relation that guides the tool invFinder to find auxiliary invariants
- using an oracles that checks whether a ground formula is an invariant in the small reference model
- Searching not only auxiliary invariants but also causal relations
- protocol.tbl: storing the searching result

A fragment of protocol.tbl

Table: 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)