paraverifier

Overview

- paraVerifier is composed of two parts: an invariant finder ***invFinder*** and a proof generator ***proofGen***.
- Given a protocol ***P*** and a property ***inv***, ***invFinder*** tries to find useful *auxiliary invariants* and *causal relations* which are capable of proving ***inv***.
- ***proofGen*** generalizes the auxiliary invariants and causal relations into a parameterized form, which are then used to construct a completely parameterized formal proof in a theorem prover (e.g., Isabelle) to model ***P*** and to prove the property ***inv***.
- Problem
  - Input: a parameterized (symbolic) protocol ***P(N)*** where ***N*** is arbitrary, an invariant property ***Inv***
  - Task: $P(N) \models Inv$ for any ***N***

Two Central Problems

- automatically searching auxiliary invariants
- soundness problem: formally proving all the steps

Preliminary

- A protocol is formalized as a pair (***ini, rules***), where
  - ***ini*** is a formula to specify initial states
  - ***rules*** is a set of guarded commands. Each ***rule r*** $\in$ ***rules*** is defined as ***g*** → ***S***, where ***g*** is a predicate specifying the guard, and the update ***S*** is a parallel assignment to distinct variables
- Inductive Invariant
  - Definition: Let ***P := (ini, rules)*** be a protocol. A formula ***inv*** is an inductive invariant of ***P*** if

    — the initial state satisfies the formula: $\models I \rightarrow inv$, and

    — transitions preserve the formula: for each $r \in R$, we have $\models inv \wedge \text{guard}(r) \rightarrow \text{WP}(\text{action}(r), inv)$.

  - Proposition: Assume given a protocol ***P=(I,R)*** and an requirement(invariant) ***req***, ***P*** satisfies ***req*** if there exists an inductive invariant ***inv*** of P such that $\models$***inv→req***

Main Ideas

1. concretize the requirement ***req***
2. concretize the guarded commands ***r*** with regard to ***req*** above, *i.e*. we concretize each pair (***req, r***) and record all the necessary actual parameter indics for each ***r*** seperately.
3. check whether the concretized requirement ***req*** is an inductive invariant and if necessary find a strengthening ***aux*** that will be used as candicate auxiliary invariant
4. generalize the strengthend requirement ***aux*** back to a prameterized one

Example: Mutual Exclusion Protocol

N symmetric processors, behaviour of processor i is described by:

rules

- $try(i) := a[i] = I \rightarrow a[i]' = T$
- $crit(i) := (a[i] = T \wedge x = true \rightarrow a[i]' = C \wedge x' = false)$
- $exit(i) := a[i] = C \rightarrow a[i]' = E$
- $idle(i) := a[i] = E \rightarrow a[i]' = I \wedge x' = true$

Initial states: $x = true$ and $a[i] = I$ for all i

requirement

Invariant property (where we assume parameters are pairwise disjoint):
$\neg(a[i] = C \wedge a[j] = C)$

Concretize the requirement

1. Assume req contains n parameters, $i_1, i_2, \ldots, i_n$, pairwise different.
2. Fix an injective mapping m by: $m(i_1) = 1, \ldots, m(i_n) = n$,from occured parameters to natural numbers starting from 1
3. The concretized property is denoted by $m(f)$
4. In mutual exclusion protocol, $req \equiv \neg(a[i] = C \wedge a[j] = C),\ n = 2$
5. concretized requirement $creq := \neg(a[1] = C \wedge a[2] = C)$
6. $m(i) = 1, m(j) = 2$

   the mapping m will be used later in the concretization

Concretize the guarded command

Rule with one parameter

1. In mutual exclusion protocol, $req \equiv \neg(a[i] = C \wedge a[j] = C)$, $n = 2$
2. concretized requirement $creq := \neg(a[1] = C \wedge a[2] = C)$
3. consider rule $try(k) \equiv a[k] = I \rightarrow a[k] := T$ with one parameter k
4. due to symmetries of the protocol, the considered cases are
   - $k = i$
   - $k = j$
   - $k \neq i \wedge k \neq j$
5. for the mapping m, we can concretize $try(k)$ using $k = 1, 2, 3$ in this case

Concretize the guarded command

Rule with two parameters

1. In mutual exclusion protocol, $\boldsymbol{req} \equiv \neg(a[i] = C \wedge a[j] = C),\ n = 2$
2. concretized requirement $\boldsymbol{creq} := \neg(a[1] = C \wedge a[2] = C)$
3. consider rule has disjoint parameters $iR_1 \neq\ iR2$
4. due to symmetries of the protocol, the considered cases are

$$\bullet\ as\ previous\ case\ iR_1\ can\ be\ 1, 2, 3$$

$$\bullet\ as\ iR_1 \neq iR_2,\ we\ have\ two\ choices\ for\ iR_2\ with\ a\ fixed\ iR_1$$

$$\bullet\ in\ addition,\ 3\ is\ special\ as\ it\ abstracts\ other\ cases$$

$$\bullet\ thus\ we\ have\ 3 \times 2 + 1\ cases,\ namely$$

$$(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2), (3, 4)$$

$$\bullet\ each\ case\ corresponds\ to\ a\ mapping\ m^{'}, for\ instance\ for\ (2, 3)\ we\ have$$

$$m^{'}(iR1) = 2\ and\ m^{'}(iR2) = 3$$

Concretize the guarded command

Generally. (***f/req/inv***)

1. Assume f contains n parameters, say $i_1, i_2, \ldots, i_n$.
2. The concretized property $m(f)$ is obtained by an injective mapping m with $m(i_j) = j$
3. The rule $r := g \to S$ has n' parameters, say $\mathtt{iR}_1, \mathtt{iR}_2, \ldots, \mathtt{iR}_{n'}$, pairwise different
4. The number of cases of the concretized rules to consider:
    1. for $n' = 1$ it is $n + 1$
    2. for $n' = 2$ it is $(\prod_{k=n+2-n'}^{k=n+1} k) + 1$ where 1 is for the case where both $\mathtt{iR}_1$ and $\mathtt{iR}_2$ fall out of the bound
    3. for $n' = 3$ it is $(\prod_{k=n+2-n'}^{k=n+1} k) + 1 + 3n$ where 1 is for the case where all $\mathtt{iR}_1$, $\mathtt{iR}_2$ and $\mathtt{iR}_3$ fall out of the bound, and $3n$ for the case where two of them fall out of the bound
    4. each case corresponds to a mapping m' in the obvious way, and we denote by $m'(r)$ the concretized rule

Finding Inductive Invariants

- "for each $r \in R$, we have $\models inv \wedge guard(r) \rightarrow WP(action(r), inv)$ "
- We distinguish the following three cases:
  1. $\models guard(r) \rightarrow WP(action(r), inv)$: precondition ***inv*** not needed
  2. $\models inv \rightarrow WP(action(r), inv)$: guard not needed
  3. $\models inv \wedge guard(r) \wedge inv' \rightarrow WP(action(r), inv)$:strengthening ***inv'*** needed
- Key Point
  - find all the invariants and they relations (such as ***inv' -> inv*** aboved) in case 3

Finding Inductive Invariants

- "for each $\boldsymbol{r} \in \boldsymbol{R}$, we have $\models \boldsymbol{inv} \wedge \boldsymbol{guard(r)} \rightarrow \boldsymbol{WP(action(r), inv)}$ "
- We distinguish the following three cases:
  1. $\models \boldsymbol{guard(r)} \rightarrow \boldsymbol{WP(action(r), inv)}$: precondition ***inv*** not needed
  2. $\models \boldsymbol{inv} \rightarrow \boldsymbol{WP(action(r), inv)}$: guard not needed
  3. $\models \boldsymbol{inv} \wedge \boldsymbol{guard(r)} \wedge \boldsymbol{inv'} \rightarrow \boldsymbol{WP(action(r), inv)}$:strengthening ***inv'*** needed
- For concretized ***m(inv)*** and ***m'(r)***, which is ***m'(g)→m'(S)***, we use ***m(f), m'(g), m'(S)*** accordingly

1. $\models \boldsymbol{m'(g)} \rightarrow \boldsymbol{WP(m'(S), inv)}$: precondition ***m(inv)*** not needed
2. $\models \boldsymbol{m(inv)} \rightarrow \boldsymbol{WP(m'(S), inv)}$: guard not needed
3. $\models \boldsymbol{m(inv)} \wedge \boldsymbol{m'(g)} \wedge \boldsymbol{inv'} \rightarrow \boldsymbol{WP(m'(S), m(inv))}$:strengthening ***inv'*** needed

Finding Inductive Invariants

- Case3:
  - construct a strengthening
  - an obvious choice is ***m'(g)→WP(m'(S), m(inv))***, such that (the logic formula in case holds)
  - the choice above is complex, rewrite it into a form $\neg(\bigwedge_i f_i)$
  - consider subformulas from its simplified form
  - exploit a model checker(NuSMV, Murphi) to explore an auxilary invariant

Example: mutual exclusion protocol

1. In mutual exclusion protocol, $req \equiv \neg(a[i] = C \wedge a[j] = C),\ n = 2$
2. concretized requirement $creq := \neg(a[1] = C \wedge a[2] = C)$, with $m(i) = 1, m(j) = 2$
3. consider the concretized rule $crit(1) := (a[1] = T \wedge x = true \rightarrow a[1]' = C \wedge x' = false)$
4. $WP(m(req), m'(S)) = \neg(C = C \wedge a[2] = C)$
5. invariant choice

$$
\begin{aligned}
&\neg g \vee WP(m(req), crit(1)) \\
&= \neg(a[1] = T \wedge x = true) \vee \neg(C = C \wedge a[2] = C) \\
&\equiv \neg(a[1] = T \wedge x = true \wedge C = C \wedge a[2] = C)
\end{aligned}
$$

6. our model checker returns $invOnXC(2) = \neg(x = true \wedge a[2] = C)$

Generalization

Recall $\texttt{iR}_1, \texttt{iR}_2$ parameters used in describing the rule, and i_1, i_2 parameters used in describing the invariant formulas.

- inductive strengthening f_i are generalized: here both $invOnXC(1)$ and $invOnXC(2)$ are generalized to the same formula $invOnXC(i_1)$
- The generalization of concrete causal relations into parameterized causal relations is done in two phases.
  - Phase I: relation of the mappings m and m' used before. For instance $m(i_1) = 1, m(i_2) = 2$ and $m'(\texttt{iR}_1) = 2$ gives the symbolic formulas matching the corresponding case as $\texttt{iR}_1 = i_2$, and $m'(\texttt{iR}_1) = 3$ gives $(\texttt{iR}_1 \neq i_1) \wedge (\texttt{iR}_1 \neq i_2)$.
  - Phase II: the formula field accompanied with a corresponding case labelling is also generalized in the obvious way.

Generalization

Concretize i_1, i_2	rule	case	Case	f'
mutualInv 1 2	crit($\mathtt{iR}_1$)	$\mathtt{iR}_1 = i_1$	3	invOnXC(i_2)
mutualInv 1 2	crit($\mathtt{iR}_1$)	$\mathtt{iR}_1 = i_2$	3	invOnXC(i_1)
mutualInv 1 2	crit($\mathtt{iR}_1$)	$(\mathtt{iR}_1 \neq i_1) \wedge (\mathtt{iR}_1 \neq i_2)$	2	

Implementation

```
(*
    newInvs : new Invariants(initialized to empty set)
    invs    : invariants(initialized to empty set)
    casRel  : all the causal relations constructed up to now
*)let findInvsFromRule chk choose tautchk isNew pRule paras inv newInvs invs casRel
=
   let rule = ruleApp pRule paras in
   let val (g, S) = rule in
   let inv'= preCond S inv in (* computing the pre-condition *)
      if inv = inv'  (* case analysis on inv *)
         then let relItem = (pRule, paras, inv, invHoldForRule2 inv r) in
              (newInvs, relItem::casRel)
      else if tautchk(g -> inv')
         then  let relItem = (pRule, paras, inv, invHoldForRule1, inv, r) in
               (newInvs, relItem::casRel)
      else
         (* choose a new auxiliary invariant from the conjuncts of g & !inv'*)
         (* call the function chk to guarantee newInv is an invariant of the
reference model*)
         let newInv = choose chk inv' g in
         let relItem = (pRule, paras, inv, invHoldForRule3 inv newInv) in
         (* isNew is used to check whether the invariant is new *)
         (*the meaning of the word "new" is modulo to the symmetry relation*)
         if (isNew newInv (newInvs@invs))
            then (newInvs@[normalize newInv], relItem::casRel)
         else
            error "no new invariant"
```

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