Automatically Finding Auxiliary Parameterized Invariants

We will now consider a method for finding inductive assertions for BDS’s.

Automatic Generation of Auxiliary Invariants

Goal. Compute Auxiliary Inductive Assertion of the form \( \forall i : \psi(i) \)

1. Let \( \text{reach} := \Theta \rho^* \) be the assertion characterizing all the reachable states of system \( S(N_0) \), for \( N_0 = H + 3 \).

E.g. (MUTEX): Since \( H = 1, N_0 = 4 \).

\[
\text{reach} := (x + \sum_{i=1}^{4} (\pi[i] \in \{2,3\})) = 1
\]

2. Let \( \psi_1 \) be the assertion obtained from \( \text{reach} \) by projecting away all the references to variables subscripted by indices other than 1.

E.g.: (MUTEX) \( \psi_1 : \pi[1] \in \{2,3\} \rightarrow (x = 0) \).

3. Let \( \psi(i) \) be the assertion obtained from \( \psi_1 \) by generalizing 1 into \( i \). The candidate for inductive assertion is \( \forall i : \psi(i) \).

E.g.: (MUTEX) \( \psi(i) : \pi[i] \in \{2,3\} \rightarrow (x = 0) \).

Unfortunately, \( \forall i : \psi(i) \) is not inductive over MUTEX (2) \( \implies \) algorithm is not guaranteed to produce inductive assertions.

Example: MUTEX with 1-Index Assertion

In file mutex5_inv1.pf we place:

Func abs(reach,f,t);
   Local trans := (next(y) = y) \&
   (next(P[t].loc) = P[f].loc);
   Return succ(trans,reach);
End -- Func abs(reach,f,t);

To compute_invis;
   Let reach := reachable(1);
   Let phi := 1;
   For (i in 1...N)
      Let phi := phi \& abs(reach,1,i);
   End -- For (i in 1...N)
End -- compute_invis;

Compute Auxiliary Assertion of the form \( \varphi : \forall i : \psi(i) \)

1. Let \( \text{reach} := \Theta \rho^* \) be the assertion characterizing all the reachable states of system \( S(N_0) \).

2. Let \( \rho_g \) be the transition (abstraction) relation, which contains for each finite-domain variable \( x \), the conjunct \( x' = x \).

3. For each \( i \in 1..N_0 \), let \( \rho_{1...i} \) be the transition relation which contains, for each index variable \( k : 1..N \) the conjunct \( (k' = i) \equiv (k = 1) \), and for each array variable \( y \) : array[1..N] of boolean the conjunct \( y[i] = y[1] \).

4. Let \( \psi(i) = \text{reach} \rho_g (\rho_{1...i}) \) be the assertion obtained from \( \text{reach} \) by preserving all the global variables, and porting all the properties of index 1 to index \( i \).

We take \( \varphi : \bigwedge_{i=1}^{N_0} \psi(i) \)

Since assertion \( \varphi \) is computed internally and immediately consumed, the user never gets to see it. This is why we refer to this method as Verification by invisible invariants.
Applying the Algorithm to MUTEX

Consider MUTEX with $N_0 = 3$. 

$reach: \sum_{i=1}^{4} (\pi[i] \in \{2,3\}) + x = 1$

$\psi_{1,2}: \pi[1] \in \{2,3\} \land \pi[2] \in \{2,3\} \land \pi[1] \in \{0,1\}$

$\psi(i,j): \pi[i] \in \{2,3\} \land \pi[j] \in \{2,3\} \land \pi[i] \in \{0,1\}$

$\forall i \neq j: \psi(i,j)$ is an inductive assertion over MUTEX (5), and, therefore, over all MUTEX ($N$). It also implies the property of mutual exclusion:

$\forall i \neq j: (\pi[i] = 2 \land \pi[j] = 2)$

To calc_exc;
Let exc := 1;
For (i in 1...N)
  For (j in i+1...N)
    Let exc := exc & ((P[i].loc not in 3) | (P[j].loc not in 3));
End -- For (j in i+1...N)
End -- For (i in 1...N)
End -- calc_exc;

calc_exc;
compute_invis;
Print "\n Check mutual exclusion\n";
Call inv(exc,phi,1);

Computed assertion fails to be inductive (Premise 2 fails).

Compute Auxiliary Assertion of the form $\forall i \neq j: \psi(i,j)$

1. Let $reach := \Theta \rho^*$ be the assertion characterizing all the reachable states of system $S(N_0)$.
2. Let $\rho_0$ be the transition (abstraction) relation, which contains for each finite-domain variable $x$, the conjunct $x' = x$.
3. For each $i \in 1..N_0$, let $\rho_{[i \rightarrow i]}$ be the transition relation which contains, for each index variable $k: 1..N$ the conjunct $(k' = i) \equiv (k = 1)$, and for each array variable $y: array[1..N]$ of boolean the conjunct $y'[i] = y[1]$. Similarly define $\rho_{[2 \rightarrow j]}$
4. Let $\psi(i,j) = reach \diamond (\rho_{[i \rightarrow i]} \land \rho_{[2 \rightarrow j]})$ be the assertion obtained from $reach$ by preserving all the global variables, and porting all the properties of indices 1, 2 to index $i, j$, respectively.

We take $
\varphi: \bigwedge_{i=1}^{N_0} \bigwedge_{j=i+1}^{N_0} \psi(i,j)
$

In TLV

In file mutex5_inv2.pf we modify function $abs$ and $compute_invis$ as follows:

```
Func abs(reach,f1,f2,t1,t2);
  Local trans := (next(y) = y) &
                    (next(P[t1].loc) = P[f1].loc) &
                    (next(P[t2].loc) = P[f2].loc);
  Return succ(trans,reach);
End -- Func abs(reach,f1,f2,t1,t2);
```

To compute_invis;
Let reach := reachable(1);
Let phi := 1;
For (i in 1..N)
  For (j in i+1..N)
    Let phi := phi & abs(reach,1,2,i,j);
End -- For (j in i+1..N)
End -- For (i in 1..N)
End -- compute_invis;
Lecture 10: Finding Parameterized Invariants

Example: Program Arbiter

Consider the following program \textsc{Arbiter}:

\[ p : \forall i \neq j : \neg (at^{3}[i] \land at^{3}[j]) \]

Searching for a \( \psi(i) \) inductive assertion, we obtained the calculated invariant

\[ \varphi : \forall i : \pi[i] \in \{3, 4\} \iff (x = 0 \land last = i) \]

The candidate assertion \( \varphi \) is inductive and also implies the property of mutual exclusion:

\[ p : \forall i \neq j : \neg (\pi[i] = 3 \land \pi[j] = 3) \]

Example: Program Arbiter

Consider the following program \textsc{arbiter}:

\[ p : \forall i \neq j : \neg (at_{\ell_3}[i] \land at_{\ell_3}[j]) \]

In TLV

In file \texttt{mutex.mod1.pf} we place:

\[
\begin{array}{l}
\text{Func abs(reach,f,t);} \\
\quad \text{Local trans := (next(y) = y) \&} \\
\quad \quad ((\text{next(last)}\text{=}t) = (\text{last}\text{=}f)) \& \\
\quad \quad (\text{next(P[f].loc)} = \text{P[f].loc}); \\
\quad \text{Return succ(trans,reach);} \\
\text{End -- Func abs(reach,f,t);} \\
\end{array}
\]

This time the auxiliary assertion is inductive.
MODULE MA(k,N,r,g)
DEFINE rk := |
    for (i=1; i <= N; i = i+1)
{ i=k ? r[i] : 0; }
VAR loc : 0..4;
ASSIGN init(loc) := 0; init(k) := 1;
next(loc) := case
    loc=0 & rk : 1;
    loc=0 : 4;
    loc in {1,3,4} : (loc+1) mod 5;
    loc=2 & !rk : 3;
    1 : loc;
esac;
next(k) := loc=4 ? (k mod N)+1 : k;
for (i=1; i <= N; i = i+1)
{ next(g[i]) := case
    i != k : g[i];
    loc=1 : 1;
    loc=3 : 0;
    1 : g[i];
esac; }
JUSTICE
loc != 0, loc != 1, loc != 3, loc != 4, !(loc=2 & !rk)

Analysis of Reactive Systems, NYU, Spring, 2006

Lecture 10: Finding Parameterized Invariants

Applying the Invisible Invariants Method

In file arb-inv.pf, we place:

Func abs(reach,f1,t1);
    Local trans := ((next(k)=t1) <-> (k=f1)) &
    (next(Arb.loc) = Arb.loc) &
    (next(g[t1]) = g[f1]) &
    (next(r[t1]) = r[f1]) &
    (next(Cl[t1].loc) = Cl[f1].loc);
    Return succ(trans,reach);
End -- Func abs(reach,f1,t1);

To compute_invis;
    Let reach := reachable(1);
    Let phi := 1;
    For (i in 1...N)
    { Let phi := phi & abs(reach,1,i);
        End -- For (i in 1...N)
    }
    End -- compute_invis;

MODULE MC(r,g)
VAR loc : 0..5;
ASSIGN init(loc) := 0; init(r) := 0; init(g) := 0;
next(loc) := case
    loc in {1,3,4} : (loc+1) mod 6;
    loc=0 : (0,1);
    loc=2 & g : 3;
    loc=5 & !g : 0;
    1 : loc;
esac;
next(r) := case
    loc=1 : 1;
    loc=4 : 0;
    1 : r;
esac;
JUSTICE
loc != 1, !(loc=2 & g), loc !=3, loc !=4, !(loc=5 & !g)

Analysis of Reactive Systems, NYU, Spring, 2006

Lecture 10: Finding Parameterized Invariants

To compute_prop;
    Let exc := 1;
    For (i in 1...N)
    { For (j in i+1...N)
        Let exc := exc & !(Cl[i].loc=3 & Cl[j].loc=3);
        End -- For (j in i+1...N)
    }
    End -- For (i in 1...N)
    End -- compute_prop;
    compute_invis;
    compute_prop;
    Print "\n    Check mutual exclusion\n";
    Call inv(exc,phi,1);