Automatic Generation of Auxiliary Parameterized Invariants

Wett now consider a method for finding inductive assertions for BDS's. This leads to the verification method of invisible invariants.

Since assertion $\phi$ 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.

We take $\phi$ to be the assertion $\forall i \in \mathbb{N} : \phi(i)$. We denote the $i$-th reachable state of the system $S$ by $\text{reach}_i$.

**Example:** MUTEX with 1-index assertion

```
Example: MUTEX with 1-index Assumption

Goal: Compute Auxiliary Inductive Assertion of the Form (i\langle\phi \rangle)
```

To compute $\phi$:

1. Let $\text{reach}_1 := \text{reachable}(1)$
2. Let $\phi := 1$
3. For each $i \in 1..N$:
   - Let $\phi := \phi \land \text{abs}(\text{reach}_1,i)$
4. Let $\phi := (i\langle\phi \rangle) \land \bigvee_{0}^{N} \phi(i)$

We denote this as $\text{reach}^{\phi}$.

We now consider a method for finding inductive assertions for BDS's. This leads to the verification method of invisible invariants.

**Automatic Generation of Auxiliary Parameterized Invariants**
Lecture 11: Automatic Generation of Invariants

A. Pnueli

File Continued

1. Let exc := 1;
2. For (j in 1...N)
   3. Let exc := exc & ((P[i].loc != 3) | (P[j].loc != 3));
4. End--For (j in 1...N)
5. End--For (i in 1...N)
6. End--calc_exc;
7. Compute_invis; Print "Check mutual exclusion"
8. Call inv(exc, phi, 1);

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

Deductive Verification of Reactive Systems, NYU, Fall, 2007

276

Lecture 11: Automatic Generation of Invariants

A. Pnueli

Compute Auxiliary Assertion of the form:

\[ \forall i \neq j : (\Gamma_1)^i \cup \Gamma_2 \neq (\Gamma_1)^j \]

MUTEX (N): It also implies the property of mutual exclusion:

\[ \forall i \neq j : (\Gamma_1)^i \cup \Gamma_2 \neq (\Gamma_1)^j \]

Applying the Algorithm to MUTEX

Consider MUTEX with N = 3:

\[ \text{reach} = \text{reachable}(1) \]

 fury 1: \[ \text{reach} = \text{reachable}(1) \]

In the above, \( \text{reach} \) is the assertion characterizing all reachable states.

We take:

\[ \text{reach}^+(\text{reach} \cup \text{reach}) \]

\[ \text{reach} = \text{reachable}(1) \]

\[ \text{reach}^+ = \text{reachable}(1) \]

\[ \text{reach} \cap \text{reach}^+ = \text{reachable}(1) \]

\[ \text{reach} \cup \text{reach}^+ = \text{reachable}(1) \]

\[ \text{reach}^+ = \text{reachable}(1) \]

\[ \text{reach} = \text{reachable}(1) \]

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

Deductive Verification of Reactive Systems, NYU, Fall, 2007

279
Example: A Modified Mutex

In TLV

In the mutex-modified TLA we place:

```
VARIABLE

loop forever do


1: Non-Critical

2: i := i + 1

3: Critical

4: i := i + 1

5: release x

end
```

Verifying Arbiter by the Invisible Invariants Method

Consider the following program ARBITER:

```
r; g: array[1..N] of boolean

VAR

k: 1..N

module

loop forever do

1: Non-Critical

2: r[i] := 1

3: await g[i]

4: Critical

5: r[i] := 0

6: await g[i]

end
```

for which we wish to prove

```
8 i, j : [i < j : r[i] & r[j] = 0]
```

This time the auxiliary assertion is inductive.

```
end -- Func abs(reach, f, r, t)
```


deductive verification of reactive systems, nyu, fall, 2007

Example: Program Arbiter

Verifying Arbiter by the Invisible Invariants Method

Consider the following program ARBITER:

```
r; g: array[1..N] of boolean

VAR

k: 1..N

module

loop forever do

1: Non-Critical

2: r[i] := 1

3: await g[i]

4: Critical

5: r[i] := 0

6: await g[i]

end
```

for which we wish to prove

```
8 i, j : [i < j : r[i] & r[j] = 0]
```

This time the auxiliary assertion is inductive.

```
end -- Func abs(reach, f, r, t)
```

In the mutex-modified TLA we place:

```
VARIABLE

loop forever do

1: Non-Critical

2: i := i + 1

3: Critical

4: i := i + 1

5: release x

end
```

Verifying Arbiter by the Invisible Invariants Method

Consider the following program ARBITER:

```
r; g: array[1..N] of boolean

VAR

k: 1..N

module

loop forever do

1: Non-Critical

2: r[i] := 1

3: await g[i]

4: Critical

5: r[i] := 0

6: await g[i]

end
```

for which we wish to prove

```
8 i, j : [i < j : r[i] & r[j] = 0]
```

This time the auxiliary assertion is inductive.

```
end -- Func abs(reach, f, r, t)
```

In the mutex-modified TLA we place:

```
VARIABLE

loop forever do

1: Non-Critical

2: i := i + 1

3: Critical

4: i := i + 1

5: release x

end
```

Verifying Arbiter by the Invisible Invariants Method

Consider the following program ARBITER:

```
r; g: array[1..N] of boolean

VAR

k: 1..N

module

loop forever do

1: Non-Critical

2: r[i] := 1

3: await g[i]

4: Critical

5: r[i] := 0

6: await g[i]

end
```

for which we wish to prove

```
8 i, j : [i < j : r[i] & r[j] = 0]
```

This time the auxiliary assertion is inductive.

```
end -- Func abs(reach, f, r, t)
```

In the mutex-modified TLA we place:
Lecture 11: Automatic Generation of Invariants

A. Pnueli

```
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)
```

Deductive Verification of Reactive Systems, NYU, Fall, 2007

284

Lecture 11: Automatic Generation of Invariants

```
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;
```

Deductive Verification of Reactive Systems, NYU, Fall, 2007

285

Lecture 11: Automatic Generation of Invariants

```
Applying the Invisible Invariants Method,
```

```
Inle_arb-inv.pf
```

```
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);
Returnsucc(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;
```

```
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
```

```
Check mutual exclusion
```

Deductive Verification of Reactive Systems, NYU, Fall, 2007

286

Lecture 11: Automatic Generation of Invariants

```
end -- compute-truths!
end -- for t in 1...N
end -- for k in 1...N
end -- for (t in 1...N)
end -- for (k in 1...N)
```

```
end -- Func abs(reach, f1, t1);
```

Deductive Verification of Reactive Systems, NYU, Fall, 2007

287