Verification of Invariance Properties

We may use the following basic invariance rule to prove the invariance of assertion p. That is, establish that the formula $\square\, p$, for an assertion p is $\mathcal{D}$-valid.

Rule BINV

$$\frac{\begin{array}{ll}\text{I1.} & \Theta \rightarrow p \\ \text{I2.} & p \wedge \rho \rightarrow p'\end{array}}{\square\, p}$$

An assertion p satisfying I1 and I2 is called inductive.

Claim 3. *Rule BINV is sound.*

Proof Let $\sigma : s_0, s_1, \ldots$ be a computation of $\mathcal{D}$. By premise I1, s_0 satisfies p. We show that, for every $j = 0, 1, \ldots$, the validity of p propagates from s_j to s_{j+1}. Assume that $s_j \models p$. This implies that $p(s_j[V]) = 1$. Since s_{j+1} is a $\mathcal{D}$-successor of s_j, it follows that $\rho(s_j[V], s_{j+1}[V]) = 1$. By premise I2, we infer that $p(s_{j+1}[V]) = 1$, i.e., $s_{j+1} \models p$.

By induction on $j = 0, 1, \ldots$, we conclude that every s_j satisfies p, i.e., p is a $\mathcal{D}$-invariant.

Example: Program MUX-SEM

Consider the following parameterized program coordinating mutual exclusion by semaphores.

$$y : \textbf{integer where } y = 1$$

$$\overset{N}{\underset{i=1}{\|}} P[i] :: \left[\begin{array}{l} \ell_0 : \ \textbf{loop forever do} \\ \qquad \left[\begin{array}{ll} \ell_1 : & \textbf{Non-critical} \\ \ell_2 : & \textbf{request } y \\ \ell_3 : & \textbf{Critical} \\ \ell_4 : & \textbf{release } y \end{array}\right] \end{array}\right]$$

The semaphore instructions **request** y and **release** y respectively stand for

$$\langle \textbf{when } y > 0 \textbf{ do } y := y - 1 \rangle \quad \text{and} \quad y := y + 1.$$

We use rule BINV to verify the invariance of the assertion

$$p_1 : \quad y \geq 0$$

This assertion is inductive so the proof succeeds.

For example, one of the instances of premise I2 is

$$\underbrace{y \geq 0}_{p} \wedge \underbrace{\exists i : [1..N] : \pi[i] = 2 \ \wedge \ y > 0 \ \wedge \ y' = y - 1 \ \wedge \ \pi' = (\pi \textbf{ with } [(i) := 3])}_{\rho_2} \quad \rightarrow \quad \underbrace{y' \geq 0}_{p'}$$

Next, let us try to verify the property of mutual exclusion which can be specified as the invariance of the assertion

$$p_2: \quad \neg(at\_\ell_3[1] \ \wedge \ at\_\ell_3[2])$$

This attempt fails.

Not Every Invariant Assertion is Inductive

As is already explained when one learns mathematical induction, there are valid assertions p which cannot be proven by induction, where the induction hypothesis is taken to be p itself.

For example, the claim

The sum $1+3+5+\cdots+(2k-1)$ is a perfect square

or, more mathematically

$$p: \quad \exists u: 1+3+5+\cdots+(2k-1) = u^2$$

cannot be proven by induction, using p as the induction hypothesis.

To overcome this difficulty, one often has to come up with a strengthening of p, being an assertion φ which implies p and is inductive. For the above example, this can be

$$\varphi: \quad 1+3+5+\cdots+(2k-1) = k^2$$

Rule INV

The above considerations lead to the more general INV rule.

Rule INV

For an assertion φ,

I1. $\Theta \rightarrow \varphi$

I2. $\varphi \wedge \rho \rightarrow \varphi'$

I3. $\varphi \rightarrow p$

$\square\, p$

By premises I1 and I2, φ is an invariant of the system. That is, all reachable states satisfy φ. Since, by premise I3, φ implies p, it follows that p is also a $\mathcal{D}$-invariant.

For example, we can establish the invariance of

$$p_2: \quad \neg(at\_\ell_3[1] \ \wedge \ at\_\ell_3[2])$$

using rule INV with the strengthening

$$\varphi: \quad (y \geq 0) \wedge (at\_\ell_{3,4}[1] + at\_\ell_{3,4}[2] + \cdots + at\_\ell_{3,4}[N] + y = 1)$$

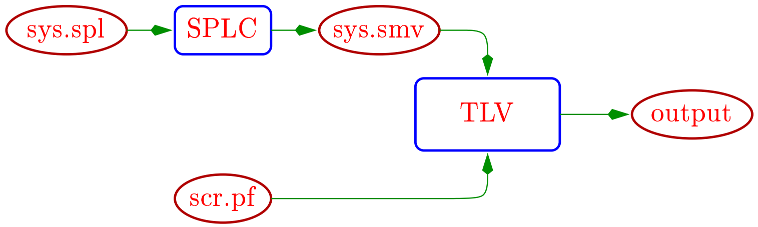
Using TLV for Incremental Strengthening

The TLV tool, developed by Elad Shahar, is a programmable symbolic calculator over finite-state systems, based on the CMU symbolic model checker SMV.

It can be used to model check LTL formulas over finite-state systems. As we will show, it can also be used for incremental development of inductive assertions.

To do so, we define a finite-state restriction of the original program, explicitly calculate the candidate assertion, and apply rule BINV.

- If the rule application produces a counter-example, the assertion is not inductive. We should strengthen it, and repeat the procedure.
- If the rule application succeeds, there are good chances (but no guarantee) that the assertion is inductive. This it the time to shift to PVS in order to get the final confirmation.



The Input File mux3.smv

```
MODULE main
DEFINE N:= 3;
VAR   y : boolean;
      P : array 1..N of process MP(y);
      Id: process Idle;
ASSIGN        init(y) := 1;
MODULE Idle
MODULE MP(y)
VAR   loc: 0..4;
ASSIGN
  init(loc) := 0;
  next(loc) :=  case
                 loc in {0,1,3,4} : (loc + 1) mod 5;
                 loc = 2 & y      : 3;
                 1                : loc;
                esac;
  next(y)    := case
                 loc = 2 & next(loc) = 3 : 0;
                 loc = 4 & next(loc) = 0 : 1;
                 1                       : y;
                esac;
JUSTICE      loc != 0,  loc != 3, loc != 4
COMPASSION   (loc = 2 & y,   loc = 3)
```


Model Checking Mutual Exclusion

In file scr1.pf, we place the text

```
Print "\n Model Check mutual exclusion between P[1] and P[2]\n";
mc ltl([]!(P[1].loc=3 & P[2].loc=3));
```

We then run

```
tlv mux3.smv
TLV version 3.1
                    . . . . .
Loaded rules file /home/amir/Tlv/Rules.tlv.

      Your wish is my command ...

>> Load "scr1.pf";

 Model Check mutual exclusion between P[1] and P[2]
Model checking...

*** Property is VALID ***
>>
```

Trying First Approximation: $\varphi_2 : \forall i \neq j : \neg(at\_\ell_3[i] \;\wedge\; at\_\ell_3[j])$

In file `scr2.pf`, we place

```
Print "\n Try deductive verification of mutual exclusion\n";
To prepare_assertion;
Let i:= N;
Let ass := 1;
While (i)
  Let j := N;
  While (j)
    Let ass := ass & (i=j | P[i].loc != 3 | P[j].loc != 3);
    Let j := j - 1;
  End  -- While (j)
  Let i := i - 1;
End -- While(i)
End -- prepare_assertion
prepare_assertion;
Call binv(ass);
```

Running this script file, we obtain:

```
>> Load "scr2.pf";
 Try deductive verification of mutual exclusion
Checking Premise I1
Premise I1 is valid. Checking Premise I2.
Premise I2 is not valid. Counter-example =
y = 1,0   P[1].loc = 0,0   P[2].loc = 2,3   P[3].loc = 3,3
```


Strengthening the Assertion

The offending transition captures a situation in which `P[3]` is already at location ℓ_3 and `P[2]` has just joined it. Is such a situation possible in a real computation?

No! because in a real computation, if any process is at ℓ_3 then y must equal 0.

Consequently, we strengthen φ_2 into

$$\varphi_3 : \quad \varphi_2 \;\wedge\; \forall i : at\_\ell_3[i] \rightarrow y = 0$$

Trying Second Approximation:

$$\varphi_3 : \forall i : (at\_\ell_3[i] \rightarrow y = 0) \;\wedge\; \forall j \neq i : \neg(at\_\ell_3[i] \;\wedge\; at\_\ell_3[j])$$

In file `scr3.pf`, we place

```
...
While (i)
  Let ass := ass & ((P[i].loc = 3) -> y=0);
  Let j := N;
  While (j)
    Let ass := ass & (i=j | P[i].loc != 3 | P[j].loc != 3);
    Let j := j - 1;
  End  -- While (j)
  Let i := i - 1;
End -- While(i)
...
```

Running this script file, we obtain:

```
>> Load "scr3.pf";
 Try deductive verification of mutual exclusion
Checking Premise I1
Premise I1 is valid. Checking Premise I2.
Premise I2 is not valid. Counter-example =
y = 0,1   P[1].loc = 0,0   P[2].loc = 4,0  P[3].loc = 3,3
```


Strengthening φ_3

The offending transition originates at a state in which `P[2]` is at location ℓ_4 while `P[3]` is at location ℓ_3. Such a state is unreachable, because the range for which mutual exclusion is ensured includes ℓ_4 together with ℓ_3.

Consequently, we strengthen φ_3 into

$$\varphi_4 : \quad \forall i : at\_\ell_3[i] \rightarrow y = 0 \;\wedge\; \forall j \neq i : \neg(at\_\ell_{3,4}[i] \;\wedge\; at\_\ell_{3,4}[j])$$

Trying next Approximation:

$\varphi_4: \quad \forall i: at\_\ell_3[i] \rightarrow y=0 \;\wedge\; \forall j \neq i: \neg(at\_\ell_{3,4}[i] \;\wedge\; at\_\ell_{3,4}[j])$

In file `scr4.pf`, we replace

```
Let ass := ass & (i=j | P[i].loc != 3 | P[j].loc != 3);
```

as it appeared in `scr3.pf`, by:

```
Let ass := ass & (i=j | P[i].loc < 3 | P[j].loc < 3);
```

Running this version, we obtain

```
...
Premise I2 is not valid. Counter-example =
y = 1,0   P[1].loc = 0,0    P[2].loc = 4,4   P[3].loc = 2,3
```

The pre-state of this counter-example is unreachable because it has `P[2]` at location ℓ_4 while $y=1$. It is thus necessary to extend the range for which $y=0$ to include also ℓ_4. Consequently, we strengthen φ_4 into

$$\varphi_5: \quad \forall i: at\_\ell_{3,4}[i] \rightarrow y=0 \;\wedge\; \forall j \neq i: \neg(at\_\ell_{3,4}[i] \;\wedge\; at\_\ell_{3,4}[j])$$

Once More: Try

$\varphi_5: \quad \forall i: at\_\ell_{3,4}[i] \rightarrow y=0 \;\wedge\; \forall j \neq i: \neg(at\_\ell_{3,4}[i] \;\wedge\; at\_\ell_{3,4}[j])$

In file `scr5.pf`, we replace

```
Let ass := ass & ((P[i].loc = 3) -> y=0);
```

as it appeared in `scr3.pf`, by:

```
Let ass := ass & ((P[i].loc > 2) -> y=0);
```

Running this version, we obtain

```
...
 Try deductive verification of mutual exclusion
Checking Premise I1
Premise I1 is valid. Checking Premise I2.
Premise I2 is valid.
 * * * Assertion p is invariant.
```