# Loop Invariants

Let us prove the following program correct for all possible values of $n$ and $n \geq 0$:

```
assign(i,0) seq
assign(p,0) seq
while not(eq(i,n)) do
    assign(i,add(i,1)) seq
    assign(p,add(p,i))
```

With the pre- and postconditions:

$$\begin{aligned} \mathrm{pre}(s) &\equiv \mathrm{lookup}(n, s, vn) \wedge vn \geq 0 \\ \mathrm{post}(s) &\equiv \mathrm{lookup}(p, s, vp) \wedge vp = \sum_{j=0}^{vn} j \end{aligned}$$

It becomes clear that the postcondition is an appropriate model for the program when we try various values for $v$, i.e., $vn = 3$. It is easy to see that in this case $vp = 6$.

# Loop Invariants

Let us use loop invariants to prove the following program correct:

```
{pre}
assign(i,0) seq
assign(p,0) seq
{inv}
while not(eq(i,n)) do
    {inv ∧ B}
   assign(i,add(i,1)) seq
   assign(p,add(p,i))
   {inv}
{inv ∧ ¬B}
{post}
```

where $B$ is computed as $(\mathrm{le}(i,n),s) \longrightarrow\!\!\!\!\gg B$.

$$\begin{aligned}
\mathrm{pre}(s) &\equiv \mathrm{lookup}(n,s,vn) \wedge vn \geq 0 \\
\mathrm{post}(s) &\equiv \mathrm{lookup}(p,s,vp) \wedge vp = \sum_{j=0}^{vn} j \\
\mathrm{inv}(s) &\equiv \mathrm{lookup}(p,s,vp) \wedge (i,s) \longrightarrow\!\!\!\!\gg vi \wedge vp = \sum_{j=0}^{vi} j
\end{aligned}$$

**NOTE:** the loop invariants can often be derived from the post condition.

# Loop Invariants

Our three proof obligations are:

$$(\mathrm{init}, S) \twoheadrightarrow Q \land [\mathrm{pre}(S) \Rightarrow \mathrm{inv}(Q)]$$
$$(\mathrm{body}, S) \twoheadrightarrow Q \land (\mathrm{guard}, S) \twoheadrightarrow B \land [(\mathrm{inv}(s) \land B) \Rightarrow \mathrm{inv}(Q)]$$
$$(\mathrm{guard}, T) \twoheadrightarrow B \land [(inv(T) \land \neg B) \Rightarrow post(T)]$$

# Loop Invariants

```
% sum.pl
:-['sem.pl'].

:- >>> 'define the parts of our program'.
init(assign(i,0) seq assign(p,0)).
guard(not(eq(i,n))).
body(assign(i,add(i,1)) seq assign(p,add(p,i))).

:- >>> 'define our sum operation'.
:- dynamic sum/2.

sum(0,0).

sum(X,Y) :-
    T1 is X-1,
    sum(T1,T2),
    Y is X+T2.

:- >>> 'first proof obligation'.
:- >>> 'assume precondition'.
:- asserta(lookup(n,s,vn)).
:- >>> 'proof the invariant'.
:- init(I),(I,s) -->> Q,lookup(p,Q,VP),lookup(i,Q,VI), sum(VI,VP).
:- retract(lookup(n,s,vn)).
```

# Loop Invariants

```
:- >>> 'second  proof obligation'.
:- >>> 'assume invariant on s'.
:- asserta(lookup(p,s,vp)).
:- asserta(lookup(i,s,vi)).
:- asserta(sum(vi,vp)).
% implies
:- asserta(sum(vi+1,vp+(vi+1))).
:- >>> 'assume guard on s'.
:- asserta((not(eq(i,n)),s) -->> true).
:- >>> 'proof the invariant on Q'.
:- body(Bd),(Bd,s) -->> Q,lookup(p,Q,VP),lookup(i,Q,VI), sum(VI,VP).
:- retract(lookup(p,s,vp)).
:- retract(lookup(i,s,vi)).
:- retract(sum(vi,vp)).
:- retract(sum(vi+1,vp+(vi+1))).
:- retract((not(eq(i,n)),s) -->> true).
```

# Loop Invariants

```
:- >>> 'third  proof obligation'.
:- >>> 'assume the invariant on s'.
:- asserta(lookup(p,s,vp)).
:- asserta(lookup(i,s,vi)).
:- asserta(sum(vi,vp)).
:- >>> 'assume NOT guard on s'.
:- asserta((not(eq(i,n)),s) -->> not(true)).
% implies
:- asserta((eq(i,n),s) -->> true).
% implies
:- asserta(sum(vn,vp)).
:- >>> 'prove postcondition on s'.
:- lookup(p,s,VP),sum(vn,VP).
:- retract(lookup(p,s,vp)).
:- retract(lookup(i,s,vi)).
:- retract(sum(vi,vp)).
:- retract((not(eq(i,n)),s) -->> not(true)).
:- retract((eq(i,n),s) -->> true).
:- retract(sum(vn,vp)).
```

# Factorial

Let us consider the factorial program:

```
assign(i,1) seq
assign(z,1) seq
while not(eq(i,n)) do
     assign(i,add(i,1)) seq
     assign(z,mult(z,i))
```

With pre- and postconditions:

$$\begin{aligned}\mathrm{pre}(S) &\equiv \mathrm{lookup}(n, S, vn) \wedge vn \geq 1\\ \mathrm{post}(Q) &\equiv \mathrm{lookup}(z, Q, vn!)\\ \mathrm{inv}(T) &\equiv \mathrm{lookup}(z, T, vz) \wedge \mathrm{lookup}(i, T, vi) \wedge vz = vi!\end{aligned}$$

# Factorial

Our three proof obligations are:

$$(\mathrm{init}, S) \twoheadrightarrow Q \wedge [\mathrm{pre}(S) \Rightarrow \mathrm{inv}(Q)]$$

$$(\mathrm{body}, S) \twoheadrightarrow Q \wedge (\mathrm{guard}, S) \twoheadrightarrow B \wedge [(\mathrm{inv}(s) \wedge B) \Rightarrow \mathrm{inv}(Q)]$$

$$(\mathrm{guard}, T) \twoheadrightarrow B \wedge [(inv(T) \wedge \neg B) \Rightarrow post(T)]$$

# Factorial

```
% fact.pl
:-['sem.pl'].

:- >>> 'define the parts of our program'.
init(assign(i,1) seq assign(z,1)).
guard(not(eq(i,n))).
body(assign(i,add(i,1)) seq assign(z,mult(z,i))).

:- >>> 'define our fact operation'.
:- dynamic fact/2.

fact(1,1).

fact(X,Y) :-
    T1 is X-1,
    fact(T1,T2),
    Y is X*T2.

:- >>> 'first proof obligation'.
:- >>> 'assume precondition'.
:- asserta(lookup(n,s,vn)).
:- >>> 'prove the invariant'.
:- init(I),(I,s) -->> Q,lookup(z,Q,VZ),lookup(i,Q,VI), fact(VI,VZ).
:- retract(lookup(n,s,vn)).
```

# Factorial

```
:- >>> 'second  proof obligation'.
:- >>> 'assume invariant on s'.
:- asserta(lookup(z,s,vz)).
:- asserta(lookup(i,s,vi)).
:- asserta(fact(vi,vz)).
% implies
:- asserta(fact(vi+1,vz*(vi+1))).
:- >>> 'assume guard on s'.
:- asserta((not(eq(i,n)),s) -->> true).
:- >>> 'prove the invariant on Q'.
:- body(Bd),(Bd,s) -->> Q,lookup(z,Q,VZ),lookup(i,Q,VI), fact(VI,VZ).
:- retract(lookup(z,s,vz)).
:- retract(lookup(i,s,vi)).
:- retract(fact(vi,vz)).
:- retract(fact(vi+1,vz*(vi+1))).
:- retract((not(eq(i,n)),s) -->> true).
```

# Factorial

```
:- >>> 'third  proof obligation'.
:- >>> 'assume the invariant on s'.
:- asserta(lookup(z,s,vz)).
:- asserta(lookup(i,s,vi)).
:- asserta(fact(vi,vz)).
:- >>> 'assume NOT guard on s'.
:- asserta((not(eq(i,n)),s) -->> not(true)).
% implies
:- asserta((eq(i,n),s) -->> true).
% implies
:- asserta(fact(vn,vz)).
:- >>> 'prove postcondition on s'.
:- lookup(z,s,VZ),fact(vn,VZ).
:- retract(lookup(z,s,vz)).
:- retract(lookup(i,s,vi)).
:- retract(fact(vi,vz)).
:- retract((not(eq(i,n)),s) -->> not(true)).
:- retract((eq(i,n),s) -->> true).
:- retract(fact(vn,vz)).
```