Pre- and Postconditions

Up to now our program specifications only considered the effect of the program on the state, we simply assumed that the state before the program executes is suitable.

Let us refine our notion of program specification a little bit to include a specification of what we expect the state to look like before our program executes.

We do this with a pair of predicates: the precondition pre and the postcondition post,[1]

Precondition – A precondition is a predicate that must always be true just prior to the execution of some section of code. If a precondition is violated, the effect of the section of code becomes undefined and thus may or may not carry out its intended work.

Postcondition – A postcondition is a predicate that must always be true just after the execution of some section of code. If the postcondition is violated then the section of code did not execute correctly.

[1] Definitions due to Wikipedia.

Pre- and Postconditions

Formally we state this as follows:

$$\forall s, \exists P, Q \; [(P, s \relbar\joinrel\gg Q \wedge \mathrm{pre}(s) \Rightarrow \mathrm{post}(Q)],$$

for some program P, where s and Q are states.

> *The program is correct if when the precondition holds on the initial state then this implies that the postcondition holds on the final state.*

Pre- and Postconditions

Consider again our swap program. Being more explicit about the contents of the initial state we have the precondition

$$\mathrm{pre}(R) \equiv \mathrm{lookup}(x, R, vx) \wedge \mathrm{lookup}(y, R, vy)$$

and the postcondition

$$\mathrm{post}(T) \equiv \mathrm{lookup}(x, T, vy) \wedge \mathrm{lookup}(y, T, vx)$$

Pre- and Postconditions

```
% swap-prepost.pl
:-['sem.pl'].

:- >>> 'show that program P="assign(t,x) seq assign(x,y) seq assign(y,t))"'.
:- >>> 'satisfies the program specification:'.
:- >>> ' pre(R) = lookup(x,R,vx) ^ lookup(y,R,vy)'.
:- >>> ' post(T) = lookup(x,T,vy) ^ lookup(y,T,vx)'.

program(assign(t,x) seq assign(x,y) seq assign(y,t)).

:- >>> 'assert precondition'.
:- asserta(lookup(x,s,vx)).
:- asserta(lookup(y,s,vy)).

:- >>> 'show that postcondition holds'.
:- program(P),
     (P,s) -->> Q,
     lookup(x,Q,vy),
     lookup(y,Q,vx).
```

Pre- and Postconditions

Consider the program that computes the maximum value of two variables, n and m, and deposits the result into z.

The precondition is

$$\mathrm{pre}(S) \equiv \mathrm{lookup}(m, S, vm) \wedge \mathrm{lookup}(n, S, vn)$$

and the postcondition is

$$\mathrm{post}(Q) \equiv \mathrm{lookup}(z, Q, VZ) \wedge V \text{ xis } (\mathrm{max}(vm, vn)) \wedge VZ = V$$

Pre- and Post-Conditions

```
% max-prepost.pl
:-[’sem.pl’].

:- >>> ’show that program P="if(le(n,m),assign(z,m),assign(z,n))"’.
:- >>> ’satisfies the program specification:’.
:- >>> ’ pre(S) = lookup(m,S,vm) ˆ lookup(n,S,vm)’.
:- >>> ’ post(Q) = lookup(z,Q,VZ)) ˆ V xis max(vm,vn) ˆ VZ = V’.

program(if(le(n,m),assign(z,m),assign(z,n))).

:- >>> ’assert precondition’.
:- asserta(lookup(m,s,vm)).
:- asserta(lookup(n,s,vn)).

:- >>> ’show that postcondition holds; case analysis on values vm and vn’.
:- >>> ’case max(vm,vn)=vm’.
:- asserta(vm xis max(vm,vn)).
% this implies that
:- asserta(true xis (vn =< vm)).
:- program(P),  (P,s) -->> Q, lookup(z,Q,VZ), V xis max(vm,vn), VZ = V.
:- retract(vm xis max(vm,vn)).
:- retract(true xis (vn =< vm)).

:- >>> ’case max(vm,vn)=vn’.
:- asserta(vn xis max(vm,vn)).
% this implies that
:- asserta(false xis (vn =< vm)).
:- program(P), (P,s) -->> Q, lookup(z,Q,VZ), V xis max(vm,vn), VZ = V.
:- retract(vn xis max(vm,vn)).
:- retract(false xis (vn =< vm)).
```