Axiomatic Semantics

- Based on techniques from predicate logic.
- More abstract than denotational semantics.
- There is no concept of “state of the machine”.
- Semantic meaning of a program is based on assertions about relationships that remain the same each time the program executes.
- Application: Proving programs to be correct.

Limitations

- Side effects disallowed in expressions.
- **goto** command difficult to specify.
- Aliasing not allowed.
- Scope rules difficult to describe ⇒ require all identifier names to be unique.

Concentrate on commands in Wren.

Assertion

A logical formula, say
(m ≠ 0 and (sqrt(m)) ^{2} *= m)*, that is true when a point in the program is reached.

Precondition: Assertion before a command.

Postcondition: Assertion after a command.

{ PRE } C *{ POST }*

Partial Correctness

If the initial assertion (the precondition) is true *and* if the program terminates, then the final assertion (the postcondition) must be true.

Precondition + Termination ⇒ Postcondition

Total Correctness

Given that the precondition for the program is true, the program must terminate and the postcondition must be true.

Total Correctness =
Partial Correctness + Termination

Assignment Command

1) *{ true }* m **:=** 13 *{ m = 13 }*

2) *{ n = 3 and c = 2 }* n **:=** c∗n *{ n = 6 and c = 2 }*

3) *{ k ≥ 0 }* k **:=** k + 1 *{ k > 0 }*

Notation

{ Precondition } command *{ Postcondition }*

P[V→E] denotes substitution of E for V in P

Axiom for assignment command

{ P[V→E] } V **:=** E *{ P }*

Work backwards:

Postcondition: P ≡ (n = 6 and c = 2)

Command: n **:=** c∗n

Precondition: P[V→E] ≡ (c∗n = 6 and c = 2)
≡ (n = 3 and c = 2)

Read and Write Commands

Notation

Use “IN = [1,2,3]” and “OUT = [4,5]” to represent input and output files.

[M]L denotes list whose head is M and tail is L.

Use small caps, K, M, N, …, to represent arbitrary numerals.

Axiom for Read Command

{ IN = [K]L and P[V→K] } **read** V *{ IN = L and P }*

Axiom for Write Command

{ OUT=L and E=K and P }
write E
{ OUT=L[K] and E=K and P }

Note: L[K] means *affix*(L,K).

Rules of Inference

$$\frac{H_1, H_2, \ldots, H_n}{H}$$

Compare with structural operational semantics.

Axiom for Command Sequencing

$$\frac{\{P\}\ C_1\ \{Q\},\ \ \{Q\}\ C_2\ \{R\}}{\{\ P\ \}\ \ C_1\ ;\ C_2\ \{\ R\ \}}$$

Axioms for If Commands

$$\frac{\{P \text{ and } B\}\ C_1\ \{Q\},\ \ \{P \text{ and not } B\}\ C_2\ \{Q\}}{\{\ P\ \}\ \ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{end if}\ \{\ Q\ \}}$$

$$\frac{\{\ P \text{ and } B\}\ \ C\ \{Q\},\ \ (P \text{ and not } B) \supset Q}{\{\ P\ \}\ \ \textbf{if}\ B\ \textbf{then}\ C\ \textbf{end if}\ \{\ Q\ \}}$$

Weaken Postcondition

$$\frac{\{P\}\ C\{Q\},\ \ Q \supset R}{\{\ P\ \}\ \ C\ \{\ R\ \}}$$

Strengthen Precondition

$$\frac{P \supset Q,\ \ \{Q\}\ C\{R\}}{\{\ P\ \}\ \ C\ \{\ R\ \}}$$

And and Or Rules

$$\frac{\{P\}\ C\ \{Q\},\ \ \{P'\}\ C\ \{Q'\}}{\{\ P \text{ and } P'\ \}\ \ C\ \{\ Q \text{ and } Q'\ \}}$$

$$\frac{\{P\}\ C\{Q\},\ \ \{P'\}\ C\{Q'\}}{\{\ P \text{ or } P'\ \}\ \ C\ \{\ Q \text{ or } Q'\ \}}$$

Observation

{ false } any-command *{ any-postcondition }*

Example

{ IN = [4,9,16] and OUT = [0,1,2] }

```
read m;  read n;
if m>=n then
            a := 2*m
        else
            a := 2*n
end if;
write a
```

{ IN = [16] and OUT = [0,1,2,18] }

{ IN = [4,9,16] and OUT = [0,1,2] } ⊃

{ IN = [4][9,16] and OUT = [0,1,2] and 4=4 }

read m;

{ IN = [9,16] and OUT = [0,1,2] and m=4 } ⊃

{ IN = [9][16] and OUT = [0,1,2] and m=4 and 9=9 }

read n;

{ IN = [16] and OUT = [0,1,2] and m=4 and n=9 }

Let S = *{ IN = [16] and OUT = [0,1,2] and m=4 and n=9 }*

and B ≡ *m≥n*

Then

(S and B) ⊃ false,

and

S ⊃ not B

So

{ S and B }, which is equivalent to false

a := 2*m

{ IN = [16] and OUT = [0,1,2] and m=4 and n=9 and a=18 },

and

{ S and not B } ⊃
{ IN = [16] and OUT = [0,1,2] and m=4 and n=9 and 2•n=18 }

a := 2*n

{ IN = [16] and OUT = [0,1,2] and m=4 and n=9 and a=18 }

Therefore by one of the If axioms,

{ *S* }

```
if m>=n   then
              a := 2*m
          else
              a := 2*n
end if;
```

{ *IN = [16] and OUT = [0,1,2] and m=4 and n=9 and a=18* }

and

{ *IN = [16] and OUT = [0,1,2] and m=4 and n=9 and a=18* }

write a

{ *IN = [16] and OUT = [0,1,2] [18] and m=4 and n=9 and a=18* }

which implies

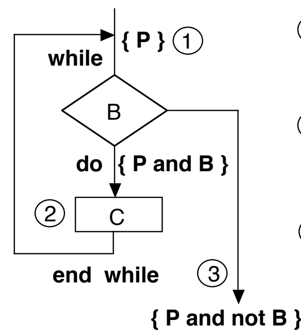
{ *IN = [16] and OUT = [0,1,2,18]* }

While Command

$$\frac{\{P \text{ and } B\}\ C\ \{P\}}{\{\ P\ \}\ \textbf{while}\ B\ \textbf{do}\ C\ \textbf{end while}\ \{\ P \text{ and not } B\ \}}$$

Loop Invariant: P

- Preserved during execution of the loop



1 **Initialization:** Show the loop invariant is initially true.

2 **Preservation:** Show the loop invariant remains true when the loop executes.

3 **Completion:** Show the loop invariant and the exit condition produce the final assertion.

Main Problem: Constructing the loop invariant.

Loop Invariant

- A relationship among the variables that does not change as the loop is executed.
- Look for some expression that can be combined with not B to produce part of the postcondition.
- Construct a table of values to see what stays constant.
- Combine what has already been computed at some stage in the loop with what has yet to be computed to yield a constant of some sort.

Look at the factorial example carefully.

Example: Exponent

{ *N≥0 and A≥0* }

```
k := N;  S := 1;
while k>0 do
    S := A*S;
    k := k-1
end while
```

{ $s = A^N$ }

Trace algorithm with small numbers A=2, N=5.

Build a table of values to find loop invariant.

k	**s**	**2^k**	**$s \cdot 2^k$**
5	1	32	32
4	2	16	32
3	4	8	32
2	8	4	32
1	16	2	32
0	32	1	32

Notice that k is decreasing and that 2^k represents the computation that still needs to be done.

The value $s \cdot 2^k = 32$ remains constant throughout the execution of the loop.

Observe that s and 2^k change when k changes.

Their product is constant, namely $32 = 2^5 = A^N$.

This suggests that $s \cdot A^k = A^N$ as part of the invariant.

The relation k≥0 seems to be invariant, and when combined with “not B”, which is k≤0, establishes k=0 at the end of the loop.

When k=0 is joined with $s \cdot A^k = A^N$, we get the postcondition $s = A^N$.

Loop Invariant:

{ k≥0 and $s \cdot A^k = A^N$ }.

Verification of Program

Initialization:

{ N≥0 and A≥0 } ⊃
{ N=N≥0 and A≥0 and 1=1 }
k := N; s := 1;
{ k=N≥0 and A≥0 and s=1 } ⊃
{ k≥0 and $s \cdot A^k = A^N$ *}*

Preservation:

{ k≥0 and $s \cdot A^k = A^N$ *and k>0 }* ⊃
{ k>0 and $s \cdot A^k = A^N$ *}* ⊃
{ k>0 and $s \cdot A \cdot A^{k-1} = A^N$ *}* ⊃
{ k>0 and $A \cdot s \cdot A^{k-1} = A^N$ *}*
s := A*s;
{ k>0 and $s \cdot A^{k-1} = A^N$ *}* ⊃
{ k-1≥0 and $s \cdot A^{k-1} = A^N$ *}*
k := k-1
{ k≥0 and $s \cdot A^k = A^N$ *}*

Completion:

{ k≥0 and $s \cdot 2^k = A^N$ *and k≤0 }* ⊃
{ k=0 and $s \cdot 2^k = A^N$ *}* ⊃ *{* $s = A^N$ *}*

Example: Nested While Loops

{ IN = [A] and OUT = [ ] and A ≥ 0 }

① **read** x;
m **:=** 0; n **:=** 0; s **:=** 0;

② **while** x>0 **do** ③ *{ outer loop invariant: C }*

x **:=** x–1; n **:=** m+2; m **:=** m+1;

④ **while** m>0 **do** ⑤ *{ inner loop invariant: D }*

m **:=** m–1; s **:=** s+1 ⑥

end while; ⑦

m **:=** n ⑧

end while; ⑨

write s ⑩

{ OUT = [A^2*] }*

Introduce boolean valued terms, called predicates, to refer to the invariants.

The outer invariant C is

C(x,m,n,s) ≡

(x≥0 and m=2(A–x) and m=n≥0 and $s=(A-x)^2$ *and OUT=[ ])*

First prove this invariant is true initially by pushing it back through the initialization code.

① ➜ ②

{ IN = [A] and OUT = [ ] and A≥0 } ⊃

{ A≥0 and 0=2(A–A) and $0=(A-A)^2$ *and IN = [A][ ] and OUT=[ ] }*

read x;

{ x≥0 and 0=2(A–x) and $0=(A-x)^2$ *and IN = [ ] and OUT=[ ] }* ⊃

{ x≥0 and 0=2(A–x) and 0=0 and $0=(A-x)^2$ *and OUT=[ ] }*

m **:=** 0;

$\{\ x\geq 0 \text{ and } m=2(A-x) \text{ and } m=0$
$\quad \text{and } 0=(A-x)^2 \text{ and } OUT=[\,]\ \} \supset$

$\{\ x\geq 0 \text{ and } m=2(A-x) \text{ and } m=0 \text{ and } 0\geq 0$
$\quad \text{and } 0=(A-x)^2 \text{ and } OUT=[\,]\ \}$

n **:=** 0;

$\{\ x\geq 0 \text{ and } m=2(A-x) \text{ and } m=n \text{ and } n\geq 0$
$\quad \text{and } 0=(A-x)^2 \text{ and } OUT=[\,]\ \}$

s **:=** 0

$\{\ x\geq 0 \text{ and } m=2(A-x) \text{ and } m=n\geq 0$
$\quad \text{and } s=(A-x)^2 \text{ and } OUT=[\,]\ \}$

Next show that the outer loop invariant and the exit condition, followed by the **write** command, produce the desired final assertion.

⑨ ➜ ⑩

$\{\ C(x,m,n,s) \text{ and } x\leq 0\ \}$

$\supset \{\ x=0 \text{ and } m=2A \text{ and } m=n\geq 0$
$\quad \text{and } s=A^2 \text{ and } OUT=[\,]\ \}$

$\supset \{\ s=A^2 \text{ and } OUT=[\,]\ \}$

and

$\{\ s=A^2 \text{ and } OUT=[\,]\ \}$

write s

$\{\ s=A^2 \text{ and } OUT=[A^2]\ \} \supset \{\ OUT = [A^2]\ \}.$

Showing preservation of the outer loop invariant (③ ➜ ⑧ ➜ ③) involves executing the inner loop, so introduce the inner loop invariant D.

$D(x,m,n,s) \equiv$

$(\ x\geq 0 \text{ and } n=2(A-x) \text{ and } m\geq 0 \text{ and } n\geq 0$
$\quad \text{and } m+s=(A-x)^2 \text{ and } OUT=[\,]\ )$

First show the inner loop invariant is initially true by starting with the outer loop invariant, combined with the loop entry condition, and pushing it through the assignment commands before the inner loop.

③ ➜ ④

$\{\ C(x,m,n,s) \text{ and } x>0\ \}$

$\equiv \{\ x\geq 0 \text{ and } m=2(A-x) \text{ and } m=n\geq 0$
$\quad \text{and } s=(A-x)^2 \text{ and } OUT=[\,] \text{ and } x>0\ \}$

$\supset \{\ x-1\geq 0 \text{ and } m+2=2(A-x+1) \text{ and } m+1\geq 0 \text{ and}$
$\quad m+2\geq 0 \text{ and } m+1+s=(A-x+1)^2 \text{ and } OUT=[\,]\ \}$

$\equiv \{\ D(x-1,m+1,m+2,s)\ \}$

since $(s=(A-x)^2 \text{ and } m+2=2(A-x+1))$
$\quad \supset\ m+1+s=(A-x+1)^2.$

Therefore, by the assignment rule, we have:

③ $\{\ C(x,m,n,s) \text{ and } x>0\ \} \supset \{\ D(x-1,m+1,m+2,s)\ \}$

x **:=** x–1; n **:=** m+2; m **:=** m+1

④ $\{\ D(x,m,n,s)\ \}$

Next we need to show that the inner loop invariant is preserved

⑤ ➜ ⑥ ➜ ⑤

$\{\ D(x,m,n,s) \text{ and } m>0\ \}$

m **:=** m-1; s **:=** s+1

$\{\ D(x,m,n,s)\ \}.$

It suffices to show

$(\ D(x,m,n,s) \text{ and } m>0\ )$

$\supset (\ x\geq 0 \text{ and } n=2(A-x) \text{ and } m\geq 0 \text{ and } n\geq 0$
$\quad \text{and } m+s=(A-x)^2 \text{ and } OUT=[\,] \text{ and } m>0\ )$

$\supset (\ x\geq 0 \text{ and } n=2(A-x) \text{ and } m-1\geq 0 \text{ and } n\geq 0$
$\quad \text{and } m-1+s+1=(A-x)^2 \text{ and } OUT=[\,]\ )$

$\equiv D(x,m-1,n,s+1).$

To complete the proof, show that the inner loop invariant, combined with the inner loop exit condition, pushed through the assignment m **:=** n, results in the outer loop invariant.

⑦ ➜ ⑧

$\{\, D(x,m,n,s) \text{ and } m\le 0 \,\}$ m := n $\{\, C(x,m,n,s) \,\}$.

It suffices to show

$(\, D(x,m,n,s) \text{ and } m\le 0 \,)$

$\supset (\, x\ge 0 \text{ and } n=2(A-x) \text{ and } m\ge 0 \text{ and } n\ge 0$
$\quad \text{and } m+s=(A-x)^2 \text{ and } OUT=[\,] \text{ and } m\le 0 \,)$

$\supset (\, x\ge 0 \text{ and } n=2(A-x) \text{ and } n=n\ge 0$
$\quad \text{and } s=(A-x)^2 \text{ and } OUT=[\,] \,)$

$\equiv C(x,n,n,s).$ ∎

Derived Rule for Assignment

$$\frac{P \supset Q[V \to E]}{\{\, P \,\}\ V := E\ \{\, Q \,\}}$$

or

$$\frac{P \supset Q(E)}{\{\, P \,\}\ V := E\ \{\, Q(V) \,\}}$$

Discovering a Loop Invariant

Make a table of values for a simple case and trace values for the relevant variables.

Let A = 3 in the previous example.

	x	m	n	s	A-x
→	3	0	0	0	0
	2	1	2	0	1
		0	2	1	
→	2	2	2	1	
	1	3	4	1	2
		2	4	2	
		1	4	3	
		0	4	4	
→	1	4	4	4	
	0	5	6	4	3
		4	6	5	
		3	6	6	
		2	6	7	
		1	6	8	
		0	6	9	
→	0	6	6	9	

Positions where the invariant C(x,m,n,s) for the outer loop should hold are marked by arrows.

Note how the variable s takes the values of the perfect squares, 0, 1, 4, and 9, at these locations.

The difficulty is to determine what s is the square of as its values increase.

Observe that x decreases as the program executes.

Since A is constant, this means the value A–x increases: 0, 1, 2, and 3.

This gives the relationship $s = (A-x)^2$.

Also note that m is always even and increases: 0, 2, 4, 6.

This produces the relation m = 2(A–x) in the outer invariant.

For the inner loop invariant, s is not always a perfect square, but m+s is.

Also, in the inner loop, n preserves the final value for m as the loop executes.

So n also obeys the relationship n = 2(A–x).

Finally, the loop entry conditions are combined with the value that causes loop exit.

For the outer loop, x>0 is combined with x=0 to add the condition x≥0 to the outer loop invariant. Combined with x≤0, this gives x=0 at a crucial point.

In a similar way, m>0 is combined with m=0 to add m≥0 to the inner loop invariant. Combined with m≤0, this gives m=0 at the appropriate point.

The condition n≥0 is added to D to enable the proof to work.

Constructing Invariants

a) PRE: { N≥0 }

k := 1; s := 0;

while k<=N **do** s := s+k; k := k+1 **end while**

POST: { s = N•(N+1)/2 }

Loop Invariant: ________________________

b) PRE: { A>0 and B>0 }

x := A; y := B;

while x<>y **do if** x>y **then** x := x-y
else y := y-x **end if**
end while

POST: { x = gcd(A,B) }

Loop Invariant: ________________________

c) PRE: { true }

k := 1; c := 0; s := 0;

while s <= 1000 **do**

s := s+k*k; c := c+1; k := k+1

end while

POST:
{"c is the smallest number of consecutive squares starting at 1 whose sum is > 1000" }

Loop Invariant: ________________________

Axiomatic Semantics for Pelican

- Assume programs have been checked for syntactic correctness.
- Transform programs so that all identifiers have unique names.

New Kind of Inference Rule

$$\frac{H_1, H_2, \ldots, H_n \;\;|\!-\;\; H_{n+1}}{H}$$

Meaning: If H_{n+1} can be proved from H_1, H_2, ... , H_n , then conclude that H is true.

Note: H_1, H_2, ... , H_n , H_{n+1} and H are generally either of the form *{ P }* C *{ Q }* or are just simple assertions.

Premises to rules may hold important information gleaned from procedure definitions.

Given declarations

procedure p_1 **is** b_1;
procedure p_2 (n : **integer**) **is** b_2;

Form assertions (premises)

body(p_1) = b_1
parameter(p_2) = n, body(p_2) = b_2

The information in constant declarations is added to the precondition.

Given declarations

const k=5;
const f=**false**;

Add these assertions to the precondition for the command that constitutes the body of the block:

k=5 and f=false

Note: An empty collection of assertions is equivalent to true.

In rules of inference, let “Procs” and “Const” stand for the collections of assertions that result from the declarations D in a block B.

Rule for Blocks (Block):

$$\frac{\text{Procs} \ \vdash \ \{\ P \text{ and } Const\ \}\ \text{C}\ \{\ Q\ \}}{\{\ P\ \}\ \text{D } \textbf{begin} \text{ C } \textbf{end}\ \{\ Q\ \}}$$

Consider an anonymous block, **declare** Blk:

```
declare
   const a = 2;
   const c = -1;
   var m,n : integer;
begin
   m := 99;
   n := a*m + c;
   write n
end
```


Want to prove that:

{ OUT = [ ] } Blk *{ OUT = [197] }*

Procs is empty (equivalent to *true*).

Const contains the assertion *a = 2 and c = -1*.

Proof Proceeds:

{ OUT = [ ] and a=2 and c=-1 } ⊃

{ OUT = [ ] and a=2 and c=-1 and 99=99 }

m := 99;

{ OUT = [ ] and a=2 and c=-1 and m=99 } ⊃

{ OUT = [ ] and a=2 and c=-1 and a•m+c = 197 }

n **:=** a∗m + c;

{ OUT = [ ] and a=2 and c=-1 and n = 197 }

write n

{ OUT = [ ][197] and a=2 and c=-1 and n = 197 } ⊃

{ OUT = [197] }

Nonrecursive Procedures

No parameter (Call_0):

$$\frac{\{\ P\ \}\ \text{B}\ \{\ Q\ \},\ \ \text{body(proc)} = \text{B}}{\{\ P\ \}\ \ \text{proc}\ \{\ Q\ \}}$$

One parameter (Call_1):

$$\frac{\{P\}\ \text{B}\ \{Q\},\ \text{body(proc)=B, parameter(proc)=F}}{\{\ P[F \rightarrow E]\ \}\ \ \text{proc(E)}\ \{\ Q[F \rightarrow E]\ \}}$$

Example: declare Blk

```
declare
  procedure addup(num : integer) is
      var k : integer;
    begin
      k := 1;
      while k<=num do
        sum := sum+k;
        k := k+1
      end while
    end
begin  addup(A)  end
```

$\underbrace{\text{begin} \ldots \text{end}}_{\text{Bod}}$, $\underbrace{\text{procedure} \ldots \text{addup(A) end}}_{\text{Blk}}$

Prove

{ $A \geq 0$ and $sum = 0$ } Blk { $sum = A \cdot (A+1)/2$ }

For this block,

sum is nonlocal,

Procs contains the assertions
body(addup) = Bod
parameter(addup) = num,
and Const is the empty (true) assertion.

Want to show:
body(addup) = Bod,
parameter(addup) = num
$\vdash$ { $A \geq 0$ and $sum = 0$ and true } addup(A) { $sum = A \cdot (A+1)/2$ }

Let P ≡ { $num \geq 0$ and $sum = 0$ }
and Q ≡ { $sum = num \cdot (num+1)/2$ }

Then P[num→A] ≡ { $A \geq 0$ and $sum = 0$ }
and Q[num→A] ≡ { $sum = A \cdot (A+1)/2$ }

Using rule for a procedure invocation with a parameter, we need to show:

{ $num \geq 0$ and $sum = 0$ }

```
k := 1;
while k<=num do
  sum := sum+k;  k := k+1
end while
```

(Bod)

{ $sum = num \cdot (num+1)/2$ }

This derivation is left as an exercise.

Notes

- The declaration **var** k : **integer** plays no role in the derivation.
- For the block Bod, Const and Procs are empty.

Conclusion

Since { *P* } Bod { *Q* },

it follows that

{ *P[num→A]* } addup(A) { *Q[num→A]* }.

Now use (Block) to get the original assertion:

{ $A \geq 0$ and $sum = 0$ } Blk { $sum = A \cdot (A+1)/2$ }

Parameter Restrictions

- Want pass by value semantics.
- Transform each procedure into one with a new local variable for the parameter that acts in place of the formal parameter.

```
procedure p(f : integer) is         procedure p(f : integer) is
begin                                   var local#f : integer;
  f := f * f;              ➔         begin
  write f                               local#f := f;
end                                     local#f := local#f * local#f;
                                        write local#f
                                     end
```

- Actual parameter may not be altered inside the procedure.
- Add a new variable in the calling environment to pass the value.

```
procedure p(f : integer ) is        procedure p(f : integer ) is
begin                                begin
  y := y + f;              ➔           y := y + f;
  read x                               read x
end                                  end
  :                                    :
p(x);                                new#x := x;
                                     p(new#x);
```

Recursive Procedures

Example:
Find the first power of 2 bigger than 1000.

Main program:

pw := 2 ; cnt := 1 ; done := **false** ; pow

where

procedure pow **is**
begin
done := pw>1000;
if not(done**) then**
cnt := cnt+1; pw := 2*pw; pow **end if**
end

Using Call_0:

$\{P\}$ pow $\{Q\}$
if $\{P_1\}$ pow $\{Q_1\}$
if $\{P_2\}$ pow $\{Q_2\}$
if $\{P_3\}$ pow $\{Q_3\}$
if $\{P_4\}$ pow $\{Q_4\}$ …

New Rule: Recursion_0

$$\frac{\{P\}\ \text{proc}\ \{Q\}\ \vdash\ \{P\}\ B\ \{Q\},\ \text{body(proc)=B}}{\{P\}\ \text{proc}\ \{Q\}}$$

Continue Example:

Want to prove:

$\{ \mathit{true} \}$

pw := 2 ; cnt := 1 ; done := **false** ; pow

$\{ pw=2^{cnt} > 1000 \text{ and } 2^{cnt-1} \leq 1000 \}$

Recursive Assumption:

$\{ pw=2^{cnt} \text{and } 2^{cnt-1} \leq 1000 \}$ = P

pow

$\{ done \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$ = Q

Assume ≡ has higher precedence than *and*.

Need to show the following correctness specification for the body of the procedure:

$\{ pw=2^{cnt} \text{and } 2^{cnt-1} \leq 1000 \}$ = P

done := pw>1000;
if not(done) **then**
cnt := cnt+1; pw := 2*pw; pow **end if**

$\{ done \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$ = Q

We are allowed to use the recursive assumption when pow is called from within itself.

P = $\{ pw=2^{cnt} \text{ and } 2^{cnt-1} \leq 1000 \} \supset$

$\{ (pw>1000) \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$

done := pw>1000;

$\{ done \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$ = S

Let B = not(done)

Case 1: S and B

S and B ⊃

$\{ done=false \text{ and } pw \leq 1000 \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \} \supset$

$\{ 2^{cnt+1-1} \leq 1000 \text{ and } 2 \cdot pw=2^{cnt+1} \}$

cnt := cnt+1;

$\{ 2^{cnt-1} \leq 1000 \text{ and } 2 \cdot pw=2^{cnt} \}$

pw := 2*pw;

$\{ 2^{cnt-1} \leq 1000 \text{ and } pw=2^{cnt} \}$ = P

pow

$\{ done \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$ = Q

by the recursion assumption.

Case 2: S and not(B)

S and not(B) ⊃

$\{ done \equiv (pw>1000) \text{ and } pw=2^{cnt}$
$\text{and } 2^{cnt-1} \leq 1000 \}$ = Q

Now assemble the proof:

{ true } ⊃

{ 2=2 }

pw := 2;

{ pw=2 } ⊃

{ pw=2 and 1=1}

cnt := 1;

{ pw=2 and cnt=1} ⊃

{ pw=2 and cnt=1 and false=false}

done := **false**;

{ pw=2 and cnt=1 and done=false} ⊃

{ $pw=2^{cnt}$ and $2^{cnt-1} \leq 1000$ } = P

pow

{ done≡(pw>1000) and $pw=2^{cnt}$ and $2^{cnt-1} \leq 1000$ } = Q

For partial correctness, we can assume termination of the code.

Inspection indicates that termination of pow means that done=true.

Therefore, we may conclude:

{ pw>1000 and $pw=2^{cnt}$ and $2^{cnt-1} \leq 1000$ }

which implies

{ $pw=2^{cnt}>1000$ and $2^{cnt-1} \leq 1000$ }

Recursive Procedure with a Parameter

Recursion$_1$

$$\frac{\forall f(\{P[F\rightarrow f]\}\ \text{proc}(f)\ \{Q[F\rightarrow f]\})\quad \vdash \{P\}\ B\ \{Q\},\ \text{body(proc)=B. parameter(proc)=F}}{\{ P[F\rightarrow E] \}\ \text{proc}(E)\ \{ Q[F\rightarrow E] \}}$$

Example: Number of Digits

```
procedure count(m : integer) is
   begin
      if m < 10 then
                  ans := 1
               else
                  count(m/10);
                  ans := ans+1
      end if;
   end;
```

Use a global variable "ans" to hold the answer as we return from the recursive calls.

Want to prove:

{κ > 0 }
  num := κ;
{ num > 0 } = P[F→E]
  count(num)
{ $10^{ans-1} \leq num < 10^{ans}$ }. = Q[F→E]

"num" is the original actual parameter E.

"m" is the formal parameter F for each call.

Substitute the body of the procedure and bind the formal parameter to the actual parameter.

Must show

{ m = > 0 } = P
  if m < 10
    then ans := 1
    else count(m/10); ans := ans+1
  end if;
{ $10^{ans-1} \leq m < 10^{ans}$ } = Q

assuming as an induction hypotheses

∀f({ f > 0 } count(f) { $10^{ans-1} \leq f < 10^{ans}$ }),

which is ∀f({ P[F→f] } count(f) { Q[F→f] })

Use the If-Else rule.

Case 1: m < 10

$\{ m > 0 \text{ and } m<10 \} \supset$

$\{ 10^{1-1} \leq m < 10^{1}\}$

ans := 1

$\{ 10^{ans-1} \leq m < 10^{ans} \}$

Case 2: m ≥ 10

$\{ m > 0 \text{ and } m \geq 10 \} \supset$

$\{ m/10 \geq 1 \} \supset$

$\{ m/10 > 0 \}$

count(m/10) -- use f = m/10

$\{ 10^{ans-1} \leq m/10 < 10^{ans} \} \supset$

$\{ 10^{ans+1-1} \leq m < 10^{ans+1} \}$

ans := ans+1

$\{ 10^{ans-1} \leq m < 10^{ans} \}$

Termination

Most commands terminate unconditionally.

Problem Areas

- Indefinite iteration (**while**).
- Calling a recursively defined procedure.

Defn: A partial order > on a set W is **well-founded** if there exists no infinite decreasing sequence of distinct elements from W.

Consequence

Given a sequence of elements $\{x_i | i \geq 1\}$ from W such that $x_1 > x_2 > x_3 > x_4 > \ldots$, the sequence must stop (or repeat) after a finite number of elements.

If the partial order is **strict** (asymmetric) any decreasing sequence must have distinct elements and so must be finite.

Examples of Well-founded Orderings

1. Natural numbers N ordered by >.
2. Cartesian product NxN with a lexicographic ordering:
   $\langle m_1,m_2\rangle > \langle n_1,n_2\rangle$
   if $[m_1 > n_1]$ or $[m_1 = n_1$ and $m_2 > n_2]$.
3. The positive integers P ordered by the relation "properly divides":
   m > n if (∃k[m = n•k] and m≠n).

Steps in Showing Termination (while)

1. Find a set W with a strict well-founded ordering >.
2. Find a **termination expression** E with the properties:
   a) Whenever control passes through the top of the iterative loop, the value of E is in W, and
   b) E takes a smaller value with respect to > each time the top of the iterative loop is passed.

In the context of a **while** command,
"**while** B **do** C **end while**"
with invariant P, the two conditions take the form

a) $P \supset E \in W$

b) $\{ P \text{ and } B \text{ and } E=A \}$ C $\{ A > E \}$

Example

$\{ N \geq 0 \text{ and } A \geq 0 \}$

```
k := N;  s := 1;
while k>0 do
     s := A*s;
     k := k-1
end while
```

$\{ s = A^{N} \}$

Take W = N, the set of natural numbers ordered by >.

Therefore, m ∈ W if and only if m ≥ 0.

Take E = k as the termination expression.

The loop invariant P is

$$\{ k \geq 0 \text{ and } s \bullet A^k = A^N \}$$

The conditions on the termination expression must hold at the location of the invariant.

The two conditions follow immediately:

a) $P \supset$

$k \geq 0$ and $s \bullet A^k = A^N \supset$

$E = k \in W$

b) $\{ P$ and B and $E = D \} \supset$

$\{ k \geq 0$ and $s \bullet A^k = A^N$ and $k > 0$ and $k = D \} \supset$

$\{ k - 1 = D - 1 \}$

s := A*s;   k := k-1

$\{ E = k = D - 1 < D \}$

What if N≥0 is missing from Precondition?

Termination of Recursive Procedures

Use an induction proof for termination.

Example: A procedure counts the digits in a number.

```
procedure count(m : integer) is
  begin
    if m < 10 then
          ans := 1
      else
          count(m/10);
          ans := ans+1
    end if
  end;
```

This procedure terminates (normally) if it is passed a nonnegative integer.

$\{ num = K > 0 \}$

count(num)

$\{ 10^{ans-1} \leq num < 10^{ans} \}$.

The depth of recursion depends on the number of digits in num.

Lemma: If num>0, the command "count(num)" halts.

Proof: Induction on the number of digits in num.

Basis: num has one digit, that is, $0 \leq num < 10$.

Then count(num) terminates because the if test succeeds.

Induction Step: As an induction hypothesis, assume that count(num) terminates when num has k digits, namely $10^{k-1} \leq num < 10^k$.

Suppose that num has k+1 digits, namely $10^k \leq num < 10^{k+1}$. Then num/10 has k digits.

So count(num) causes the execution of the code:

```
if num < 10
   then ans := 1
   else count(num/10);
        ans := ans+1
end if
```

which terminates since count(num/10) terminates.