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 \((m \neq 0 \text{ and } (\sqrt{m})^2 = m)\), that is true when that point in the program is reached.

Precondition
An assertion before a command.

Postcondition
An 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 \text{ and } c = 2 \} n := c*n \{ n = 6 \text{ and } c = 2 \}
3) \{ k \geq 0 \} \ k := k + 1 \{ k > 0 \}

Notation

\{ Precondition \} command \{ Postcondition \}

P[V\rightarrow E] denotes substitution of E for V in P

Axiom for assignment command

\{ P[V\rightarrow E] \} V := E \{ P \}

Work backwards:

Postcondition: P \equiv (n = 6 \text{ and } c = 2)

Command: \ n := c*n

Precondition: P[V\rightarrow E] \equiv (c*n = 6 \text{ and } c = 2)
\equiv (n = 3 \text{ 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, \(K, M, N, \ldots\), to represent arbitrary numerals.

Axiom for Read Command

\{ IN = [K]L \text{ and } P[V\rightarrow K] \} \text{ read } V \{ IN = L \text{ and } P \}

Axiom for Write Command

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

Note: L[K] means affix(L,K).
Rules of Inference

\[ H_1, H_2, \ldots, H_n \\
\hline
H \]

Compare with structural operational semantics.

Axiom for Command Sequencing

\[ \{ P \} \ C_1 \ \{ Q \}, \ \{ Q \} \ C_2 \ \{ R \} \\
\hline\{ P \} \ C_1 ; C_2 \ \{ R \} \]

Axioms for If Commands

\[ \{ P \ and \ B \} \ C_1 \ \{ Q \}, \ \{ P \ and \ not \ B \} \ C_2 \ \{ Q \} \\
\hline \{ P \} \ if \ B \ then \ C_1 \ else \ C_2 \ end \ if \ \{ Q \} \]

\[ \{ P \ and \ B \} \ C \ \{ Q \}, \ (P \ and \ not \ B) \Rightarrow \ R \\
\hline \{ P \} \ if \ B \ then \ C \ end \ if \ \{ Q \} \]

Weaken Postcondition

\[ \{ P \} \ C \ \{ Q \}, \ Q \Rightarrow R \\]

\[ \{ P \} \ C \ \{ R \} \]

Strengthen Precondition

\[ P \Rightarrow Q, \ \{ Q \} \ C \ \{ R \} \\]

\[ \{ P \} \ C \ \{ R \} \]

And and Or Rules

\[ \{ P \} \ C \ \{ Q \}, \ \{ P' \} \ C \ \{ Q' \} \\]

\[ \{ P \ and \ P' \} \ C \ \{ Q \ and \ Q' \} \]

\[ \{ P \ or \ P' \} \ C \ \{ Q \ or \ Q' \} \]

Observation

\{ false \} \ any-command \ \{ any-postcondition \}
Therefore by one of the if axioms,
\[
\{ S \}
\begin{align*}
\text{if } m &\geq n & \text{then} \\
& a := 2 \times m \\
\text{else} & \\
& a := 2 \times n
\end{align*}
\text{end if;}
\]
\[
\{ \text{IN} = [16] \text{ and OUT} = [0,1,2] \text{ and } m=4 \text{ and } n=9 \text{ and } a=18 \}
\]
and
\[
\{ \text{IN} = [16] \text{ and OUT} = [0,1,2] \text{ and } m=4 \text{ and } n=9 \text{ and } a=18 \}
\]
\text{write} a
\[
\{ \text{IN} = [16] \text{ and OUT} = [0,1,2,18] \}
\]
which implies
\[
\{ \text{IN} = [16] \text{ and OUT} = [0,1,2,18] \}
\]

\text{While Command}
\[
\{ P \text{ and } B \} \text{ C } \{ P \} \\
\{ P \} \text{ whileB do C end while} \{ P \text{ and not } B \}
\]

\text{Loop Invariant P}
- Preserved during execution of the loop

\text{Main Problem: Constructing the loop invariant.}

\text{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.

\text{Example: Exponent}
\[
\{ N \geq 0 \text{ and } A \geq 0 \}
\begin{align*}
k &:= N; \ s := 1; \\
\text{while} k > 0 \text{ do} \\
& S := A \times S; \\
& k := k - 1
\end{align*}
\text{end while}
\[
\{ s = A^N \}
\]
Trace algorithm with small numbers A=2, N=5.

Build a table of values to find loop invariant.

Add a column to the table for the value of \(2^k\), since a is the variable that gets multiplied.

The value \(s \times 2^k = 32\) 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 invariant.

The relation $k \geq 0$ seems to be invariant, and when combined with “not B”, which is $k \leq 0$, 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:
\[
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \}.
\]

---

**Example: Nested While Loops**

\[
\text{IN} = [A] \text{ and } \text{OUT} = [ ] \text{ and } A \geq 0
\]

1. read $x$;
   \[
m := 0; \quad n := 0; \quad s := 0;
   \]

2. while $x > 0$ do
   
   \[
x := x - 1; \quad n := m+2; \quad m := m+1;
   \]

3. outer loop invariant: $C$

4. while $m > 0$ do
   
   \[
m := m-1; \quad s := s+1
   \]

5. inner loop invariant: $D$

6. end while

7. end while

8. \[
m := n
\]

9. end while

10. writes

\[
\text{OUT} = [A^2]
\]

---

**Verification of Program**

**Initialization**

\[
\{ N \geq 0 \text{ and } A \geq 0 \} \supset
\{ N=N \geq 0 \text{ and } A \geq 0 \text{ and } 1=1 \}
\]

\[
k := N; \quad s := 1;
\]

\[
\{ k \geq 0 \text{ and } A \cdot s \cdot A^k = A^N \} \supset
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \}
\]

**Preservation**

\[
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \text{ and } k > 0 \} \supset
\{ k > 0 \text{ and } s \cdot A^k = A^N \} \supset
\{ k > 0 \text{ and } A \cdot s \cdot A^k = A^N \}
\]

\[
s := A \cdot s;
\]

\[
\{ k \geq 0 \text{ and } A \cdot s \cdot A^k = A^N \} \supset
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \}
\]

\[
k := k-1
\]

\[
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \}
\]

**Completion**

\[
\{ k \geq 0 \text{ and } s \cdot 2^k = A^N \text{ and } k \leq 0 \} \supset
\{ k = 0 \text{ and } s \cdot 2^k = A^N \} \supset \{ s = A^N \}
\]

---

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

The outer invariant $C$ is

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

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

1. \[\rightarrow\] 2

\[
\{ \text{IN} = [A] \text{ and } \text{OUT} = [ ] \text{ and } A \geq 0 \} \supset
\{ A \geq 0 \text{ and } 0=2(A-A) \text{ and } 0=(A-A)^2
\text{ and } \text{IN} = [A][A] \text{ and } \text{OUT}=[ ] \}
\]

read $x$;

\[
\{ x \geq 0 \text{ and } 0=2(A-x) \text{ and } 0=(A-x)^2
\text{ and } \text{IN} = [ ] \text{ and } \text{OUT}=[ ] \} \supset
\{ x \geq 0 \text{ and } 0=2(A-x) \text{ and } 0=0 \text{ and } 0=(A-x)^2
\text{ and } \text{IN} = [ ] \text{ and } \text{OUT}=[ ] \}
\]
\[ m := 0; \]  
\{ x \geq 0 \text{ and } m = 2(A-x) \} \Rightarrow \{ x \geq 0 \text{ and } m = 0 \text{ and } 0 = (A-x)^2 \text{ and } \text{OUT} = [ ] \} \]

\[ n := 0; \]  
\{ x \geq 0 \text{ and } m = 2(A-x) \} \Rightarrow \{ x \geq 0 \text{ and } m = n \text{ and } n \geq 0 \text{ and } 0 = (A-x)^2 \text{ and } \text{OUT} = [ ] \} \]

\[ s := 0; \]  
\{ x \geq 0 \text{ and } m = 2(A-x) \} \Rightarrow \{ x \geq 0 \text{ and } m = n \geq 0 \text{ and } s = (A-x)^2 \text{ and } \text{OUT} = [ ] \} \]

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

\[ \begin{align*}  
\text{First show the inner loop invariant is initially true by starting with the outer loop invariant, combined with the loop entry condition, and pushing it through the assignment commands before the inner loop.} \\
\text{Next we need to show that the inner loop invariant is preserved} \\
\text{To complete the proof, show that the inner loop invariant, combined with the inner loop exit condition, pushed through the assignment } m := n, \text{ results in the outer loop invariant.} 
\end{align*} \]
7 \Rightarrow 8

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

It suffices to show

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

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

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

\equiv C(x, n, n, s).

\textbf{Derived Rule for Assignment}

\[ \frac{P \supset Q \{ \nu \rightarrow E \} }{\{ P \} \nu := E \{ Q \} } \]

or

\[ \frac{P \supset Q(E) }{\{ P \} \nu := E \{ Q(V) \} } \]

\textbf{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.

\begin{array}{cccccc}
\hline
x & m & n & s & A-x \\
\hline
3 & 0 & 0 & 0 & 0 \\
2 & 2 & 2 & 0 & 1 \\
1 & 3 & 4 & 0 & 2 \\
0 & 5 & 6 & 0 & 3 \\
\hline
\end{array}

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 \( x \leq 0 \) to the outer loop invariant. Combined with \( x \leq 0 \), this gives \( x=0 \) at a crucial point.

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

The condition \( n \geq 0 \) is added to \( D \) to enable the proof to work.
Constructing Invariants

a) PRE: \( \{ N \geq 0 \} \)
\[
k := 1; \quad s := 0;
\textbf{while} \ k \leq N \ \textbf{do} \ s := s+k; \quad k := k+1 \ \textbf{end while}
\]
POST: \( \{ s = N(N+1)/2 \} \)

Loop Invariant: _______________________

b) PRE: \( \{ A > 0 \text{ and } B > 0 \} \)
\[
x := A; \quad y := B;
\textbf{while} x \neq y \ \textbf{do} \quad \text{if } x > y \quad \text{then} x := x-y
\text{else} y := y-x \ \textbf{end if}
\textbf{end while}
\]
POST: \( \{ x = \gcd(A,B) \} \)

Loop Invariant: _______________________

c) PRE: \( \{ \text{true} \} \)
\[
k := 1; \quad c := 0; \quad s := 0;
\textbf{while} s \leq 1000 \ \textbf{do} \quad s := s+k^2; \quad c := c+1; \quad k := k+1
\textbf{end while}
\]
POST:
\( \{ \text{"c is the smallest number of consecutive squares starting at 1 whose sum is } > 1000\} \)

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

\[
\frac{H_1, H_2, \ldots, H_n \vdash H_{n+1}}{H}
\]

**Meaning** If \( H_{n+1} \) can be proved from \( H_1, H_2, \ldots, H_n \), then conclude that \( H \) is true.

**Note:** \( H_1, H_2, \ldots, H_n, H_{n+1} \) 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

\[
\text{procedure } p_1 \text{ is } b_1; \\
\text{procedure } p_2 \text{ (n : integer) is } b_2;
\]

Form assertions (premises)

\[
\text{body}(p_1) = b_1 \\
\text{parameter}(p_2) = n, \text{body}(p_2) = b_2
\]

The information in constant declarations is added to the precondition.

Given declarations

\[
\text{const } k=5; \\
\text{const } f=\text{false}
\]

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

\[
k=5 \text{ and } f=\text{false}
\]

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):

\[
\text{Procs} \vdash \{P \text{ and Const} \} C \{Q\} \\
\{P\} D \begin{array}{c} \text{begin} \\ C \end{array} \{Q\}
\]

Consider an anonymous block, declareBlk:

\[
\text{declare} \\
\text{const } a = 2; \\
\text{const } c = -1; \\
\text{var } m,n : \text{integer} \\
\begin{array}{c} \text{begin} \\ m := 99; \\ n := a \times m + c; \\ \text{writeln} \end{array} \end{array} \end{array} \end{array} \{Q\}
\]

Want to prove that:

\[
\{ \text{OUT} = [] \} \text{ Blk } \{ \text{OUT} = [197] \}
\]

Procs is empty (equivalent to true).

Const contains the assertion \( a = 2 \text{ and } c = -1 \).

Proof Proceeds:

\[
\{ \text{OUT} = [] \} \text{ and } a=2 \text{ and } c=-1 \} \supset \\
\{ \text{OUT} = [] \} \text{ and } a=2 \text{ and } c=-1 \text{ and } 99=99 \} \\
\{ m := 99; \} \supset \\
\{ \text{OUT} = [] \} \text{ and } a=2 \text{ and } c=-1 \text{ and } m=99 \} \supset \\
\{ \text{OUT} = [] \} \text{ and } a=2 \text{ and } c=-1 \text{ and } a \times m+c = 197 \} \\
\{ n := a \times m + c; \} \supset \\
\{ \text{OUT} = [] \} \text{ and } a=2 \text{ and } c=-1 \text{ and } n = 197 \} \supset \\
\{ \text{OUT} = [197] \}
\]

**Nonrecursive Procedures**

No parameter (Call\(_0\)):

\[
\{ P \} B \{ Q \}, \text{body(proc)} = B \\
\{ P \} \text{ proc } \{Q\}
\]

One parameter (Call\(_1\)):

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

Want to show:

- `body(addup) = Bod`,
- `parameter(addup) = num`

- `{ A ≥ 0 and sum = 0 and true }`  
  ```
  addup(A) { sum = A*(A+1)/2 }
  ```

Let `P ≡ { num ≥ 0 and sum = 0 }`  
and `Q ≡ { sum = num*(num+1)/2 }`

Then `P[num→A] ≡ { A ≥ 0 and sum = 0 }`  
and `Q[num→A] ≡ { sum = A*(A+1)/2 }`

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

- `{ num ≥ 0 and sum = 0 }`  
  ```
  k := 1;
  while k <= num do
    sum := sum + k;  k := k + 1
  end while
  ```
- `{ sum = num*(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 ≥ 0 and sum = 0 } Blk { sum = A*(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
begin
  f := f * f;
  write 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.
Recursive Procedures

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

Main program:
\(pw := 2\) ; \(cnt := 1\) ; \(done := \text{false}\); \(\text{pow}\)

where

procedure \(\text{pow}\) is
begin
\(\text{done} := pw > 1000\);
if \(\text{not}(\text{done})\) then
\(cnt := cnt + 1\); \(pw := 2 \times pw\); \(\text{pow}\) end if
end

Using Call\(0\):
\(\{P\} \text{ pow } \{Q\}\)
if \(\{P_1\} \text{ pow } \{Q_1\}\)
if \(\{P_2\} \text{ pow } \{Q_2\}\)
if \(\{P_3\} \text{ pow } \{Q_3\}\)
if \(\{P_4\} \text{ pow } \{Q_4\}\) …

Recursion 0
\(\{P\} \text{ proc } \{Q\} \vdash \{P\} B\{Q\}, \text{ body(proc)} = B\)
\(\{P\} \text{ proc } \{Q\}\)

Continue Example
Want to prove:
\(\{ \text{true} \}\)
\(pw := 2; cnt := 1; done := \text{false}\); \(\text{pow}\)
\(\{pw = 2^{cnt} > 1000 \text{ and } 2^{cnt-1} \leq 1000\}\)

Recursive Assumption:
\(\{pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = P\)
\(\text{pow}\)
\(\{\text{done} = (pw > 1000) \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = Q\)
Assume \(=\) has higher precedence than \(\text{ and }\).

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 \(\text{not}(\text{done})\) then
\(cnt := cnt + 1\); \(pw := 2 \times pw\); \(\text{pow}\) end if
\(\{\text{done} = (pw > 1000) \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = Q\)

We are allowed to use the recursive assumption when \(\text{pow}\) 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;
\(\{\text{done} = (pw > 1000) \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = S\)
Let \(B = \text{not}(\text{done})\)

Case 1: \(S \text{ and } B\)
\(S \text{ and } B \supset\)
\(\{\text{done} = \text{false} \text{ and } pw \leq 1000 \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} \supset\)
\(\{2^{cnt+1-1} \leq 1000 \text{ and } 2 \times pw = 2^{cnt+1}\}\)
cnt := cnt + 1;
\(\{2^{cnt-1} \leq 1000 \text{ and } 2 \times pw = 2^{cnt}\}\)
pw := 2 \times pw;
\(\{2^{cnt-1} \leq 1000 \text{ and } pw = 2^{cnt}\} = P\)
\(\text{pow}\)
\(\{\text{done} = (pw > 1000) \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = Q\)
by the recursion assumption.

Case 2 \(S \text{ and } \text{not}(B)\)
\(S \text{ and } \text{not}(B) \supset\)
\(\{\text{done} = (pw > 1000) \text{ and } pw = 2^{cnt} \text{ and } 2^{cnt-1} \leq 1000\} = Q\)
Now assemble the proof:

\[
\{ \text{true} \} \models \\
\{ 2=2 \} \\
\{ \text{pw}=2 \} \models \\
\{ \text{pw}=2 \text{ and } 1=1 \} \\
\text{cnt} := 1; \\
\{ \text{pw}=2 \text{ and } \text{cnt}=1 \} \models \\
\{ \text{pw}=2 \text{ and } \text{cnt}=1 \text{ and } \text{false}=\text{false} \} \\
\text{done} := \text{false} \\
\{ \text{pw}=2 \text{ and } \text{cnt}=1 \text{ and } \text{done}=\text{false} \} \models \\
\{ \text{pw}=2^{\text{cnt}} \text{ and } 2^{\text{cnt}-1} \leq 1000 \} = P \\
\text{pow} \\
\{ \text{done}(\text{pw}>1000) \text{ and } \text{pw}=2^{\text{cnt}} \}
\text{ and } 2^{\text{cnt}-1} \leq 1000 \} = Q
\]

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

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

Therefore, we may conclude:

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

which implies

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

---

**Recursive Procedure with a Parameter**

Recursion 1

\[
\forall f. (\{ P[F \rightarrow f] \} \text{ proc}(f) \{ Q[F \rightarrow f] \}) \models \\
\{ P[F \rightarrow E] \} \text{ proc}(E) \{ Q[F \rightarrow E] \}
\]

**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:

\[
\{ \text{num} = K > 0 \} = P[F \rightarrow E] \\
\text{count(num)} \\
\{ 10^{\text{ans}-1} \leq \text{num} < 10^{\text{ans}} \}. = Q[F \rightarrow E]
\]

“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

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

assuming as an induction hypotheses

\[
\forall f. (\{ f = K > 0 \} \text{ count}(f) \{ 10^{\text{ans}-1} \leq f < 10^{\text{ans}} \}),
\]

which is

\[
\forall f. (\{ P[F \rightarrow f] \} \text{ count}(f) \{ Q[F \rightarrow f] \})
\]
Use the If-Else rule.

**Case 1**  $m < 10$

\[
\begin{align*}
\{ & m = k > 0 \text{ and } m < 10 \} \implies \\
& \{ 10^{k-1} \leq m < 10^k \} \\
& \text{ans} = 1 \\
& \{ 10^{\text{ans}-1} \leq m < 10^{\text{ans}} \}
\end{align*}
\]

**Case 2**  $m \geq 10$

\[
\begin{align*}
\{ & m = k > 0 \text{ and } m \geq 10 \} \implies \\
& \{ m/10 = k/10 \text{ and } m/10 \geq 1 \} \implies \\
& \{ m/10 = k/10 \geq 1 > 0 \} \\
& \text{count}(m/10) \quad \text{-- use } f = m/10 \\
& \{ 10^{\text{ans}-1} \leq m/10 < 10^{\text{ans}} \} \implies \\
& \{ 10^{\text{ans}+1-1} \leq m < 10^{\text{ans}+1} \} \\
& \text{ans} := \text{ans}+1 \\
& \{ 10^{\text{ans}-1} \leq m < 10^{\text{ans}} \}
\end{align*}
\]

### 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 $\mathbb{N}$ ordered by $>$.  
2. Cartesian product $\mathbb{N} \times \mathbb{N}$ with a lexicographic ordering:
   \[
   <m_1, m_2> > <n_1, n_2> \\
   \text{if } [m_1 > n_1] \text{ or } [m_1 = n_1 \text{ and } m_2 > n_2].
   \]
3. The positive integers $\mathbb{P}$ ordered by the relation “properly divides”:
   \[
   m > n \text{ if } (\exists k[m = n \cdot k] \text{ and } m \neq n).
   \]

### 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
```
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**

\[
\begin{align*}
\{ & N \geq 0 \text{ and } A \geq 0 \} \\
& k := N; \ s := 1; \\
& \text{while} k > 0 \ \text{do} \\
& \quad s := A \cdot s; \\
& \quad k := k-1 \\
& \text{end while} \\
& \{ s = A^N \}
\end{align*}
\]

Take $W = \mathbb{N}$, the set of natural numbers ordered by $>$. Therefore, $m \in W$ if and only if $m \geq 0$.

Take $E = k$ as the termination expression.
The loop invariant $P$ is
\[
\{ k \geq 0 \text{ and } s \cdot A^k = A^N \}
\]

The conditions on the termination expression must hold at the location of the invariant.
The two conditions follow immediately:

a) $P \supset \{ k \geq 0 \text{ and } s \cdot A^k = A^N \} \supset E = k \in W$

b) $\{ P \text{ and } B \text{ and } E = D \} \supset \{ k \geq 0 \text{ and } s \cdot A^k = A^N \text{ and } k > 0 \text{ and } k = D \} \supset \{ k - 1 = D - 1 \}$

$s := A \cdot s; \ k := k-1$

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

What if $N \geq 0$ 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.

\[
\{ \text{num} = k > 0 \} \text{ count(num)} \{ 10^{\text{ans}-1} \leq \text{num} < 10^{\text{ans}} \}.
\]

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, $0 \leq \text{num} < 10$.

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

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