Derivation of Algorithms

Lecture 4
Formal Program Construction

COMP H 4018
Lecturer: Stephen Sheridan

How to prove a loop is correct

In this example we shall use only a loop with a single guard, i.e.

do B → S od

Note: P is the invariant and t is the bound function

Step 1: Annotate the code as follows

```
{P , bound t}
do B →
        {P ∧ B}
          S
        {P , proof 1}
od
{Q, proof, termination: proof }
```

Step 2: Prove each of the following:

Proof 0: "establish P" i.e. {R} S {P}
Proof 1: {P ∧ B} S {P}
Proof 2: [P ∧ ¬ B ⇒ Q]
Proof 3: t ≥ 0 and t decreasing.

Ex1 : Prove

```
|[ con N:int;  {N ≥ 0}
   var x : int;
  x:=0;
  do x ≠ N →
        x : = x + 1
  od
   {x=N}
]|
```

Introduce in variant P : $0 \le x \le N$
Bound function : N-x

```
|[ con N : int  {N ≥ 0}
  var x  :int;
  x :=0;
 {0 ≤ x ≤ N  ,  N-x}
  do x ≠ N →
        {P ∧ x ≠ N}
        x := x+1
        {P  ,  proof 1}
   od
  {x=N,  proof 2 : termination,  proof 3}
]|
```

Proof 0: $\{N \ge 0\}\ x:=0\ \{0 \le x \le N\}$

$$
\begin{array}{ll}
 & (0 \le x \le N)(x := 0) \\
\equiv & \{ \text{ substitution } \} \\
 & 0 \le 0 \le N \\
\equiv & \{ \text{ arithmetic } \} \\
 & 0 \le 0 \wedge 0 \le N \\
\equiv & \{ \text{ arithmetic } \} \\
 & \textbf{true} \wedge 0 \le N \\
\equiv & \{ \text{ true - false } \} \\
 & N \ge 0
\end{array}
$$

Proof 1: $\{ 0 \le x \le N \wedge x \ne N \}\ x:=x+1\ \{0 \le x \le N\}$

$$
\begin{array}{ll}
 & (0 \le x \le N)(x := x+1) \\
\equiv & \{ \text{ substitution } \} \\
 & 0 \le x + 1 \le N \\
\equiv & \{ \text{ inequality } \} \\
 & 0 \le x+1 \wedge x + 1 \le N \\
\Leftarrow & \{ \text{ given} \} \\
 & 0 \le x \le N \wedge x \ne N
\end{array}
$$

Proof 2: $[P \wedge \neg (x \ne N) \Rightarrow x = N]$

$$
\begin{array}{ll}
 & [0 \le x \le N \wedge \neg (x \ne N) \Rightarrow x = N] \\
\equiv & \{ \neg \neg \} \\
 & (0 \le x \le N \wedge (x = N) \Rightarrow x = N \\
\equiv & \{ \Rightarrow \} \\
 & \text{true}
\end{array}
$$

Proof 3: Termination

(1)
$$
\begin{array}{ll}
 & (N - x \ge 0)(x := 0) \\
\equiv & \{ \text{ substitution } \} \\
 & N - 0 \ge 0 \\
\Leftarrow & \{ \text{ given } \} \\
 & N \ge 0
\end{array}
$$

(2)
$$
\begin{array}{ll}
 & (N - x)(x : = x + 1) \\
\equiv & \{ \text{ substitution } \} \\
 & N - (x + 1) \\
\equiv & \{ \text{ arithmetic } \} \\
 & (N - x) - 1 \\
< & \{ \text{ arithmetic } \} \\
 & N -x
\end{array}
$$

Formal Programming Techniques

In this part of the course we look at methods for making programs using the formal proof techniques illustrated above. The idea is to develop the program and its proof of correctness in parallel. The formal proof techniques should be used as part of the construction process and should guide the development. The steps involved are:

1. Write down a formal specification of the problem, e.g.

```
|[ con N : int; { N ≥ 0 } f : array[0..N) of int;
  var sum : int;
   S
  { sum = (+j : 0 ≤ j < N : f.j) }
]|
```

2. Use one of the methods we will cover as part of this course to derive a solution for S such that the formal specification below holds.

```
|[ {P}
    S
    {Q}
]|
```

Method 1: Replacing Constants by variables

The idea is to replace a constant in the post condition with a variable to find the invariant P. The loop is then constructed under the invariance of P. The following examples illustrate the approach.

Example 1
Solve:

```
|[ con A, B :int {A ≥ 0 ∧ B ≥ 0};
  var y : int;
   exponentiation
   {y = A^B}
]|
```

Problem: Must weaken post-condition $y = A^B$ to derive an invariant.

Solution: Replace the constant B with a variable, say $x : int$. This gives

$$y = A^x$$

Post - condition now becomes

$$y = A^x \wedge x = B$$

Using method 1 we now have

invariant P: $y = A^x \wedge x \leq B$

guard : $\neg(x = B)$

$\{y = A^x \wedge x \leq B\}$
do $x \neq B \rightarrow S$ od
$\{y = A^x \wedge x \leq B \wedge \neg\, x \neq B\}$
$\{y = A^x \wedge x = B\}$
$\{y = A^B\}$

To Do !

{1} : Establish P.
{2} : Evaluate S while making progress.
{3} : Prove termination.

1: Establish P

By definition: $0^0 = 1$, $A^o = 1$.

$\therefore$ y, x : = 1, 0

Proof:

$$
\begin{aligned}
& (y = A^x \wedge x \leq B)(y,\ x := 1, 0) \\
\equiv\ & \{ \text{substitution} \} \\
& 1 = A^o \wedge 0 \leq B \\
\equiv\ & \{ \text{def 1, above} \} \\
& 1 = 1 \wedge 0 \leq B \\
\equiv\ & \{ \text{arithmetic, true - false} \} \\
& B \geq 0 \\
\Leftarrow\ & \{ \text{given} \} \\
& A \geq 0 \wedge B \geq 0
\end{aligned}
$$

2: Evaluate S: (loop body)

$$
\begin{aligned}
& \{y = A^x \wedge x \leq B \wedge x \neq B\} \\
& S \\
& \{y = A^x \wedge x \leq B\}
\end{aligned}
$$

Incrementing x by 1 gives

$$
\begin{aligned}
& y = A^{x+1} \wedge x + 1 \leq B \\
\equiv\ & \{ \text{arithmetic} \} \\
& y = A^x.A \wedge x + 1 \leq B
\end{aligned}
$$

This suggests the concurrent assignment

$$y, x := y * A,\ x + 1$$

Proof

$$
\begin{aligned}
& (y = A^x \wedge x \leq B)(y,\ x := A * y, x + 1) \\
\equiv\ & \{ \text{substitution} \} \\
& A * y = A^{x+1} \wedge x + 1 \leq B \\
\equiv\ & \{ \text{arithmetic} \} \\
& A * y = A^x * A \wedge x + 1 \leq B \\
\equiv\ & \{ \text{arithmetic} \} \\
& y = A^x \wedge x + 1 \leq B \\
\Leftarrow\ & \{ x \leq B \wedge x \neq B \} \\
& y = A^x \wedge x \leq B
\end{aligned}
$$

3: Termination

Bound function : B - x

A: $B - x \geq 0 \Leftarrow x := 0$

B:

$$
\begin{aligned}
& (B - x)(x := x + 1) \\
\equiv\ & \{ \text{substitution} \} \\
& B - (x - 1) \\
<\ & \{ \text{arithmetic} \} \\
& B - x.
\end{aligned}
$$

Example 2

Solve

```
|[ con N : int {N ≥ 0 ; f : array [0..N] of int;
   var x : int ;
   summation
   { x = ( + i : 0 ≤ i < N : f.i ) }
]|
```

The post condition { x = (+ i : 0 ≤ i < N : f.i) } has 2 constants : 0 and N.

Replace N with variable n : int giving invariants

P0: $x = (+ i : 0 \leq i < n : f.i)$

P1: $0 \leq n \leq N$

Outline solution

```
{ P0 ∧ P1 }
do n ≠ N → {P0 ∧ P1 ∧ n ≠ N}
             S
             {P0 ∧ P1}
od
{P0 ∧ P1 ∧ n = N}
```

1: Establish P0∧ P1

Summation over an empty range is 0

∴ x, n : = 0, 0

Proof :

(P0 ∧ P1)(x, n : = 0, 0)

≡ { def P0 ∧ P1 }

(x = (+ i : 0 ≤ i < n : f.i) ∧ 0 ≤ n ≤ N)(x, n : = 0, 0)

≡ { substitution }

0 = (+ i : 0 ≤ i < 0 : f.i) ∧ 0 ≤ 0 ≤ N

≡ { empty range }

0 = (+ i : **false** : f.i) ∧ 0 ≤ 0 ∧ 0 ≤ N

≡ { def }

0 = 0 ∧ true ∧ 0 ≤ N

⇐{ arithmetic , true - false }

N ≥ 0

2: Evaluate S

{P0 ∧ P1 ∧ n ≠ N} S { P0 ∧ P1}

Incrementing n by 1 in P0 gives

(+ i : 0 ≤ i < n : f.i)(n := n + 1)

≡ {substitution}

(+ i : 0 ≤ i < n + 1 : f.i)

≡ { split off i = n }

(+ i : 0 ≤ i < n : f.i) + f.n

≡ { P0 }

x + f.n

n := n + 1 ⇐ x ≠ N

This gives S:

{P0 ∧ P1 ∧ n ≠ N}

x : = x + f.n;

n : = n + 1

{P0 ∧ P1}

3: Termination

bound : N - n.

A. N - n ≥ 0 ⇐ n := 0

B. N - (n + 1) < N - n.

Proof:

(N - n)(n : = n + 1)

≡ { subtitution }

N - (n + 1)

< { arithmetic }

N - n

Program

|[con N : int {N ≥ 0} ; f : array [0..N) of int;

var x : int;

var n : int;

n, x : = 0, 0;

{invariant : P0 ∧ P1, bound : N - n}

do n ≠ N →

{P0 ∧ P1 ∧ n ≠ N}

x : = x + f.n

; n : = n + 1

{P0 ∧ P1}

od

{x = (+ i : 0 ≤ i < n : f.i) ∧ n = N}

{x = (+ i : 0 ≤ i < N : f.i}

]|