FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Eng.ª de Software

2008/2009

EXERCÍCIOS

1. Assumindo que x e y são inteiros, determine a pré-condição mais fraca dos triplos que se seguem.

a. {P} x := z+1; y := x+y; {y > 5}

wp(y:=x+y;,y>5) = x+y>5

wp(x:=z+1;,x+y>5) = z+1+y>5 = z+y>4

b. {P} y := x–y; x := x-y; y := y+x {x=Y ∧ y=X}

wp(y:=y+x, x=Y /\ y=X) = x=Y /\ y+x=X

wp(x:=x-y , x=Y /\ y+x=X) = x-y = Y /\ y+x-y = X

wp(y := x–y;, x-y = Y /\ y+x-y = X) = y = Y /\ x= X

c. {P} if (x>y) y:=x; {y=max(x,y)}

[x>y /\ wp(y:=x;,y=max(x,y))]\/[x<=y /\ y=max(x,y)]

[x>y /\ x=max(x,x)] \/ true

true

d. {P} if (x>=10) then x := x/2; else x := x+5; {x>=5 ∧ x<=15}

x>=10 /\ wp(x:=x/2;,x>=5 /\ x<=15) \/ x<10 /\wp(x:=x+5;,x>=5 /\x<=15)

x>=10 /\x>=10 /\x<=30 \/ x<10 /\x+5>=5 /\x+5<=15

10<=x<=30 \/ 0<=x<10

0<=x<=30

2. Prove que o seguinte triplo de Hoare é verdadeiro

a. {True} b:=2; a:=5+b; b:=a+2 {b>a}

wp(b:=a+2;,b>a) = a+2>a

wp(a:=5+b;,a+2>a) = 5+b+2>5+b

wp(b:=2;,7+b>5+b) = 7+2>5+2 = 9>7

True => 9>7

True => True

True

b. Escreva a pré, P, e a pós-condição, S, do programa que se segue:

```
{P}
While (i<a.length) {
    j:=i+1;
    while (j<a.length) {
            if (a[i] > a[j] {
                    s:=a[i]; a[i]:=a[j]; a[j]:=s;
            }
            j++;
    }
    i++;
}
{S}
```

```
{P} = a!=nil ∧ i=0;

{S} = IsSorted(a) and IsPermutation(a~, a);

IsSorted(s:seq of real) res: bool ==
        forall i,j in set inds s & i<j => s(i) <= s(j);

IsPermutation(s,t:seq of real) res:bool ==
        SeqToBag(s) = SeqToBag(t);

SeqToBag(s:seq of real) res:map real to nat ==
        {v |-> len [i | i in set inds s and s(i)=v] | v in set elems s};
```

c. Considere o programa da alínea anterior. Qual o invariante do ciclo mais interior desse programa? Formalize as obrigações de prova para correcção parcial deste ciclo.

```
while (j<a.length) {
        if (a[i] > a[j] {
                s:=a[i]; a[i]:=a[j]; a[j]:=s;
        }
        j++;
}
```

Invariante (I):
a[i] <= a[j-1] and j<=a.length and j>i
Pré-condição (P):
a != null and j=i+1 and i<a.length
Pós-condição (Q):
forall k in set inds a & i<k<a.length | a[i]<=a[k]

Para a correcção parcial temos que provar:

P=>I
a != null and j=i+1 and i<a.length and j>i => a[i] <= a[j-1] and j<=a.length

{I /\ C}S{I}
(considerando a condição do ciclo C=j<a.length e S o corpo do ciclo)
{a[i] <= a[j-1] and j<=a.length and j>i /\ j<a.length}

```
        if (a[i] > a[j] {
                s:=a[i]; a[i]:=a[j]; a[j]:=s;
        }
        j++;
```

{a[i] <= a[j-1] and j<=a.length and j>i }

(I /\ ~C) => Q
(a[i]<=a[j-1] and j<=a.length and j>i /\ j>=a.length) =>
forall k in set inds a & i<k<a.length | a[i]<=a[k]

3. Assumindo que x e y são inteiros, use o método da pré-condição mais fraca ("weakest precondition") para determinar se os triplos que se seguem se verificam.

a. {y+x < 8} y := x–y; x := y+1; y := x-1 {y < x+2}

Weakest precondition

wp(y:=x-1;,y<x+2) = x-1<x+2 = true

wp(x:=y+1;,x-x<3) = y+1-y-1<3

wp(y:=x-y;,y-y<3) = x-y-x+y<3

y+x<8 => true?

true

b. {x>a ∧ y>b} t:=x; x:=x+y; y:=t {x=a+b ∧ y=a}

wp(y:=t;,x=a+b /\ y=a) = x=a+b /\ t=a

wp(x:=x+y;, x=a+b /\ t=a) = x+y=a+b /\ t=a

wp(t:=x;, x+y=a+b /\ t=a) = x+y=a+b /\ x=a = x=a /\ y=b

Para provar que o triplo é verdade teria que se provar a seguinte implicação:

x>a /\ y>b => x=a /\ y=b

O que é falso, por exemplo:

se x for a+1 e y for y+1 no final de executar o programa x=a+b+2 e y=a+1 o que torna a pós-condição falsa

4. Assumindo que x, y, i, j, k, m são inteiros, calcule a pré-condição mais fraca de

a. {P} i := x; j := y; k := i + j; m := k / 2; {0<=m<=20}

Solução = P ≡ (0 <= x + y <= 40)

b. Verifique o seguinte triplo de Hoare

{x >0 /\ x<100} y:=1; if (x>50) then y:=0; {y=0 \/ y=1}

```
Solução = wp(y:=1;,wp(if(x>50) then y:=0;, y=0 \/ y=1))
          wp(y:=1;,[x>50 /\ wp(y:=0;,y=0 \/ y=1)] \/ [x<=50 /\(y=0 \/ y=1)])
          wp(y:=1;,(x>50 /\ true) \/ (x<=50 /\ (y=0 \/ y=1)))
          wp(y:=1;, x>50 \/ (x<=50 /\ (y=0 \/ y=1)))
          x>50 \/ x >= 50

x>0 /\ x <100 => x>50 \/ x<=50
false
```

c. Assuma o seguinte triplo de hoare:

{ i ≥ 0 } while i > 0 do i := i−1 { i = 0 }

i. Indique qual o invariante do ciclo.

Solução = i >= 0

ii. Como demonstraria a correcção total?

Dado while b do C, a invariante do ciclo (I) é uma qualquer expressão onde {I /\ b} C {I} se verifica. Neste caso, assumindo que I = i ≥ 0, verificamos demonstrando que:

wp(i := i - 1, i ≥ 0) = i - 1 ≥ 0 <=> i ≥ 1,
efectivamente, i ≥ 0 /\ i >0→ i ≥ 1.

A correcção parcial de um triplo {X} while b do C {Y} e invariante I, verifica-se se:

X → I (i.e., a pré-condição implica a Invariante),
¬b /\ I → Y (e.g., a negação da condição e a Invariante implicam a pós-condição).

No primeiro caso, é trivial verificar que i ≥ 0 → i ≥ 0.
No segundo caso, também é trivial verificar que, se i ≤ 0 /\ i ≥ 0, então i = 0.

A correcção total de um triplo {X} while b do C {Y} e invariante I, verifica-se adicionalmente aos pontos anteriores, se a variante (V) decrescer em cada iteração. Assumindo,

V = i /\ i>=0 e em cada iteração, i := i -1, é trivial verificar essa asserção.

5. Considere o seguinte programa em que todas as variáveis representam inteiros:

```
{Q}
i := x div y;
j := i * 100;
if (j >= 48) j := 50; else j := 45;
{50 <= j <= 100}
```

a.1. Calcule a pré-condição mais fraca (Q).

[j>=48 /\ wp(j:=50;.50<=j<=100] \/ [j<48 /\ wp(j:=45;,50<=j<=100]=

(j>=48 /\ true)\/ False=

j>=48

wp(j:=i*100;,j>=48) = i*100>48 = i>48/100

wp(i:=x div y;,i>48/100) = x div y > 48/100

a.2. Demonstre que {x = y /\ x > 0} é uma pré-condição válida, embora não a mais fraca.

b. Considere o seguinte troço de um programa, em que *A* e *B* são vectores e *i* e *n* são inteiros:

```
while (i < n) {
      B[n - i - 1] := A[i];
       i := i + 1;
}
```

b.1. Escreva a pré e a pós-condição deste programa.

b.2. Qual o invariante do ciclo presente no programa? Qual é a sua função variante? Formalize as obrigações de prova para correcção total desse ciclo.

6. Indique a pré-condição mais fraca de

```
wp (i=1;while (i < len a and a(i)>a(i+1)) i++;, a(i) = Min(a))
```

Nota: a função Min determina o elemento mínimo de um vector de inteiros.

```
exists i in set inds a &
a[i]=Min(a) and forall k : nat1 & k>0 and k<i and a[k]>a[k+1];
```

7. Qual é o invariante do ciclo que se segue?

```
{x >= 0}
y := 1;
z := 0;
while (z != x) {
      z := z + 1;
      y := y * z
}
{z = x and y = x!}

Invariante: y=z!  /\ z<=x
```

```
Prova de correcção parcial
(1) O invariante verifica-se antes do ciclo
y=z! /\ z<=x
porque
1=0! /\ 0<=0
```

Trivial

(2) O invariante verifica-se depois do ciclo
~C /\ Inv => Post
~(z!=x) /\ y=z! /\ z<=x => z=x /\ y=x!
z=x /\ y=z! /\ z<=x => z=x /\ y=x!
z=x /\ y=z! => z=x /\ y=x!
z=x /\ y=x! => z=x /\ y=x!
true

(3) O invariante verifica-se durante o ciclo
Se y=z!/\z<=x se verifica na iteração m então também se verifica na iteração m+1
Notação: y_m = valor de y depois de m iterações,
z_m = valor de z depois de m iterações
Provar: if $y_m = z_m!$ /\ $z_m<=x$ then $y_{m+1} = z_{m+1}!$ /\ $z_{m+1}<=x$
$z_{m+1}<=x$ verdade porque $z_m<x$
Olhando para o ciclo, pode-se ver que
$y_{m+1}=y_m*z_{m+1}$
Pela hipótese, $y_m=z_m!$, então $y_{m+1}=z_m!*z_{m+1} = z_{m+1}!$

8. Qual é o invariante do ciclo?

```
public Remainder: ( nat  * nat1 ) ==> nat
  Remainder(X,Y) ==
  (  dcl Q: nat := 0;
     dcl R: nat := X;
     while (R >= Y) do
       R = R – Y;
       Q = Q + 1;
     return R;
```

Pré-condição: X>=0 /\ Y>0, a correcção total não existe para Y=0
Invariante: (X = Q * Y + R) /\ (0 <= R) , guard R <= Y
Pós-condição: X=Q*Y+R /\ 0<=R<Y

Prova
1 {x=Q*y+x /\ x>=0}R=x{x=Q*y+R/\R>=0}
2 {x=0*y+x/\x>=0}Q=0{x=Q*y+x/\x>=0}
3 {x=0*y+x/\x>=0}Q=0;R=x{x=Q*y+R/\x>=0}
4 {x=(Q+1)*y+R/\R>=0}Q=Q+1{x=Q*y+R/\R>=0}
5 {x=(Q+1)*y+R-y/\R-y>=0}R=R-y{x=(Q+1)*y+R/\R>=0}
6 {x=(Q+1)*y+R-y/\R-y>=0}R=R-y;Q=Q+1{x=Q*y+R/\R>=0}
7 (x=Q*y+R/\R>=0/\R>=y)->(x=(Q+1)*y+R-y/\R-y>=0)
8 {x=Q*y+R/\R>=0/\R>=y}R=R-y;Q=Q+1{x=Q*y+R/\R>=0}
9 {x=Q*y+R/\R>=0}while(R>=y){R=R-y;Q=Q+1}{x=Q*y+R/\R>=0 /\R<y} (consequência 6,7)
10 {x=0*y+x/\x>=0}Remainder{x=Q*y+R/\R>=0/\R<y} (composição 3,9)
11 (x>=0 /\ y>=0)->(x=0*y+x/\x>=0), para mostar que a pré-condição implica a wp
12 {x=0*y+x/\x>=0}Remainder{x=Q*y+R/\R>=0/\R<y} consequência

9. Considere a função que calcula o máximo divisor comum (mdc) entre dois números. Qual é o invariante do ciclo que se segue?

```
{ m>0 ∧ n>0 }
 x := m;
 y := n;
 while (x != y) {
      if (x > y) then x := x-y;
       else y := y-x;
 }
 result := x
{result = mdc(m,n)}
```

```
Propriedades de mdc:
      x>y => mdc(x,y) = mdc(x-y,y)
      mdc(x,y) = mdc(y,x)
      mdc(x,x) = x
Invariante: mdc(m,n) = mdc(x,y) /\ x>0 /\ y>0
```

GCD Proof

This is an example of a program to compute the greatest common divisor (GCD) of two positive integers – this is the largest number that is a whole divisor of each number. In this proof we rely on the following properties of GCD without proving them

```
X>Y=>GCD(X,Y) = GCD(X–Y,Y)
GCD(X,Y) = GCD(Y,X)
GCD(X,X) = X
```

In this example, only a proof of partial correctness of the program is provided. A separate analysis is required to see that the program is totally correct, that is, that it halts for all values that satisfy the pre-condition. This is shown by noticing that one of the inner loops will keep iterating as long as the values of A and B are not equal, and each such iteration must reduce the absolute difference of these values. Hence the total number of iterations of the inner and the outer loops can be no more than this difference.

To prove partial correctness of

```
{ X > 0 ∧Y > 0 }
A := X; B := Y;
{Inv}
while A<>B do
begin
while A > B do A := A–B;
while B > A do B := B–A
end
{ A = GCD(X,Y) }
```

we first need to formulate a loop invariant Inv. The loop invariant characterizes the “approximating strategy” of this program. In this case, that strategy is to preserve GCD(X,Y) = GCD(A,B) at all times, while always decreasing the absolute difference of A and B. Therefore, when finally A = B and all loops terminate, GCD(X,Y) = GCD(A,B) = GCD(A,A) = A. The loop invariant Inv is taken to be GCD(X,Y) = GCD(A,B) ∧ A>0 ∧ B>0.

Step 1: (actually it’s three steps that we abbreviate to one):

Clearly,

```
{X>0 ∧ Y>0}
A := X; B := Y;
{Inv}
```

by two applications of the axiom of assignment, plus the sequential rule. Since this GCD program involves two nested loops, we need to formulate the loop invariants for these loops in order to complete the proof. Fortunately, the same loop invariant suffices for all the loops.

```
{Inv}
while A<>B do
begin
{Inv}
while A > B do A := A–B;
{Inv}
while B > A do B := B–A
end
{A = GCD(X,Y)}
```

Step 2:

```
{Inv}
while A > B do A := A–B;
```

{Inv}

is proven by step 2A and the while rule since the post-condition Inv ∧ A<=B can be weakened to Inv.

Step 2A:

{Inv ∧ A>B } A:=A–B {Inv}

by the axiom of assignment since

Inv[A->A–B] ≡ GCD(X,Y) = GCD(A–B,B) ∧ A–B>0 ∧ B>0

which is logically equivalent to Inv ∧ A>B.

Step 3:

{Inv}
while B>A do B:=B–A
{Inv}

is proven by step 3A and the while rule since the post-condition Inv ∧ B<=A can be weakened to Inv.

Step 3A:

{Inv ∧ B>A} B := B–A {Inv}

by the axiom of assignment since

Inv [B -> B–A] ≡ GCD(X,Y) = GCD(A,B–A) ∧ A>0 ∧ B–A>0

which is logically equivalent to Inv ∧ B>A.

Step 4:

By Steps 2 and 3 and the sequential execution rule

{Inv}
begin
while A > B do A := A–B;
while B > A do B := B–A
end
{Inv}

Step 5:

By Step 4 and the while rule

{Inv}
while A<>B do
begin
while A>B do A := A–B;
while B>A do B := B–A
end
{Inv ∧ A=B }

Step 6:

Inv ∧ A=B ≡ (GCD(X,Y) = GCD(A,B) ∧ A>0 ∧ B>0 ∧ A=B) => A=GCD(X,Y), so by Step 5 and weakening the post-condition

{Inv}
while A<>B do
begin
while A > B do A := A–B;
while B > A do B := B–A
end
{ A = GCD(X,Y) }

Step 7:

By Steps 1 and 6 and the sequential rule, the program is proven.