Ficha 1

Algoritmos e Complexidade CC

Análise de correcção de programas

1 Especificação Formal de Programas

1. Discuta a validade dos seguintes triplos de Hoare:
   (a) $\{\mathtt{j} = \mathtt{a}\}$ $\mathtt{j} := \mathtt{j} + 1$ $\{\mathtt{a} = \mathtt{j} + 1\}$
   (b) $\{\mathtt{i} = \mathtt{j}\}$ $\mathtt{i} := \mathtt{j} + \mathtt{i}$ $\{\mathtt{i} > \mathtt{j}\}$
   (c) $\{\mathtt{j} = \mathtt{a} + \mathtt{b}\}$ $\mathtt{i} := \mathtt{b};\ \mathtt{j} := \mathtt{a}$ $\{\mathtt{j} = 2 * \mathtt{a}\}$
   (d) $\{\mathtt{i} > \mathtt{j}\}$ $\mathtt{j} := \mathtt{i} + 1;\ \mathtt{i} := \mathtt{j} + 1$ $\{\mathtt{i} > \mathtt{j}\}$
   (e) $\{\mathtt{i}\ != \ \mathtt{j}\}$ $\mathtt{if}\ (\mathtt{i} > \mathtt{j})\ \mathtt{then}\ \mathtt{m} := \mathtt{i} - \mathtt{j}\ \mathtt{else}\ \mathtt{m} := \mathtt{j} - \mathtt{i}$ $\{\mathtt{m} > 0\}$
   (f) $\{\mathtt{i} = 3 * \mathtt{j}\}$ $\mathtt{if}\ (\mathtt{i} > \mathtt{j})\ \mathtt{then}\ \mathtt{m} := \mathtt{i} - \mathtt{j}\ \mathtt{else}\ \mathtt{m} := \mathtt{j} - \mathtt{i}$ $\{\mathtt{m} - 2 * \mathtt{j} = 0\}$
   (g) $\{\mathtt{x} = \mathtt{b}\}$ $\mathtt{while}\ (\mathtt{x} > \mathtt{a})\ \mathtt{x} := \mathtt{x} - 1$ $\{\mathtt{b} = \mathtt{a}\}$
2. Escreva especificações (contratos para triplos de Hoare) para os seguintes programas:
   (a) Um programa que coloca na variável `Z` a soma dos valores das variáveis `X` e `Y`.
   (b) Um programa que coloca na variável `Z` o máximo divisor comum das variáveis `X` e `Y`.
   (c) Um programa que coloca na variável `Z` o mínimo múltiplo comum das variáveis `X` e `Y`.
   (d) Um programa que recebe dois arrays `A` e `B` como parâmetros, e verifica que eles têm um elemento em comum.
   (e) Um programa que recebe dois arrays `A` e `B` como parâmetros, e retorna o prefixo mais longo que os dois têm em comum.
   (f) Um programa que ordena um array `A`.
3. Sejam P, Q dois predicados arbitrários, e seja S um programa arbitrário. O que significa a validade dos seguintes triplos (tendo em conta que a notação [ ] representa correcção total):
   (a) $\{P\}$ `S` $\{\mathsf{true}\}$
   (b) $[P]$ `S` $[\mathsf{true}]$
   (c) $\{P\}$ `S` $\{\mathsf{false}\}$
   (d) $[P]$ `S` $[\mathsf{false}]$
   (e) $\{\mathtt{false}\}$ `S` $\{Q\}$

(f) $[\texttt{false}]\ \texttt{S}\ [Q]$

(g) $\{\texttt{true}\}\ \texttt{S}\ \{Q\}$

(h) $[\texttt{true}]\ \texttt{S}\ [Q]$

4. Dê, para cada programa, uma pré-condição que seja suficiente para garantir a pós-condição. Sugestão: tente encontrar a pré-condição mais fraca que permita atingir esse objectivo.

   (a) $\{?\}\ \texttt{x} := \texttt{x} + \texttt{1};\ \texttt{s} := \texttt{s} + \texttt{x}\ \{\texttt{s} > 0\}$

   (b) $\{?\}\ \texttt{Yold} := \texttt{y};\ \texttt{if}\ (\texttt{I} = 0)\ \texttt{then}\ \texttt{y} := 0\ \texttt{else}\ \texttt{y} := \texttt{y}/\texttt{x}\ \{\texttt{x} < \texttt{Yold}\}$

5. Considere o seguinte programa, para o qual vamos utilizar a pós-condição $\texttt{r} < \texttt{y} \quad \wedge$ $\texttt{x} = \texttt{r} + (\texttt{y} * \texttt{q})$.

```
r:=x;
q:=0;
while y <= r
   { r:=r-y; q:=q+1 }
```

   (a) Escreva uma pré-condição para uma especificação de correcção parcial para este programa.

   (b) Escreva uma pré-condição para uma especificação de correcção total para este programa.

2 Regras de Inferência da Lógica de Hoare

1. Considere as seguintes regras de inferência

Consequência

$$\frac{P \Rightarrow P' \quad \{\,P'\,\}\,S\,\{\,Q'\,\} \quad Q' \Rightarrow Q}{\{\,P\,\}\,S\,\{\,Q\,\}}\ (\Rightarrow)$$

Atribuição

$$\frac{}{\{\,Q[x \setminus E]\,\}\ x := E\ \{\,Q\,\}}\ (:=)$$

Sequência

$$\frac{\{\,P\,\}\,S_1\,\{\,R\,\} \quad \{\,R\,\}\,S_2\,\{\,Q\,\}}{\{\,P\,\}\,S_1\,;\,S_2\,\{\,Q\,\}}\ (;)$$

Condicional

$$\frac{\{\,P \wedge c\,\}\,S_1\,\{\,R\,\} \quad \{\,P \wedge \neg c\,\}\,S_2\,\{\,Q\,\}}{\{\,P\,\}\ if\ (c)\ then\ S_1\ else\ S_2\ \{\,Q\,\}}\ (;)$$

Ciclo

$$\frac{\{\,I \wedge c\,\}\ S\ \{\,I\,\}}{\{\,I\,\}\ \mathit{while}\ (c)\ S\ \ \{\,I \wedge \neg c\,\}}\ \text{(while)}$$

Usando estas regras apresente regras derivadas cujas conclusões sejam:

(a) $\{\,P\,\}\ x := E\ \{\,Q\,\}$

(b) $\{\,P\,\}\ \mathit{while}\ (c)\ S\ \ \{\,Q\,\}$

(c) $\{\,P\,\}\ x := E_1\ ;\ \mathit{while}\ (c)\ \{S\ ;\ x := E_2\}\ \ \{\,Q\,\}$. Note que esta última construção corresponde ao comando `for` do C.

2. Apresente uma prova que justifique cada um dos seguintes triplos de Hoare:

(a) $\{\mathtt{i} > \mathtt{j}\}\ \mathtt{j} := \mathtt{i} + 1;\ \mathtt{i} := \mathtt{j} + 1\ \{\mathtt{i} > \mathtt{j}\}$

(b) $\{\mathtt{i}\ != \ \mathtt{j}\}\ \mathtt{if}\ (\mathtt{i} > \mathtt{j})\ \mathtt{then}\ \mathtt{m} := \mathtt{i} - \mathtt{j}\ \mathtt{else}\ \mathtt{m} := \mathtt{j} - \mathtt{i}\ \{\mathtt{m} > 0\}$

(c) $\{\mathtt{a} > \mathtt{b}\}\ \mathtt{m} := 1; \mathtt{n} := \mathtt{a} - \mathtt{b}\ \{\mathtt{m} * \mathtt{n} > 0\}$

(d) $\{\mathtt{s} = 2^{\mathtt{i}}\}\ \mathtt{i} := \mathtt{i} + 1; \mathtt{s} := \mathtt{s} * 2\ \{\mathtt{s} = 2^{\mathtt{i}}\}$

(e) $\{\mathtt{True}\}\ \mathtt{if}\ (\mathtt{i} < \mathtt{j})\ \mathtt{then}\ \mathtt{min} := \mathtt{i}\ \mathtt{else}\ \mathtt{min} := \mathtt{j}\ \{\mathtt{min} \leq \mathtt{i} \wedge \mathtt{min} \leq \mathtt{j}\}$

(f) $\{\mathtt{i} > 0 \wedge \mathtt{j} > 0\}\ \mathtt{if}\ (\mathtt{i} < \mathtt{j})\ \mathtt{then}\ \mathtt{min} := \mathtt{i}\ \mathtt{else}\ \mathtt{min} := \mathtt{j}\ \{\mathtt{min} > 0\}$

3 Introdução aos invariantes de ciclo

1. Encontre um invariante para o ciclo no programa seguinte e prove a sua preservação. O invariante deverá implicar que, no caso de terminação do ciclo, a pós-condição $\mathtt{s} = 2^{\mathtt{i}}$ é satisfeita.

```
while i < n {
   i:=i+1; s:=s*2
 }
```

2. Encontre um invariante para o ciclo no programa seguinte e prove a sua preservação. O invariante deverá implicar que, no caso de terminação do ciclo, a pós-condição $\mathtt{r} < \mathtt{y}\ \wedge\ \mathtt{x} = \mathtt{r} + (\mathtt{y} * \mathtt{q})$ é satisfeita.

```
r:=x;
q:=0;
while y <= r {
   r:=r-y; q:=q+1
}
```

3. Considere o seguinte algoritmo de multiplicação de dois números inteiros.

```
res := 0;
while (y>0) {
  res := res + x;
  y = y-1
 }
```

(a) Escreva predicados que descrevam a pré e pós condição deste algoritmo. Procure que a pré-condição garanta a terminação do algoritmo.

(b) Apresente um invariante de ciclo que, no caso da terminação do ciclo, implique a pós-condição que identificou na alínea anterior.

(c) Prove a preservação do invariante de ciclo.

4. Repita as duas últimas alíneas do exercício anterior agora para o seguinte algoritmo que toma partido de que a divisão e multiplicação por 2 são operações muito eficientes (trata-se, em representação binária, de *shifts*).

```
res := 0;
while (y>0) {
   if (y % 2 != 0) then { y: =y - 1; res := res + x}
   x := x*2;
   y := y/2;
  }
```

5. Para cada um dos programas seguintes determine um variante e um invariante que lhe permita (apenas) provar a terminação dos ciclos em causa. Determine ainda a pré-condição necessária a que o ciclo termine de facto.

(a)
```
while (i < n)
     { i:=i+1; s:=s*2
     }
```

(b)
```
r:=x;
q:=0;
while (y <= r) {
      r:=r-y; q:=q+1
     }
```

(c)
```
res := 0;
while (y>0) {
      res := res + x;
      y = y-1
   }
```

(d)
```
res := 0;
while (y>0) {
      if (y % 2 != 0) then {
            y := y - 1;
            res := res + x
           }
      x := x*2;
      y := y/2
     }
```

(e)
```
Min = A[1][1];
i := 1;
while (i<=n) {
```

```
            j := 1;
            while (j<=n) {
                  if (Min > A[i][j])
                  then Min := A[i][j]
                  j := j + 1
               }
            i := i+1
        }
```

(f)
```
Min = A[1][1];
i := 1; j:= 2
while (i<=n) {
      if (Min > A[i][j])
      then Min := A[i][j]
      j := j + 1;
      if (j > n) then {
            j := 1; i:= i+1
        }
   }
```

6. Determine as condições de verificação necessárias para provar a correcção total dos seguintes algoritmos anotados (em comentário).

(a)
```
// x >= 0 && y > 0
r := x;
// x >= 0 && y > 0  && r == x
while (r>=y) {
   // r >= 0 && (\exists q >= 0 : q * y + r = x) ; r
   r := r - y;
// 0 <= r < y && (\exists q >= 0 : q * y + r = x)
```

(b)
```
// n >= 0
k = 1; i=0;
// n > 0 && k = 1
while (i < n) {
      // i <= n 0 && k = i! ; n-i
      i:=i+1; k:=k*i
// k = n!
```

(c)
```
// n > 0
i = n-1; k = 0;
// n > 0 && i = n-1 && k = 0
while (i > 0) {
      // i >= 0 && k < n ; i
      if (a[i] < a[i-1]) then {
            t:=a[i]; a[i]:=a[i-1];a[i-1]:=t;
            k=k+1
         }
      i:=i-1;
   }
```

```
// k < n
```

7. Considere os seguintes (extractos de) programas que calculam o factorial (da variável x)

```
f = 1;
while (x>0) {
    f = f*x; x=x-1;
}
```
Programa 1

```
f = 1; i=0;
while (i<x) {
    i=i+1;f = f*i;
}
```
Programa 2

   (a) Apresente uma especificação (pré e pós condições) que traduza o enunciado informal apresentado acima.
   (b) Para cada um dos programas acima, apresente um invariante para a prova de correcção (parcial).

8. Considere os seguintes programas (anotados em comentário). Apresente as condições de verificação necessárias à prova da correcção parcial destes programas.

   (a)
```
// n = n0 > 0
x=1; y=0;
// n = n0 > 0 /\ x=1 /\ y = 0
while (n>1) {
    //  x = fib (n0 - n + 1) /\ y = fib (n0 - n) /\ n >= 1
        x = x+y; y = x-y; n = n-1;
    }
// x = fib (n0)
```
   (b)
```
// x == x0 >= 0
a = x; b = 0;
// (x == x0 >= 0) /\ (b <= sqrt x0 <= a)
while ((a-b) > epsilon) {
    // (x == x0 >= 0) /\ (b <= sqrt x0 <= a)
    m = (a+b)/2;
    // (x == x0 >= 0) /\ (b <= sqrt x0 <= a) /\ (m = (a+b)/2)
    if (m^2 > x) then a = m; else b = m;
}
// b <= sqrt x0 <= a
```
   (c)
```
// (exists (i in [0..n-1]) v[i] == x)
k = 0;
// (exists (i in [k..n-1]) v[i] == x)
while (v[k] != x)
    // (exists (i in [k..n-1]) v[i] == x)
    k=k+1
// v[k] == x
```
   (d)
```
// s = 0 && i = 0
while (i < N) {
  // s = sum (k=0..i-1) a[k] && i <= N
  s = s + a[i];
  i = i+1;
}
// s = sum (k=0..N-1) a[k]
```

```
if (s > 10) then res = 1;
else res = 0;
// sum (k=0..N-1) a[k] > 10 <==> res = 1
```

(e)
```
// True
v = 0;
i = 0;
// v = 0 && i = 0
while (i<=N) {
  // v = sum (k=0..i-1) b[k] * 2^(N-k)  && i <= N+1
  v = v*2 + b[i];
  i = i+1
}
// v = sum (k=0..N) b[k] * 2^(N-k)
```

(f)
```
// (a = a0 > 0) /\ (b = b0 > 0)
while (a != b)
    // mdc(a0,b0) = mdc(a,b)
    if (a > b) then a = a - b;
    else b = b - a;
// a = mdc(a0,b0)
```