

# Ficha 1

## Algoritmos e Complexidade

### Análise de correcção de programas

## 1 Especificações

1. Descreva por palavras as seguintes especificações.

- (a)  $\begin{cases} \text{pré-condição: } & x == x_0 \geq 0 \wedge y == y_0 > 0 \\ \text{pós-condição: } & 0 \leq r < y_0 \wedge q * y_0 + r == x_0 \end{cases}$
- (b)  $\begin{cases} \text{pré-condição: } & x == x_0 \geq 0 \wedge y == y_0 > 0 \\ \text{pós-condição: } & 0 \leq r < y_0 \wedge (\exists_{q \geq 0}. q * y_0 + r == x_0) \\ & (x, y, q, r \text{ variáveis de tipo inteiro}) \end{cases}$
- (c)  $\begin{cases} \text{pré-condição: } & x == x_0 \wedge y == y_0 \\ \text{pós-condição: } & (x == x_0 \vee x == y_0) \wedge x \geq x_0 \wedge x \geq y_0 \end{cases}$
- (d)  $\begin{cases} \text{pré-condição: } & x == x_0 \geq 0 \wedge e == e_0 > 0 \\ \text{pós-condição: } & |r * r - x_0| < e_0 \\ & (x, e, r \text{ variáveis de vírgula flutuante}) \end{cases}$
- (e)  $\begin{cases} \text{pré-condição: } & \forall_{0 \leq i < N}. A[i] == a_i \\ \text{pós-condição: } & 0 \leq p < N \wedge \forall_{0 \leq i < N}. (A[i] == a_i \wedge A[p] \leq a_i) \\ & (A \text{ array de tipo numérico; } N \text{ constante de tipo inteiro, } p \text{ variável de tipo inteiro}) \end{cases}$

2. Escreva especificações (pré- e pós-condições) para os seguintes problemas:

- Um programa que, coloca na variável **r** a soma dos valores (iniciais) das variáveis **x** e **y**.
- Um programa que, para valores não negativos da variável **y**, coloca na variável **r** o produto dos valores (iniciais) das variáveis **x** e **y**.
- Um programa que, para valores não negativos da variável **y**, coloca na variável **r** o produto dos valores (iniciais) das variáveis **x** e **y**, sem alterar os valores dessas variáveis.
- Um programa que coloca na variável **r** o mínimo múltiplo comum das variáveis **x** e **y**.
- Um programa que recebe dois arrays **A** e **B** com **N** elementos como parâmetros, e calcula o comprimento do prefixo mais longo que os dois têm em comum.

- (f) Um programa que recebe dois arrays A e B (com N elementos) como parâmetros, e verifica se eles têm um elemento em comum.
3. Apresente especificações para cada uma das funções (pré-definidas em C) sobre *strings*. Sempre que necessário, use a notação  $\text{len}(s)$  para se referir ao comprimento da string  $s$ .
- `char *strcpy (char *s1, char *s2)` que copia a string  $s2$  para  $s1$ .
  - `char *strcat (char *s1, char *s2)` que acrescenta a string  $s2$  a  $s1$ .
  - `int strcmp (char *s1, char *s2)` que compara as strings  $s1$  e  $s2$ .
  - `char *strstr (char *s1, char *s2)` que determina a posição onde a string  $s2$  ocorre em  $s1$ .

## 2 Invariantes de ciclo

Para cada um dos programas seguintes, apresente as condições de estabelecimento, preservação e utilidade do invariante fornecido.

1. Cálculo da divisão inteira e do resto da divisão inteira entre dois positivos
- ```
// x == x0 >= 0  &&  y == y0 > 0
r = x;
q = 0;
while (y <= r) {
    // r >= 0  &&  y == y0
    // &&  q*y0 + r == x0
    r = r-y;
    q = q+1;
}
// 0 <= r < y0  &&  q*y0 + r == x0
```

2. A sequência de Fibonacci  $\{F_n\}_{n \geq 0}$  define-se como:
- $$F_i = \begin{cases} i & \text{se } i < 2 \\ F_{i-1} + F_{i-2} & \text{se } i \geq 2 \end{cases}$$
- O programa ao lado calcula o  $n$ -ésimo número de Fibonacci.
- ```
// n == n0 > 0
i = 1; r = 1; s = 0;
while (i < n) {
    // n==n0 && i<=n
    // && r == F(i)
    // && s == F(i-1)
    r = r+s;
    s = r-s;
    i = i+1;
}
// r == F(n0)
```

3. O algoritmo de Euclides para o cálculo do máximo divisor comum (mdc) entre dois inteiros positivos baseia-se em duas propriedades fundamentais:

- $\forall_x. mdc(x, x) = x$
- $\forall_{x,y}. mdc(x, y) = mdc(x + y, y) = mdc(x, x + y)$

Use estas propriedades para provar a correcção do programa para calcular o mdc de dois inteiros positivos.

$$\left\{ \begin{array}{l} \text{pré-condição: } a == a_0 > 0 \wedge b == b_0 > 0 \\ \text{pós-condição: } a == \text{mdc}(a_0, b_0) \end{array} \right.$$

Use como invariante o predicado

$$I \doteq \text{mdc}(a, b) == \text{mdc}(a_0, b_0)$$

```
while (a != b)
    if (a > b) a = a-b;
    else b = b-a;
```

### 3 Determinação de invariantes de ciclo

1. Considere a seguinte especificação de um programa que calcula o somatório dos elementos de um vector.

$$\left\{ \begin{array}{l} \text{pré-condição: } N \geq 0 \wedge (\forall_{0 \leq i < N}. A[i] == a_i) \\ \text{pós-condição: } s == \sum_{i=0}^{N-1} a_i \end{array} \right.$$

Encontre invariantes de ciclo que lhe permitam provar a correcção parcial dos seguintes programas (face a essa especificação). Apresente ainda as condições de estabelecimento, preservação e utilidade de cada um desses invariantes.

(a)  $s = 0; p = 0;$   
 $\text{while } (p < N) \{$   
 $\quad s = s + A[p]; p = p + 1;$   
 $\}$

(b)  $s = 0; p = N;$   
 $\text{while } (p > 0) \{$   
 $\quad p = p - 1; s = s + A[p];$   
 $\}$

2. Considere a seguinte especificação de um programa que calcula o produto de dois números inteiros.

$$\left\{ \begin{array}{l} \text{pré-condição: } x = x_0 \wedge y = y_0 \geq 0 \\ \text{pós-condição: } r = x_0 * y_0 \end{array} \right.$$

Encontre invariantes de ciclo que lhe permitam provar a correcção parcial dos seguintes programas (face a essa especificação). Apresente ainda as condições de estabelecimento, preservação e utilidade de cada um desses invariantes.

(a)  $r = 0; i = 0;$   
 $\text{while } (i < y) \{$   
 $\quad r = r + x; i = i + 1$   
 $\}$

(b)  $r = 0;$   
 $\text{while } (y > 0) \{$   
 $\quad r = r + x; y = y - 1$   
 $\}$

(c)  $r = 0;$   
 $\text{while } (y > 0) \{$   
 $\quad \text{if } (y \% 2 != 0) \{$   
 $\quad \quad y = y - 1; r = r + x$   
 $\quad \}$   
 $\quad x = x * 2; y = y / 2$

}

3. Considere a seguinte especificação de um programa que calcula o quadrado de um número inteiro.

$$\begin{cases} \text{pré-condição: } & x = x_0 \geq 0 \\ \text{pós-condição: } & r = x_0^2 \end{cases}$$

Encontre invariantes de ciclo que lhe permitam provar a correcção parcial dos seguintes programas (face a essa especificação).

(a) `r = 0; i = 0;  
while (i<x) {  
 i = i+1; r = r+x;  
}`

(b) `r = 0; i = 0; p = 1;  
while (i<x) {  
 i = i+1; r = r+p; p = p+2;  
}`

4. O programa ao lado calcula a posição num array de um elemento que sabemos existir no array.

```
// exists (i in [0..n-1]) v[i] == x
k = 0;
while (v[k] != x)
    k=k+1
// v[k] == x
```

5. Considere a seguinte especificação de um programa que calcula o factorial de um número inteiro não negativo.

$$\begin{cases} \text{pré-condição: } & x = x_0 \geq 0 \\ \text{pós-condição: } & f = x_0! = \prod_{i=1}^{x_0} i \end{cases}$$

Encontre invariantes de ciclo que lhe permitam provar a correcção parcial dos seguintes programas (face a essa especificação).

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

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

6. Considere uma implementação de polinómios em que os coeficientes são guardados num vector, do coeficiente de maior grau até ao coeficiente de grau 0 (incluindo os coeficientes nulos).

Por exemplo o polinómio  $4x^5 - 3x^2 + 1$  será representado por um array cujos primeiros (6) valores serão  $[4, 0, 0, -3, 0, 1]$ .

Considere a seguinte especificação de um programa que calcula o valor de um polinómio num ponto.

$$\begin{cases} \text{pré-condição: } & N \geq 0 \\ \text{pós-condição: } & r == \sum_{k=0}^N a[k] * x^{N-k} \end{cases}$$

Encontre invariantes de ciclo que lhe permitam provar a correcção parcial dos seguintes programas (face a essa especificação). Apresente ainda as condições de estabelecimento, preservação e utilidade de cada um desses invariantes.

(a) `int i, r;`

```
i = 0; r = 0;
while (i <= N) {
    r = r + a[i] * pow (x,N-i);
    i = i+1;
}
```

(b) `int i, r;`

```
i = 0; r = 0;
while (i<=N) {
    r = r*x + a[i];
    i = i+1;
}
```

(c) `int i, r, p;`

```
r = 0; p = 1; i = N;
while (i>=0) {
    r = r + a[i]*p;
    p = p*x;
    i = i-1;
}
```

## 4 Correcção total

1. (a) Prove a correcção parcial do programa ao lado usando para isso o invariante fornecido.  
 $\text{r} = 0; \text{i} = 0;$   
 $\text{while } (\text{i} \neq \text{x}) \{$   
 $\quad // \text{r} == \text{i} * \text{x}$   
 $\quad // \&& \text{x} == \text{x0}$   
 $\quad // \&& \text{y} == \text{y0}$   
 $\quad \text{i} = \text{i}+1; \text{r} = \text{r}+\text{x};$   
 $\}$   
 $// \text{r} == \text{x0} * \text{y0}$
2. Para cada um dos programas apresentados na secção anterior, determine um variante de ciclo (e eventualmente reforce o invariante apresentado) de forma a provar a correcção total face às especificações apresentadas.
3. Considere o seguinte programa para determinar se um vector tem elementos repetidos. Determine um variante adequado para provar a terminação desse programa.

```
r = False; i = 0; j = 1;
while ((i<N-1) && !r) {
    if (a[i] == a[j]) r = True;
    j = j+1;
    if (j == N) { i = i+1; j = i+1; }
}
```