C1. Introdução à Análise de Correcção dos Algoritmos

Copiar link

Correcção de um algoritmo

> *Um algoritmo diz-se **correcto** se para todos os valores dos* inputs *(variáveis de entrada) ele pára com os valores esperados (i.e. correctos...) dos* outputs *(variáveis de saída). Neste caso diz-se que ele **resolve** o problema computacional em questão.*

Nem sempre a incorrecção é um motivo para a inutilidade de um algoritmo:

- Em certas aplicações basta que um algoritmo funcione correctamente para *alguns dos seus inputs*.
- Em problemas muito difíceis, poderá ser suficiente obter *soluções aproximadas* para o problema.

A análise da correcção de um algoritmo pretende determinar se ele é correcto, e em que condições.

A demonstração da correcção de um algoritmo cuja estrutura não apresente fluxo de controlo pode ser efectuada por simples inspecção. Exemplo:

```
int soma(int a, int b) {
  int sum = a+b;
  return sum;
}
```

Em alguns casos a correcção advém da própria especificação. Considere-se por exemplo uma implementação recursiva da noção de *factorial* de um número. A implementação segue de perto a definição, pelo que a sua correcção é imediata — uma vez que a própria definição é algorítmica, trata-se apenas de verificar se a sua codificação na linguagem de programação escolhida é correcta.

```
int factorial(int n) {
  int f;
  if (n<1) f = 1;
  else f = n*factorial(n-1);
  return f;
}
```

No entanto, no caso geral esta análise poderá apresentar uma dificuldade muito elevada e deve por isso ser efectuada com algum grau de formalismo, possivelmente recorrendo a uma *lógica de programas*.

Especificação de programas

Pré-condições e pós-condições

A análise de correcção dos algoritmos baseia-se na utilização de *asserções*: proposições lógicas sobre o estado actual do programa (o conjunto das suas variáveis). Por exemplo,

- $x > 0$
- $a[i] < a[j]$
- $\forall i.\, 0 \leq i < n \Rightarrow a[i] < 1000$

Pré-condição:

> *É uma propriedade que se* assume como verdadeira *no estado inicial de execução do programa, i.e., só interessa considerar as execuções do programa que satisfaçam esta condição.*

Pós-condição:

> *É uma propriedade que se* deseja provar verdadeira *no estado final de execução do programa.*

Triplos de Hoare

Um triplo de Hoare escreve-se como $\{P\}\ C\ \{Q\}$, em que

- C é o programa cuja correcção se considera
- P é uma *pré-condição* e Q é uma *pós-condição*

O triplo $\{P\}\ C\ \{Q\}$ é *válido* quando todas as execuções de C partindo de estados iniciais que satisfaçam P, *caso terminem*, resultem num estado final do programa que satisfaz Q.

Exemplo
O triplo $\{x = 10 \wedge y = 20\}\ \mathbf{swap}\ \{y = 10 \wedge x = 20\}$ associa a um programa $\mathbf{swap}$ uma especificação que exprime o seguinte:

> *se as variáveis x e y tiverem inicialmente os valores 10 e 20 respectivamente, então no final da execução estes valores terão sido trocados.*

Claramente, não é a especificação que esperamos de um típico programa $\mathbf{swap}$, que deverá trocar os valores das duas variáveis *quaisquer que sejam esses valores iniciais.*

Para escrever uma especificação adequada necessitaremos de recorrer a *variáveis auxiliares*. Veja-se a seguinte versão:

$$\{x = x_0 \wedge y = y_0\}\ \mathbf{swap}\ \{y = x_0 \wedge x = y_0\}$$

As variáveis x_0, y_0, ditas auxiliares, não podem ser utilizadas no programa $\mathbf{swap}$ — apenas podem ocorrer na especificação. É assim garantido que os seus valores não serão alterados durante a execução do programa, e por isso o valor de x_0 (resp. y_0) no final da execução é igual ao valor inicial da variável x (resp. y). Sendo assim, a especificação acima realmente capta a troca dos valores, tal como desejado.

Note-se que, depois de escrito o programa $\mathbf{swap}$, ele terá que ser *mostrado correcto* (ou *verificado*) em ordem a esta especificação, o que estudaremos nos próximos módulos.

Exercício

Traduza por palavras suas o significado de cada uma das seguintes especificações para um programa C. Excepto quando indicado, considere que todas as variáveis são de tipo inteiro.

1. $\{x \geq 0 \wedge y > 0\}\ C\ \{0 \leq r < y \wedge q * y + r = x\}$
   Esta especificação admite como solução programas triviais, que alteram os valores dos inputs. Por forma a ser possível referir na pós-condição os *valores iniciais* das variáveis, é necessário utilizar *variáveis auxiliares* como no exemplo seguinte.

2. $\{x = x_0 \geq 0 \wedge y = y_0 > 0\}\ C\ \{0 \leq r < y_0 \wedge q * y_0 + r = x_0\}$
   As variáveis x_0, y_0 são auxiliares, não podendo ser utilizadas no programa C.

3. $\{x = x_0 \geq 0 \wedge y = y_0 > 0\}\ C\ \{0 \leq r < y_0 \wedge (\exists_{q \geq 0}.\ q * y_0 + r = x_0)\}$

4. $\{x = x_0 \wedge y = y_0\}\ C\ \{(x = x_0 \vee x = y_0) \wedge x \geq x_0 \wedge x \geq y_0\}$

5. $\{x = x_0 \geq 0 \wedge e = e_0 > 0\}\ C\ \{|r * r - x_0| < e_0\}$
   (x, e, r variáveis de vírgula flutuante)

6. $\{\forall_{0 \leq i < N}.\ A[i] = a_i\}\ C\ \{0 \leq p < N \wedge (\forall_{0 \leq i < N}.\ A[i] = a_i \wedge a_p \leq a_i)\}$
   (A um *array* de tipo inteiro)

Exercício

De forma inversa, escreva agora especificações formais (pré-condições e pós-condições) correspondentes às seguintes especificações informais:

1. Um programa que coloca na variável r a soma dos valores (iniciais) das variáveis x e y.

2. Um programa que, para valores não negativos da variável y, coloca na variável r o produto dos valores (iniciais) das variáveis x e y.

3. 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.

4. Um programa que coloca na variável r o mínimo múltiplo comum das variáveis x e y.

5. Um programa que recebe como input dois arrays A e B com N elementos, e calcula o comprimento do *prefixo mais longo* que os dois têm em comum.

6. Um programa que procura uma ocorrência de k entre os índices a e b de um array A , colocando o índice dessa ocorrência na variável r. Caso $k \notin A[a...b]$, r tomará o valor -1.

7. Um programa que soma todos os elementos de um *array* com N posições, guardando o resultado na variável $result$.

Exercício

Pronuncie-se sobre a validade dos seguintes triplos de Hoare. Corrija os triplos inválidos, modificando a *pós-condição* por forma a ser tão informativa quanto possível.

1. $\{j = a\} \ j := j + 1 \ \{a = j + 1\}$

2. $\{i = j\} \ i := j + i \ \{i > j\}$

3. $\{i = i_0\} \ j := i + 1; \ i := j + 1 \ \{i = i_0 + 1\}$

4. $\{i \neq j\} \ \textbf{if } (i > j) \textbf{ then } \ m := i - j \textbf{ else } \ m := j - i \ \{m > 0\}$

5. $\{x = b\} \ \textbf{while } (x > a) \ x := x - 1 \ \{x = a\}$

Prova de Correcção de Programas Sem Ciclos

No caso dos programas sem construções iterativas ou recursivas, é fácil determinar uma *condição de verificação:* uma fórmula lógica (de primeira ordem) tal que, se essa fórmula for válida, então o triplo será também ele de certeza válido.

Vejamos como exemplo o seguinte triplo:

$$\{i = i_0\} \;\; j := i + 1; \; i := j + 1 \;\; \{i = i_0 + 1\}$$

Calculamos a condição de verificação *propagando para trás a pós-condição.* Para isso questionamos: que condição terá de ser verdade no estado intermédio, entre as duas instruções, para que o triplo seja válido?

Para que seja $i = i_0 + 1$ depois da execução de $i := j + 1$, terá de ser verdade, antes da execução desta instrução, a condição $i = i_0 + 1[j + 1/i]$, ou seja, a pós-condição em que substituímos a variável atribuída pela expressão que lhe foi atribuída, resultando em $j + 1 = i_0 + 1$. Tecnicamente estamos a calcular uma noção conhecida pela *pré-condição mais fraca* da instrução $i := j + 1$ em ordem à condição $i = i_0 + 1$.

Podemos representar isto da seguinte forma:

$$\begin{array}{l} \{\mathbf{i = i_0}\} \\ \quad j := i + 1 \\ \{j + 1 = i_0 + 1\} \\ \quad i := j + 1 \\ \{\mathbf{i = i_0 + 1}\} \end{array}$$

E questionamos novamente: que condição terá de ser verdade no estado inicial do programa, entre as duas instruções, para que o triplo seja válido? Ou por outras palavras, qual será a pré-condição mais fraca de $j := i + 1$ em ordem à condição intermédia $\{j + 1 = i_0 + 1\}$?
Basta agora substituir, em $j + 1 = i_0 + 1$, a variável j pela expressão $i + 1$, obtendo-se $j + 1 = i_0 + 1[i + 1/j] \;\equiv\; i + 1 + 1 = i_0 + 1$. Representamos:

$$
\begin{array}{l}
\{\mathbf{i} = \mathbf{i_0}\} \\
\{i + 1 + 1 = i_0 + 1\} \\
\quad j := i + 1 \\
\{j + 1 = i_0 + 1\} \\
\quad i := j + 1 \\
\{\mathbf{i} = \mathbf{i_0} + \mathbf{1}\}
\end{array}
$$

Note-se que não é possível propagar mais a condição, uma vez que o estado inicial foi já alcançado. A *condição de verificação* será então a seguinte implicação:

$i = i_0 \ \rightarrow \ i + 1 + 1 = i_0 + 1$, que é claramente inválida.

A justificação para o cálculo da condição de verificação é simples: se a pré-condição pode ser assumida no estado inicial, e foi calculada uma condição suficiente e necessária (por ser a a mais fraca) que deve ser verdade no início do programa para que o triplo seja válido, então a pré-condição terá de ser mais forte do que a condição propagada.

Observe-se que se a pós-condição fosse $i = i_0 + 2$ a condição de verificação seria
$i = i_0 \ \rightarrow \ i + 1 + 1 = i_0 + 2$, uma condição válida (e claro, também o triplo seria válido).

Execução Condicional

O cálculo da pré-condição mais fraca de um condicional exige um pouco mais de cuidado. Dado o comando $\ \textbf{if}\ (b)\ \ C_1\ \ \textbf{else}\ \ C_2$ e uma pós-condição ψ, que que condição terá de ser verdade no estado inicial para que a pós-condição seja verdade no estado final?

Começamos por calcular as pré-condições mais fracas ϕ_1 e ϕ_2 de cada um dos ramos do condicional em ordem a ψ. A pré-condição do condicional será então $(b \rightarrow \phi_1) \wedge (\neg b \rightarrow \phi_2)$.

Exercício

Prove a validade de cada um dos seguintes triplos de Hoare, começando por gerar as respectivas condições de verificação.

1. $\{i > j\} \ j := i + 1; \ i := j + 1 \ \{i > j\}$
2. $\{s = 2^i\} \ i := i + 1; \ s := s * 2 \ \{s = 2^i\}$
3. $\{\mathbf{True}\} \ \mathbf{if} \ (i < j) \ m := i \ \mathbf{else} \ m := j \ \{m \leq i \wedge m \leq j\}$
4. $\{i \neq j\} \ \mathbf{if} \ (i > j) \ m := i - j \ \mathbf{else} \ m := j - i \ \{m > 0\}$