Verificação de Algoritmos Recursivos

Para demonstrar a corretude parcial de algoritmos que possuem chamadas a funções, devemos nos certificar de que:

Em toda chamada de função, as pré-condições da função são satisfeitas.

Além disso, para demonstrar a corretude total de tais algoritmos, é necessário garantir que toda chamada de função efetivamente termina de ser executada. Nesse sentido:

1. Se um algoritmo A chama um algoritmo B que não pode direta ou indiretamente chamar o algoritmo A, então a terminação da chamada a B no código de A pode ser demonstrada apenas garantindo-se que as pré-condições que garantem a terminação de B são atendidas na chamada a B feita em A.

2. Se um algoritmo A chama um algoritmo B que pode, direta ou indiretamente, chamar o algoritmo A, então a terminação de uma chamada a B no código de A pode ser provada da seguinte maneira:

    2.1. Seja G = (V,E) o grafo direcionado cujos vértices são todos os algoritmos do código em questão, e tal que um par (u,v) é uma aresta do grafo sse há no código do algoritmo "u" uma chamada ao algoritmo "v".

    2.2. A cada algoritmo C que, em G, pertença à componente fortemente conexa de A (*), atribuir uma MEDIDA DE TERMINAÇÃO, isto é, uma expressão em termos de elementos da entrada e da pré-condição de C. Embora cada tal algoritmo C tenha a sua própria medida de terminação, todas as medidas de terminação dos algoritmos pertencentes à CFC de A devem assumir valores num mesmo conjunto bem-fundado (o qual, da mesma forma que no caso de variantes de laço, é frequentemente o conjunto dos números naturais).

        *: A "componente fortemente conexa" (CFC) de A é o conjunto dos algoritmos que (a) podem ser direta ou indiretamente chamados a partir de A, E QUE (b) podem direta ou indiretamente chamar A. Assim, por exemplo, pela definição do caso 2 acima, o algoritmo B e o próprio algoritmo A pertencem à CFC de A. Veja também:

            https://en.wikipedia.org/wiki/Strongly_connected_component

    2.3. Finalmente, para provar que a chamada ao algoritmo B no código de A termina, devemos:

        2.3.1. Mostrar que essa chamada "diminui a medida de terminação". Mais precisamente, seja VB o valor assumido pela medida de terminação de B no caso dos argumentos da chamada a B em questão. Seja também VA o valor assumido pela medida de terminação de A no caso dos argumentos da execução atual de A. Nós devemos mostrar então que VB < VA.

        2.3.2. Mostrar que o algoritmo B sempre termina de executar. Em particular, isso implica em realizar os passos 1 e 2 acima para cada chamada de função realizada em B.

Resumindo, portanto, para mostrar a terminação das chamadas de função (potencialmente) recursivas feitas num algoritmo A, é suficiente:

a) Atribuir medidas de terminação a todas as funções da Componente Fortemente Conexa de A (no grafo das chamadas de função);

b) Mostrar que todas as chamadas entre funções dessa CFC diminuem a medida de terminação.

Essa estratégia é ilustrada a seguir.

Tamanho de Lista Encadeada

Para provarmos a corretude de uma função recursiva que calcule o tamanho de uma lista encadeada, é útil recordar como listas de números naturais podem ser definidas indutivamente (*):

*: http://wikiufc.gelsol.org/courses/matdisc/attachments/1243/download

a) O conjunto L das listas de números naturais é indutivamente definido assim:

- [] ∈ L.
- n ∈ N e c ∈ L ==> n::c ∈ L.

onde "N" denota o conjunto dos números naturais e "n::c" uma lista cuja "cabeça" é "n" e cuja "cauda" é "c".

b) O tamanho de uma lista é calculado recursivamente assim:

$$|l| = \begin{cases} 0, & \text{se } l = []. \\ 1 + |c|, & \text{se } l = n::c, \text{ para certos "n" e "c".} \end{cases}$$

Além disso, para relacionar listas encadeadas e listas indutivamente definidas, definamos:

c) REGISTRO Nó:
  num: N;
  próx: ponteiro para Nó;
FIM.

d) Um ponteiro para Nó "p" representa uma lista l ∈ L sse:

1. p = nulo e l = [], ou

2. p != nulo, l = n::c (para certos "n" e "c"), p->num = n e p->próx representa "c".

Assim sendo, podemos escrever:

```
==========================================================================
Algoritmo:            tamanho
Entrada:              p: ponteiro para Nó
Pré-condição:         p representa uma certa lista l ∈ L
Medida de Terminação: |l|
Saída:                t ∈ N
Pós-condição:         t = |l|
--------------------------------------------------------------------------
1. SE p = nulo ENTÃO RETORNE 0.
2.             SENÃO RETORNE 1 + tamanho(p->próx).
==========================================================================
```

TEOREMA 1: o algoritmo "tamanho" é totalmente correto (no sentido técnico de
--------- "correção total").

==========
PROVA:

Primeiro, observe que o algoritmo não pode gerar erros de execução. Em particular, a dereferência executada pelo operador -> na linha 2 é, devido ao resultado do teste da linha 1, sempre feita sobre um ponteiro não nulo.

Observe agora que, se o teste da linha 1 resulta em verdadeiro, então o algoritmo termina retornando 0, que é a saída correta, pois, devido à pré-condição, "p" representa uma lista vazia.

Quando, porém, o teste da linha 1 resulta em falso, então a chamada "tamanho(p->próx)" é realizada. Nesse caso, observe que, pela pré-condição, "p" representa uma lista l = n::c e p->próx representa "c", de forma que a pré-condição da função chamada é satisfeita. Além disso, a medida de terminação da chamada é |c| = |l| - 1 < |l|, logo a chamada termina. Nesse caso, pela pós-condição da função chamada, a chamada tamanho(p->próx) retorna |c|, e portanto o valor retornado na linha 2 é 1 + |c| = |l|, CQD.
==========

EXERCÍCIOS
---

1. Escreva e prove a corretude de uma versão recursiva do algoritmo de Euclides para o cálculo do MDC de 2 números.

2. Escreva e prove a corretude de um algoritmo recursivo de busca binária cujos argumentos sejam um vetor v[1..n], índices "a" e "b", e um número "x" (a ser procurado em v[a..b]).