Projeto e Análise de Algoritmos (2023-1)

Flávio L. C. de Moura

April 4, 2023

1 Resumo da aula: COQ:MAX:PAA:INVARIANTE

Prezados alunos, na aula de hoje resolvemos o seguinte exercício:

Considere o algoritmo MaxVecElt a seguir que recebe um vetor com n elementos como argumento e retorna o maior elemento deste vetor:

$max \leftarrow A[0]$;
for $i = 1$ *to* $n - 1$ **do**
  if $max < A[i]$ **then**
    $max \leftarrow A[i]$;
  end
end
return max;

Algoritmo 1: MaxVecElt($A[0..n-1]$)

Mostre que este algoritmo é correto utilizando a seguinte invariante:

$\boxed{\text{Antes da i-ésima iteração, a variável } max \text{ contém o maior elemento do subvetor } A[0..i-1].}$

Solução: A prova é dividida em três partes:

1. **Inicialização**: Precisamos provar que a invariante vale antes da primeira iteração do laço. Neste caso, temos que $max = A[0]$ (linha 1) e $i = 1$ (linha 2), e a invariante afirma que a variável max contém o maior elemento do subvetor $A[0]$, o que é verdade.

2. **Manutenção**: Nesta etapa devemos considerar uma iteração qualquer do laço, digamos k ($1 \leq k \leq n-1$), e assumindo que a invariante vale antes da \$k\$-ésima iteração, precisamos provar que ela continua valendo antes da \$(k+1)\$-ésima iteração. Antes da \$k\$-ésima iteração temos que $i = k$, e assumimos que max contém o maior elemento do subvetor $A[0..k-1]$. Durante a \$k\$-ésima iteração, max é comparado com $A[k]$ (linha 3), e se $max < A[k]$ então max assume o valor de $A[k]$ (linha 4). Caso contrário max não muda de valor. Em ambos os casos, temos que max contém o maior elemento do subvetor $A[0..k]$, e portanto a invariante continua verdadeira antes da \$(k+1)\$-ésima iteração.

3. **Terminação**: Por fim, a terminação nos permite concluir que o algoritmo MaxVecElt é correto. De fato, ao final da última iteração, temos que $i = n$, e neste caso, a invariante nos diz que max contém o menor elemento do subvetor $A[0..n-1]$ que corresponde a dizer que MaxVecElt é correto. □

Em seguida consideramos uma versão deste algoritmo utilizando listas ao invés de vetores, e por fim vimos uma versão recursiva deste algoritmo ainda sobre listas:

if $l = h :: tl$ **then**
    if $max < h$ **then**
        MaxListEltRec(tl, h)
    else
        MaxListEltRec(tl, max)
    end
else
    return max
end

Algoritmo 2: MaxListEltRec(l, max)

e por fim, implementamos MaxListEltRec no Coq como sendo a função `elt_max` a seguir:

```
Fixpoint elt_max (l: list nat) (max:nat) :=
 match l with
 | nil => max
 | h::tl => if max <? h then (elt_max tl h) else (elt_max tl max)
 end.
```

Como exercício, prove que MaxListEltRec retorna o maior elemento da lista recebida como argumento, digamos l, desde que max seja inicializado com qualquer valor que não exceda o maior elemento de l. Em particular, podemos inicializar max com qualquer elemento de l:

Teorema 1 *Sejam l uma lista de naturais, e max um natural. Se $max \in l$ então MaxListEltRec(l, max) retorna o maior elemento da lista l.*

Como exercício, prove o teorema anterior.