UFMG

Linguagens de Programação - Lista de Exercícios 2

Arthur Pontes Nader - 2019022294
Belo Horizonte - MG - Brasil

Questões: Semântica Formal

1) O programa a seguir em **WHILE** atribui a Z o valor de n^m:

$$x := n;\ y := m;\ Z := 1;\ while\ \neg\ (y = 0)\ do\ (Z := Z * x;\ y := y - 1)$$

2) Os seguintes passos mostram a interpretação da expressão booleana na linguagem **WHILE**:

$$\mathcal{B}[\neg(x = 1)]\ s\ \text{-}\!\!>\ deve - se\ avaliar\ \mathcal{B}[(x = 1)]\ s\ primeiro$$

$$\begin{aligned}\mathcal{B}[(x = 1)]\ s\ &= (\mathrm{A}[x]\ s = A[1]s\ ) \\ &= (s\ x = N[1]) \\ &= (3 = 1) \\ &= ff\end{aligned}$$

Então:

$$\begin{aligned}\mathcal{B}[\neg(x = 1)]\ s\ &= \mathcal{B}\ [\neg(ff)] \\ &= tt\end{aligned}$$

3) As seguintes expressões mostram as substituições para expressões Booleanas da linguagem **WHILE:**

$$tt\ [y \to a_0] = tt$$
$$ff\ [y \to a_0] = ff$$
$$(\neg\ a_1)[y \to a_0] = \neg(a_1[y \to a_0])$$
$$(a_1 = a_2)[y \to a_0] = (a_1[y \to a_0] = a_2[y \to a_0])$$
$$(a_1 \leq a_2)[y \to a_0] = (a_1[y \to a_0] \leq a_2[y \to a_0])$$
$$(a_1 \wedge a_2)[y \to a_0] = (a_1[y \to a_0]) \wedge (a_2[y \to a_0])$$

4) A seguinte árvore de derivação representa o programa:

$$\dfrac{\dfrac{L1}{E1}\qquad\dfrac{\dfrac{L2}{E3}\qquad\dfrac{\dfrac{L3}{E5}\qquad\dfrac{}{E6}\ w\ ff}{E4}\ w\ tt}{E2}\ w\ tt}{\langle z:=0;\ while\ y\leq x\ do\ (z:=z+1;\ x:=x-y)\ s_{5,17,0}\rangle\rightarrow s_{5,2,3}}\ w\ tt$$

$L1,\ L2\ e\ L3\ são\ as\ folhas,\ o\ que\ equivale\ aos\ terminais$

$E1\ =\ \langle\ (z:=z+1;\ x:=x-y)\ s_{5,17,0}\rangle\rightarrow s_{5,12,1}$

$E2\ =\ \langle z:=0;\ while\ y\leq x\ do\ (z:=z+1;\ x:=x-y)\ s_5\ s_{12}\ s_1\rangle\rightarrow s_5\ s_2\ s_3$

$E3\ =\ \langle(z:=z+1;\ x:=x-y)\ s_{5,12,1}\rangle\rightarrow s_{5,7,2}$

$E4\ =\ \langle z:=0;\ while\ y\leq x\ do\ (z:=z+1;\ x:=x-y)\ s_{5,7,2}\rangle\rightarrow s_{5,2,3}$

$E5\ =\ \langle(z:=z+1;\ x:=x-y)\ s_{5,7,2}\rangle\rightarrow s_{5,2,3}$

$E6\ =\ \langle z:=0;\ while\ y\leq x\ do\ (z:=z+1;\ x:=x-y)\ s_{5,2,3}\rangle\rightarrow s_{5,2,3}$

5)

a - Determinar se o programa termina ou não depende do valor de x no estado em que o while é executado. Se x for menor do que 1 no estado em questão, atribuições sucessivas de x = x - 1 nunca fará com que x tenha valor igual a 1 para que o laço possa ser encerrado. Assim, o programa entra em laço infinito, sendo que sempre será executará a regra $[while^{tt}]$. Caso x tenha valor maior ou igual a 1, o laço termina em algum instante, ou seja, quando a regra $[while^{ff}]$ for executada.

b - O programa sempre termina, pois não importa o quão grande seja o valor de x, quando se repete o comando x = x - 1 sucessivamente, haverá um momento em que x será menor do que 1, fazendo assim com que o laço termine. Dessa forma, conclui-se que a regra $[while^{tt}]$ sempre será aplicada em algum instante pelo programa

c - Como a condição do while é sempre verdadeira e o comando skip não é capaz de alterar essa condição, a regra $[while^{tt}]$ sempre será aplicada, fazendo com que o programa entre em um laço infinito

6)
a - O tratamento do comando while pode ser feito da seguinte forma:

```
| ( While (b , stm1 ) )  =>
  let val bool  =  (evalB b s) in
    if (bool) then
      let val s2  =  (eval Stm stm1 s) in
        evalStm stm newS
      end
    else s
  end
```

b - Pode-se estender a linguagem com o comando “repeat S until b” da seguinte forma:

$$\frac{\langle S,\ s\rangle \rightarrow s_1\ ,\ \langle repeat\ S\ until\ b\ \rangle\ s_1 \rightarrow\ s_2 \quad if\ \mathcal{B}[b]\ s_1\ =\ ff}{\langle repeat\ S\ until\ b\ \rangle\ s_0 \rightarrow\ s_2}$$

$$\frac{\langle S,\ s\rangle \rightarrow s_1 \quad if\ \mathcal{B}[b]\ s_1\ =\ tt}{\langle repeat\ S\ until\ b\ \rangle\ s_0 \rightarrow\ s_2}$$

c - Em “datatype Stm = ” deve-se adicionar a seguinte linha:

```
| Repeat of Bexpr  *  Stm
```

E em “evalStm” adiciona-se o seguinte trecho:

```
| (Repeat (b, stm1))  =>
  let val s2  =  (evalStm stm1 s) val b2  =  (evalB b s2) in
    if b2 then
      s2
    else
      evalStm stm s2
  end
```

d - Para demonstrar que as duas expressões são equivalentes, deve-se mostrar que uma para de executar o laço se, e somente se, a outra também para. As duas expressões executam “S” e podem ou não alterar o valor de “b” ao final do processo. Assim, quando “b” for falso, a primeira expressão pára e a segunda entra

na parte do “if”. Dentro desse escopo do “if”, o “skip” é chamado, o que faz com que a execução também pare. Dessa forma, as duas expressões são semanticamente equivalentes.

Questões: Binding e Escopo

1)
a - Se a linguagem possuir escopo estático, então ao executar “p” a saída do programa será 1, pois a chamada da função “q” dentro da função “r” não alteraria o valor de x dentro de “r”.

b - Já se a linguagem tiver escopo dinâmico, a saída será 2, pois nesse caso a função “q” também estaria incrementando o valor de x no escopo da função “r”.

2)
a -

- Escopo de g - bloco 1
- Escopo do primeiro let - bloco 2
- Escopo da função f - bloco 3
- Escopo da função h - bloco 4
- Escopo do let dentro da função h - bloco 5

b - Os nomes definidos neste programa são:

- “g” - função
- “x” - parâmetro de g
- “inc” - variável inteira dentro do escopo do let na função g
- “f” - função
- “y” - parâmetro de f
- “h” - função
- “z” - parâmetro de h
- “inc” - variável inteira dentro do escopo do let na função h

c - Escopo de cada um dos nomes definidos:

- “g” - bloco 1
- “x” - bloco 1
- “inc” - bloco 2 (primeira ocorrência)
- “f” - bloco 2
- “y” - bloco 3
- “h” - bloco 2
- “z” - bloco 4
- “inc” - bloco 5 (segunda ocorrência)

d -
Escopo estático:

g(5) - {inc = 1, f (y) = y + 1, h (z) = z + 1}
h(5) = 5 + 1 = 6

O valor de g 5 é 6.

Escopo dinâmico:

g(5) - {inc = 1, f (y) = y + 1, inc = 2, f (y) = y + 2, h (z) = z + 2}
h(5) = 5 + 2 = 7

O valor de g 5 é 7.

Essa diferença ocorre pois no escopo estático, “inc” será associado ao seu valor correspondente no estado em que “f” é declarada e não mudará após isso, sendo portanto igual a 1. No caso do escopo dinâmico, o valor de “inc” quando “h” chama “f” seria o valor associado a essa variável no estado de “h”. No caso, “inc” foi alterado de 1 para 2 dentro de “h”, o que ocasionou na diferença de resultados ao final da execução.

5)
a - O problema da função bad_max está no caso em que a função é chamada para listas muito grandes, pois nesse caso haveria muitas comparações em sua execução, o que faria com que o programa demorasse muito tempo para determinar o elemento máximo da lista. Assim, apesar de funcionar, a função bad_max não é eficiente para resolução do problema no caso em que a lista de entrada possui muitos valores.

b - A seguinte função corrige o problema exposto anteriormente:

```
fun good_max [] = 0
  | good_max (h:: []) = h
  | good_max (h:: t) =
    let
      val h_aux = good_max t
    in
      if h > h_aux then h else h_aux
    end;
```

7) A função ficará da seguinte forma após as substituições:

$$
\begin{aligned}
&fun\ expr\ ()\ = \\
&\quad let \\
&\quad\quad val\ x\ =\ 1 \\
&\quad in \\
&\quad\quad let\ val\ x\ =\ 2\ in\ x\ +\ 1\ end\ + \\
&\quad\quad let\ val\ y\ =\ x\ +\ 2\ in\ y\ +\ 1\ end \\
&\quad end;
\end{aligned}
$$

Como resultado de sua avaliação, obtém-se o valor 7.