Semântica das Linguagens de Programação

1º Teste (15 de Maio de 2020, 9:30) / Duração: 2:30

Responda às questões do teste em folhas manuscritas. Coloque o seu nome e número no topo de cada folha. No final digitalize ou fotografe as folhas e faça o upload das suas respostas no Blackboard, dentro do prazo de duração do teste.

Questão 1 Considere a seguinte expressão do λ-calculus puro:

$$(\lambda y.\lambda a.\,(\lambda x.\lambda y.\,y\,x\,z)(y\,a))\,((\lambda x.\lambda y.\,x)(\lambda x.\lambda y.x))$$

1. Indique as variáveis livres e ligadas da expressão.
2. Quantos β-redexes encontra na expressão? Sublinhe cada um deles e indique a expressão resultante de fazer a sua redução (isto é, um passo de redução a partir da expressão inicial).
3. Apresente um λ-termo que
   (a) está na formal normal;
   (b) não está na forma normal, mas é fortemente normalizáveis;
   (c) é normalizável, mas não fortemente normalizável;
   (d) não é normalizável.

Questão 2 Considere os seguintes termos do lambda calculus:

$$\begin{array}{lcl@{\qquad\qquad}lcl} F & \equiv & (\lambda a.\lambda b.\,b) & K & \equiv & (\lambda a.\lambda b.\,a) \\ I & \equiv & (\lambda x.\,x) & B & \equiv & (\lambda x.\lambda y.\lambda z.\,x\,(y\,z)) \end{array}$$

1. Apresente a sequência da *ordem aplicativa* de redução até à forma normal da expressão

   $$F\,B\,(I\,B)\,(K\,K\,B)\,(K\,I)$$

   Sublinhe o β-redex que é selecionado em cada passo de redução.
2. Considere a expressão $B\,(F\,I)$. Coloque anotações de tipo nas variáveis que estão a ser abstraídas de forma a que esta expressão seja tipificável, e indique qual o tipo da expressão. (**Nota:** não precisa de apresentar a prova formal do juizo de tipificação.)

Questão 3 Considere a seguinte expressão da linguagem de programação funcional estudada:

$$\begin{array}{ll} \mathsf{let} & \mathsf{emp} \equiv \lambda x.\lambda l.\, \mathsf{listcase}\ l\ \mathsf{of}\ (<x,0>, \lambda h.\lambda t.\, \mathsf{if}\ h>0\ \mathsf{then}\ <h,x>\ \mathsf{else}\ <x,h>)\,, \\ & \mathsf{par} \equiv 8\,, \\ & \mathsf{lis} \equiv 9::(5+2)::\mathsf{nil} \\ \mathsf{in} & \mathsf{emp}\,((\lambda y.\, y*y)\ \mathsf{par})\ \mathsf{lis} \end{array}$$

1. Calcule, passo a passo, o seu valor usando a semântica de avaliação *call-by-name*.
2. Construa uma árvore de prova do juízo

$$l : \mathsf{List\,Bool},\ f : \mathsf{Int} \to \mathsf{List\,Bool} \to \mathsf{Bool} \vdash \lambda x.\, \mathsf{liscase}\ l\ \mathsf{of}\ (\mathsf{True}, \lambda h.\lambda t.\, h \wedge (f\ x\ t)) : \mathsf{Int} \to \mathsf{Bool}$$

3. Considere a seguinte função escrita em Haskell:

```
fun [] = []
fun [x] = [x]
fun (x:y:xys) = x : fun ((x+y):xys)
```

   (a) Defina uma função fun, equivalente a esta, na linguagem funcional que estudou.

   (b) Apresente uma expressão da linguagem funcional que estudou (versão *call-by-name*) equivalente à expressão Haskell `fun [1,5..]`

Questão 4 Pretende-se estender a linguagem de programação funcional, com um novo tipo de dados para representar *rose trees* polimórficas, ou seja, o equivalente ao seguinte tipo de dados do Haskell:

```
data RTree a = Empty | Node a [RTree a]
```

1. Defina a sintaxe abstracta das novas expressões e do novo tipo, e as regras de inferência de tipo para as novas expressões.
2. Indique as novas formas canónicas da linguagem e as novas regras de avaliação *call-by-value*.
3. Defina uma função sumRT, de tipo RTree Int $\to$ Int, que soma todos os elementos de uma *rose tree* de inteiros. Deve definir também todas as funções auxiliares que utilizar.
4. As *rose trees* podem ser vistas como estruturas de dados, construidas à custa das alternativas e dos tuplos, vendo os seus construtores e eliminadores como açucar sintáctico.

   Apresente esta definição alternativa das *rose trees*.