Linguagens de Programação 2018/2019

Departamento de Informática, Universidade de Évora

1º Trabalho Prático

– Cálculo Lambda: termo α-equivalente –

1 Objectivo

Utilizando uma linguagem de programação à escolha (entre *C*, *Java* ou *Python*) em conjunto com o gerador de analisadores sintáticos correspondentes (*Flex/Bison*, *Jlex/CUP* ou *PLY*), pretende-se implementar um **interpretador** para cálculo lambda que transforme um termo-λ num termo α-equivalente, onde todas as variáveis têm nomes distintos.

2 Cálculo lambda

O **cálculo lambda** é um sistema matemático que ilustra alguns conceitos importantes de linguagens de programação de uma forma simples e pura. O cálculo lambda tradicional possui três partes principais: uma notação para definir funções, um sistema de prova para resolver equações entre expressões e um conjunto de regras de cálculo, a *redução*, que permite fazer uma avaliação simbólica das expressões.

2.1 Notação

Assumindo que o carácter '\' faz o papel do símbolo λ e que as variáveis têm um nome constituido por uma única letra minúscula, a sintaxe para termos lambda é definida pela gramática BNF

```
<termo> ::= <variavel>
          | (\<listav>.<termo>)
          | (<termo> <termo>)
```

onde

```
<listav> ::= <variavel>
           | <variavel><listav>
```

Nesta gramática, a segunda produção representa a λ-*abstração* e a terceira a *aplicação*. Para não sobrecarregar os termos com parêntesis, adoptam-se as seguintes convenções:

- a aplicação é a construção com maior prioridade;

- a aplicação associa à esquerda;
- o corpo de uma λ-abstração estende-se para a direita até onde for possível.

3 Implementação

Este trabalho consiste na implementação de um interpretador da linguagem descrita que transforme um termo-λ num termo α-equivalente (onde todas as variáveis têm nomes distintos), utilizando uma das linguagens à escolha para construir o analisador sintático:

- `C` recorrendo ao `Flex/Bison`
- `Java`[1] recorrendo ao `JLex/CUP`
- `Python` recorrendo ao `PLY`

O programa deverá ter duas fases:

1. análise sintática do termo-λ construindo a sua árvore de sintaxe abstrata;
2. construção de um termo α-equivalente a partir do termo-λ encontrado.

O interpretador lê um termo-λ da sua entrada padrão (*standard input*) e, caso não ocorra nenhum erro sintático, escreve no terminal (*standard output*) o termo original e o termo α-equivalente. Depois de escrever o resultado da redução, o programa termina. Se for detetado algum erro sintático, o programa termina com essa indicação.

Os termos-λ original e encontrado deverão ser escritos a partir da árvore de sintaxe abstrata em linhas distintas e prefixados por '`<<`' e '`>>`', respetivamente.

3.1 Organização

A função principal, que inicia a execução do interpretador, deverá encontrar-se num ficheiro com nome `lambda.c`, `Lambda.java` ou `lambda.py`.

Deverá ser entregue um ficheiro `makefile` que:

- compile o programa através das invocações `make` ou `make all`,
- execute o programa através do comando `make run`, e
- apague os ficheiros criados na compilação através de `make clean`

4 Exemplos

Os seguintes exemplos, apresentam a saída produzida pelo interpretador, a menos da inclusão de mais parêntesis.

[1] Utilizando o JDK 7 ou posterior.

1. termo: `x`
   `<< x`
   `>> x`

2. termo: `\x.x`
   `<< \x.x`
   `>> \x.x`

3. termo: `(\x.x) x`
   `<< (\x.x) x`
   `>> (\y.y) x`

4. termo: `(\x.\y.x y) y`
   `<< (\x.\y.x y) y`
   `>> (\x.\z.x z) y`

5. termo: `(\x.x) (\x.x) z`
   `<< (\x.x) (\x.x) z`
   `>> (\x.x) (\y.y) z`

6. termo: `(\fx.f x) (\x.x) z`
   `<< (\fx.f x) (\x.x) z`
   `>> (\fx.f x) (\y.y) z`

7. termo: `\fx.f x`
   `<< \fx.f x`
   `>> \fx.f x`

8. termo: `(\fx.f x) (\x.x)`
   `<< (\fx.f x) (\x.x)`
   `>> (\fx.f x) (\y.y)`

9. termo: `(\xy.x y) (\y.y x)`
   `<< (\xy.x y) (\y.y x)`
   `>> (\ab.a b) (\c.c x)`

10. termo: `(\a.(\b.(\c.(\d.d c) (b c)) (b a)) (\bcd.b c (c d))) (\ab.b)`
    `<< (\a.(\b.(\c.(\d.d c) (b c)) (b a)) (\bcd.b c (c d))) (\ab.b)`
    `>> (\a.(\b.(\c.(\d.d c) (b c)) (b a)) (\efg.e f (f g))) (\hi.i)`

5 Observações

5.1 Realização do Trabalho

O trabalho deverá ser realizado por grupos de **dois** alunos.

Faz parte da realização do trabalho a **elaboração de 5 novos exemplos** que ilustrem quer o correto funcionamento da implementação feita, quer eventuais problemas que não tenham sido completamente resolvidos.

5.2 Entrega do Trabalho

Os elementos a entregar serão:

- os ficheiros com o código fonte da implementação e o `makefile`;
- o(s) ficheiro(s) com os exemplos;
- o relatório em formato PDF.

O relatório deverá incluir:

- a identificação dos autores do trabalho;
- a descrição das árvores de sintaxe abstratas;
- uma descrição sucinta (e esclarecedora) do funcionamento do interpretador;
- os exemplos elaborados (ver secção anterior) e a descrição do que cada um deles mostra;
- referências à bibliografia consultada

Estes elementos deverão ser entregues através do `moodle` até até à data lá indicada, num ficheiro de nome `xxxxx_xxxxx_xxxxx.tar.gz` (ou `xxxxx_xxxxx.zip`), onde `xxxxx` é o número de um dos alunos que fazem parte do grupo.

5.3 Apresentação do trabalho

O trabalho realizado por cada grupo será apresentado na aula prática seguinte. Apesar do trabalho ser de grupo, cada aluno, a título individual, tem a responsabilidade de responder por todo o trabalho. Assim, é indispensável que cada membro do grupo programe efectivamente.

5.4 Fraudes

Cuidado com as fraudes!

Considera-se fraude todas as situações onde o código entregue não foi escrito (integralmente ou não) pelo grupo; cada grupo é responsável pelo seu trabalho e não o pode oferecer, directa ou indirectamente, a outro grupo ou obtê-lo por outra forma. Nestes casos será aplicado o **código de conduta** do Departamento de Informática.

Referem-se de seguida algumas situações de fraude comuns:

- se alguém dum grupo "oferecer"o trabalho resolvido (ou parte dele) a um elemento de outro grupo, trata-se duma fraude envolvendo dois grupos;
- se dois grupos se juntam para fazer o trabalho e depois o entregam em duplicado (não é necessário ser cópia integral), então também se considera fraude de ambos os grupos.