Universidade de Évora

Escola de Ciências e Tecnologia

Licenciatura em Engenharia Informática

Linguagens de Programação 2016/2017

Trabalho Prático

Interpretador de Cálculo Lambda

Docente:

Teresa Gonçalves

Discentes:

David Parreira, nº 33257;

Rute Veladas nº 33402.

Évora, 29 de Maio de 2017

Introdução

No âmbito da disciplina de Linguagens de Programação foi solicitada a elaboração de um interpretador de cálculo lambda.

Para o primeiro trabalho, é pedido um interpretador que transforme o termo lambda original no termo a-equivalente onde todas as variáveis têm nomes distintos.

Para o segundo trabalho é pedido um interpretador que faça a avaliação desse termo equivalente e o transforme num termo equivalente o mais simples possível utilizando a redução.

Optou-se por utilizar linguagem de programação C recorrendo aos analisadores sintáticos Flex e Bison.

Funcionamento do Interpretador

A primeira fase da realização do primeiro trabalho consiste na implementação de um analisador lexical e de um analisador sintático para a nossa linguagem. O analisador lexical foi implementado através do uso do Flex e o analisador sintático foi implementado através do uso do Bison.

Os terminais da linguagem são os seguintes:

- As letras, que são minúsculas:
  [a-z] {
  yylval.letter = strdup(yytext);
  return LETTER;
  }
- Lambda, que pela notação escolhida é o equivalente a “!”:
  "!" return LAMBDA;
- Ponto:
  "." return PONTO;
- Parêntesis esquerdo:
  “(“ return PARE;
- Parêntesis direito:
  “)“ return PARD;
- Espaços, novas linhas e tabs (serão ignorados):
  [\n\t]

Para o analisador sintático procedeu-se à elaboração das regras da gramática, e a gramática utilizada foi a seguinte:

```
program: terms ;
terms : term
        | term terms
        ;
term: abs
```

```
        | app
        | PARE terms PARD
        ;
abs: LAMBDA LETTER PONTO terms ;
app : LETTER
        | LETTER terms
        ;
```

A regra *terms* da gramática utilizada permite ao programa analisar se existem um ou vários termos.

A regra *term* define que um termo pode ser uma abstração, uma aplicação, ou qualquer uma destas dentro de parêntesis.

A regra *abs* define em que consiste uma abstração e a regra *app* define em que consiste uma aplicação, sendo que uma aplicação pode consistir numa única letra, ou no conjunto de vários termos.

Adotaram-se ainda as seguintes convenções fornecidas pela professora no enunciado:

- 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.

A segunda fase da realização do primeiro trabalho consiste na construção do termo a-equivalente a partir do termo lambda encontrado.

O interpretador lê um termo lambda através de standard input e caso não ocorra nenhum erro sintático, escreve no terminal o termo original e o termo a-equivalente e o programa acaba. Se for detetado algum erro sintático o programa termina com essa indicação.

Para o segundo trabalho foi implementada a estratégia de redução call-by-name, neste o interpretador escreve o termo original e o reduzido a partir da árvore de sintaxe abstrata em linhas distintas.

Exemplos Elaborados – Primeiro Trabalho

O nosso interpretador foi testado com todos os exemplos fornecidos pela docente no enunciado. A gramática elaborada aceita todos os casos.

1. ← (!x. (x y (!a.a)) (!x. x))
   → (!o (o y (!m.m)) (!s.s))
   Mostra o caso de uma única abstração cujo termo após o ponto é uma aplicação composta por dois termos, em que um deles é composto duas aplicações seguidas de uma abstração.
2. ← (!x. !y. x y) y (!x. x (!x. x) x)
   → (!o. !m. o m) y (!s. s (!b. b) s)
   Mostra o caso de uma variável independente (y) e uma abstração x com outra abstração x dentro.
3. ← !x. (!a. a y) (!x. x x)
   → !o. (!m. m y) (!s. s s)
   Mostra outro caso de uma variável independente (y) dentro de duas abstrações x e a respetivamente.
4. ← !x. x y
   → !o. o y
   Aqui demonstramos o caso mais básico de uma abstração simples com a variável independente y.

Exemplos Elaborados – Segundo Trabalho

Para o segundo trabalho não elaboramos todos os exemplos pois o trabalho não está concluído e apenas resolve alguns casos.

1. ← !w. (!x. x y) w
   → !w. w y
   Mostra o caso de uma abstração cujo corpo contém uma abstração e uma aplicação.
2. ← (!a. a a)(!b. !c . b c)
   → (!b.!c.bc)(!a.!d.ad)
   →(!c.(!a.!d.ad)c)
   → (!c.!d.cd)
   Mostra o caso de duas abstrações seguidas.
3. ← !y. y w z
   → !y. y w z
   Mostra o caso simples em que o programa deteta que não existem reduções possíveis e imprime a árvore original.
4. ← !y. (!a. a y)(!b. b b)
   → !y. (!b. b b) y
   → !y. y y
   Mostra o caso de uma abstração com duas abstrações no seu corpo.

Conclusão

Na realização do primeiro trabalho deparámo-nos com algumas dificuldades, mas conseguimos concluir o trabalho sendo que este funciona completamente.

Ao realizar o segundo trabalho não conseguimos concluir o mesmo pois deparámo-nos com várias dificuldades na manipulação da árvore de sintaxe abstrata, sendo que este apenas resolve alguns casos.

Não conseguimos juntar ambos os trabalhos sendo que cada um corre separadamente e, assim sendo, o segundo trabalho não reduz logo do inicio ao fim, efetua apenas uma redução de cada vez.

Bibliografia

- Slides/Exercícios fornecidos pela docente.
- http://www.utdallas.edu/~gupta/courses/apl/lambda.pdf