Processamento de Linguagens
Relatório do Trabalho Prático nr. 2
logoLISS

Ana Sampaio (54740) Hugo Frade (54750)
Miguel Costa (54746)
LEI 3º ano

Junho de 2011

Resumo

Este relatório é referente ao trabalho práctico nº 2 da Unidade Curricular de Processamento de Linguagens. Neste documento é exposto com detalhe todo o processo de resolução do mesmo.

O segundo trabalho prático consiste no desenvolvimento de um compilador capaz de gerar código para uma máquina de stack virtual. Este compilador processa uma linguagem específica, e bastante curiosa, a LogoLISS , cuja gramática foi fornecida. Para tal, foi utilizada a ferramenta Yacc com auxilio do Flex.

Conteúdo

1 Introdução

No âmbito da Unidade Curricular (UC) de Processameto de Linguagens foi proposto pelos docentes a realização de um trabalho prático que aborda conteúdos leccionado nesta UC. Em particular, este trabalho envolve o temática da Análise Semântica e Transformação especificada através de Gramáticas Tradutoras, tradução essa dirigida pela Sintaxe, utilizando o gerador Yacc .

Neste relatório apresentamos uma proposta de resolução do problema, bem como uma explicação detalhada da mesma. Para isso, dividimos este documento em algumas secções, que consideramos importantes. A primeira secção, Descrição do trabalho práctico, destina-se à apresentação do problema. O capítulo seguinte é onde explicamos em que se baseia a resolução do grupo e revela as opções tomas. Seguidamente apresenta-se os resultados obtidos, em forma de crítica. Por fim, expomos as nossas conclusões. Encontra-se em anexo a gramática fornecida e o código resultante da resolução do trabalho práctico.

2 Descrição do trabalho prático

O âmbito fundamental deste trabalho destina-se a desenvolver um compilador para a linguagem LogoLISS, com base na GIC fornecida pelos docentes, e recorrendo ao gerador Yacc com auxilio do Flex. O compilador de LogoLISS deve gerar pseudo-código, Assembly da Máquina Virtual VM .

2.1 LogoLISS

A linguagem LogoLISS é uma simbiose da famosa linguagem Logo e a LISS. A primeira, concebida pelo famoso professor americano Seymour Papert, é a linguagem funcional da Tartaruga em movimento, que foi utilizada para o ensino da programação. Já a segunda, provem de uma versão simplificada da linguagem LISS (Language of Integers, Sequences and Sets), concebida há vários anos por Leonor Barroca e Pedro Henriques, para ensino da compilação.

Tendo por base a LISS, LogoLISS permite manusear escalares (valores atómicos) bem como vectores (arrays) do tipo inteiro, na forma de constantes e de variáveis. Como é usual neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver redeclarações, nem utilizações sem declaração prévia; se nada for explicitado, o valor da variavel após a declaração é indefinido. Sobre inteiros, estão disponíveis as habituais operações aritméticas e lógicas. Além destas, a linguagem LISS permite ainda ler do standard input e escrever no standard output. As instruções vulgares para controlo do fluxo de execução (condicional e cíclica) estão também incluidas na linguagem.

Da linguagem Logo, a LogoLISS herda os movimentos da Tartaruga num plano de coordenadas cartesianas.

A gramática independente de contexto da linguagem LogoLISS é apresentada em Apêndice.

O objectivo específico inserido no contexto deste problema é implementar a funcionalidade base relativa ao comando da *Tartaruga* e sobre os *escalares* do tipo **inteiro**. A implementação de *arrays*, que envolve a operação de *indexação*, foi considerada de carácter opcional.

3 Resolução do problema e opções tomadas

3.1 Estruturas de dados

Durante a realização do compilador recorremos à utilização de estruturas de dados, tanto para analisar a informação lida, como para armazena essa mesma informação.

listVar

Esta estrutura tem como função guardar todas as declarações de variaveis de uma linha antes de ser adicionada na tabelas de identificadores. É constituida por um nome, valor e tem um apontador para a próxima variavel, se existir:

```
typedef struct listV {
    char* nome;
    char* valor;
    struct listV* seg;
} listVar;
```

Variáveis

A estrutura é utilizada para guardar o nome e o valor da variável no momento da sua declaração ou atribuir algum valor.

```
typedef struct vars{
char* nome;
char* valor;
} Variaveis;
```

Constantes

A estrutura é utilizada para guardar o valor que uma dada variável possui, e que poderá ser um inteiro ou um boolean. Caso seja inteiro, tal valor é guardado na variável valorI. Caso contrário, é guardado na variavel valS.

```
typedef struct consts {
   int valI;
   char* valS;
} Constantes;
```

Union

Foi definida uma union que ilustra o tipo de valores que uma variavel pode tomar.

```
union{
   int valn;
   char *vals;
   float valr;
   Variavel var;
   Constantes cons;
}
```

Hash table

Recorremos a uma tabela de Hash para representar a informação relativa aos identifcadores. Todos os identificadores constituindo as instruções formam as palavras reservadas.

```
typedef struct hasht {
   char* key;
   int type;
   int eGlobal;
   int add;
   struct hasht * next;
} TabelaHash, *ELEM;
```

Deste modo, os campos de que necessitamos para esta estrutura foram:

- key, representada pelo nome da variável; também usada para aceder ao índice da variável;
- type, que toma o valor de 1 se for um inteiro, 2 se for um boleano e 3 se for um array;
- eGlobal, que toma o valor 0 se a variável for global, e 1 caso contrário. Esta distinção foi feita devido às regras de manipulação de dados da Máquina Virtual;
- add, que corresponde ao endereço (i.e., posição de memória) onde é guardada a variável na stack usada pela Máquina Virtual.

3.2 Regras de Tradução para Assembly da VM

3.2.1 Operações de base

Utilizamos os seguintes operações de base para realizar operações aritméticas:

- ADD: coloca na pilha o resultado da soma dos dois inteiros associados;
- SUB: coloca na pilha o resultado da diferença dos dois inteiros associados;
- MUL: coloca na pilha o resultado da multiplicação dos dois inteiros associados;
- DIV: coloca na pilha o resultado da divisão dos dois inteiros associados.

3.2.2 Manipulação de dados

Utilizados operaçoes de manipulação de dados para empilhar/ retirar da pilha os valores, e também arquivar-los:

- PUSHI: coloca o valor associado na pilha;
- PUSHG: coloca na pilha o valor localizado na posição indicada da stack das variáveis globais;
- PUSHL: coloca na pilha o valor localizado na posição indicada da stack das variáveis locais;
- DUP: duplica e empilha os valores inteiros associados de topo na pilha;
- POP: retira da pilha o número de valores indicado.

- STOREL: retira um valor da pilha e arquiva-a na pilha das variáveis locais, na posição indicada;
- STOREG: retira um valor da pilha e arquiva-a na pilha das variáveis globais, na posição indicada.

3.2.3 Input-Output

Utilizamos algumas instruções que permitem realizar desenhos, utilizando a versão gráfica da máquina virtual:

- DRAWLINE: dadas as coordenadas (valores inteiros) no inicio e fim, permite desenhar um seguemento;
- DRAWPOINT: dadas as coordenadas, desenha um ponto;
- OPENDRAWINGAREA: abre uma janela de desenho com as dimensões indicadas;
- REFRESH: refresca a janela gráfica.

3.2.4 Inicialização e fim

Estas são as instruções de inicio e fim do programa:

- START
- STOP

4 Resultados obtidos

Infelizmente não conseguimos obter resultados concretos deste problema. Embora nem toda a gramática tenha sido tratada, julgamos que a parte que tratamos está idealizada de forma correcta. No entanto, o código desenvolvido não consegue gerar resultados a partir de um input. Daí que não tenha sido possível obter os resultados da Máquina Virtual.

5 Conclusão

Após a realização deste trabalho práctico constatamos que este trabalho nos motivou bastante em relação aos conteúdos leccionados na UC, uma vez que foi possível obter uma percepção real de aplicação do que apresentamos nas aulas. Além disso, este trabalho revelou-se muito útil na medida em que nos forneceu uma ideia bastante concisa de como podemos interegir o FLEX com o YACC.

No entanto, não foi possível tratar de toda a linguagem, deixando-a um pouco desfuncional em relação a algumas operações. Não foi também possível obter resultados da Máquina Virtual, uma vez que o código desenvolvido não está prepadado para gerar resultados a partir de um input. Como trabalho futuro, propomos transformar a resolução adoptada numa aplicação funcional, capaz de gerar código Assembly para a Máquina Virtual.

6 Bibliografia

- Manual de funcionamento da Máquina Virtual:
  `http://epl.di.uminho.pt/~gepl/LP/VirtualMachines.html`.

A Gramática Utilizada

LogoLISS - A Toy Language

```
/************************ Program

Liss                  --> "PROGRAM" identifier  "{" Body  "}"

Body                  --> "DECLARATIONS" Declarations "STATEMENTS"   Statements


/************************ Declarations

Declarations          --> Declaration
                        | Declarations Declaration

Declaration           --> Variable_Declaration


/************************ Declarations: Variables

Variable_Declaration --> Vars "->" Type ";"

Vars                  --> Var
                        | Vars "," Var

Var                   --> identifier Value_Var

Value_Var             -->
                           | "=" Inic_Var

Type                  --> "INTEGER"
                        | "BOOLEAN"
                        | "ARRAY" "SIZE" number

Inic_Var              --> Constant
                        | Array_Definition

Constant              --> number
                        | string
```

```
                        | "TRUE"
                        | "FALSE"


/************************ Declarations: Variables: Array_Definition

Array_Definition     --> "[" Array_Initialization "]"

Array_Initialization --> Elem
                        | Array_Initialization "," Elem

Elem                 --> number


/************************* Statements

Statements           --> Statement ";"
                        | Statements Statement ";"

Statement            --> Turtle_Commands
                        | Assignment
                        | Conditional_Statement
                        | Iterative_Statement


/************************* Turtle Statement

Turtle_Commands      --> Step
                        | Rotate
                        | Mode
                        | Dialogue
                        | Location

Step                 --> "FORWARD" Expression
                        | "BACKWARD" Expression

Rotate               --> "RRIGHT"
                        | "RLEFT"

Mode                 --> "PEN" "UP"
                        | "PEN" "DOWN"
```

```
Dialogue             --> Say_Statement
                       | Ask_Statement

Location             --> "GOTO" number "," number
                       | "WHERE" "?"


/************************ Assignment Statement

Assignment           --> Variable "=" Expression

Variable             --> identifier Array_Acess

Array_Acess          -->
                       | "[" Single_Expression "]"


/*********************** Expression

Expression           --> Single_Expression
                       | Expression Rel_Oper Single_Expression

/******** Single_Expression

Single_Expression    --> Term
                       | Single_Expression Add_Op Term

/******** Term

Term                 --> Factor
                       | Term Mul_Op Factor

/******** Factor

Factor               --> Constant
                       | Variable
                       | SuccOrPred
                       | "(" Expression ")"

/******** Operators

Add_Op               --> "+"
```

```
                        | "-"
                        | "||"

Mul_Op                --> "*"
                        | "/"
                        | "&&"
                        | "**"

Rel_Op                --> "=="
                        | "!="
                        | "<"
                        | ">"
                        | "<="
                        | ">="
                        | "in"

/******** SuccOrPredd

SuccOrPred            --> SuccPred identifier

SuccPred              --> "SUCC"
                        | "PRED"

/********************** IO Statements

Say_Statement       --> "SAY" "(" Expression ")"

Ask_Statement      --> "ASK" "(" string "," Variable ")"


/********************** Conditional & Iterative Statements

Conditional_Statement --> IfThenElse_Stat

Iterative_Statement   --> While_Stat

/******** IfThenElse_Stat

IfThenElse_Stat       --> "IF" Expression "THEN" "{" Statements "}" Else_Expression

Else_Expression       -->
                          | "ELSE" "{" Statements "}"
```

```
/******** While_Stat

While_Stat           --> "WHILE" "(" Expression ")" "{" Statements "}"
```