Processamento de Linguagens LEI (3ºano) + LCC (2ºano)

Trabalho Prático nº 2 (Yacc)

Ano lectivo 09/10

1 Objectivos e Organização

Este trabalho prático tem como principais objectivos:

- (genericamente) aumentar a experiência em engenharia de linguagens e em programação generativa (gramatical);
- (especificamente) desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, suportado numa gramática tradutora;
- (especificamente) desenvolver um **compilador** gerando código para uma **máquina de stack virtual**;
- (especificamente) utilizar geradores de compiladores baseados em gramáticas tradutoras, como o Yacc.

e como **objectivos** secundários:

- aumentar a experiência de uso do ambiente Linux, da linguagem imperativa
 C (para codificação das estruturas de dados e respectivos algoritmos de manipulação), e de algumas ferramentas de apoio à programação;
- rever e aumentar a capacidade de escrever gramáticas independentes de contexto que satisfaçam a condição LR().

Para o efeito, esta folha contém apenas 1 enunciado.

O programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, na 1ª semana de Junho, após o fim das aulas, em dia a combinar.

O relatório a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da linguagem/gramática e as regras de tradução para Assembly da VM (incluir as especificações Yacc), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LATEX.

2 Enunciado

A linguagem LogoLISS é uma simbiose da famosa linguagem de Seymour Papert, Logo—a linguagem funcional da Tartaruga em movimento, pensada por este conhecido pedagogo americano para ensino da programação, com uma versão simplificada da linguagem LISS—Language of Integers, Sequences and Sets, concebida há anos por Leonor Barroca e Pedro Henriques para ensino da compilação¹

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 da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver re-declarações, nem utilizações sem declaração prévia; se nada for explicitado, o valor da variável 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 previstas.

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

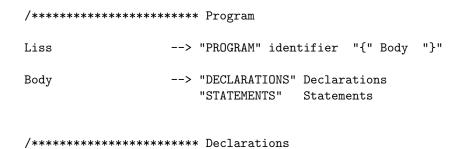
A gramática independente de contexto da dita linguagem LogoLISS é apresentada em Apêndice.

O que se pretende no contexto específico deste Trabalho Prático é implementar a funcionalidade base relativa ao comando da *Tartaruga* e sobre os *escalares* do tipo **inteiro**. A implementação de *arrays* com a operação habitual de *indexação* é opcional, sendo deixada ao critério de cada grupo.

Desenvolva, então, um compilador para LogoLISS, com base na GIC dada e recurso ao Gerador Yacc com auxilio do Flex.

O compilador de LogoLISS deve gerar pseudo-código, Assembly da Máquina Virtual VM cuja documentação completa se encontra em http://epl.di.uminho.pt/~gepl/LP/VirtualMachines.html.

A LogoLISS - A Toy Language



¹LISS é uma linguagem de programação imperativa, simplificada, que combina conceitos e funcionalidades de várias outras linguagens; embora minimalista, a sua compilação levanta alguns desafios curiosos.

Declarations --> Declaration

| Declarations Declaration

Declaration --> Variable_Declaration

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

Vars --> Var

| Vars "," Var

Var --> identifier Value_Var

Value_Var -->

| "=" Inic_Var

Type --> "INTEGER"

| "BOOLEAN"

| "ARRAY" "SIZE" number

| Array_Definition

Constant --> Sign number

| string | "TRUE" | "FALSE"

Sign

| "+" | "-"

Array_Definition --> "[" Array_Initialization "]"

Array_Initialization --> Elem

| Array_Initialization "," Elem

Elem --> Sign number

/****** Statements

Statements --> Statement

| Statements Statement

Statement --> Turtle_Commands

```
| Assignment
                      | Conditional_Statement
                      | Iterative_Statement
/***** Turtle Statement
{\tt Turtle\_Commands}
                    --> Step
                      | Rotate
                      | Mode
                      | Dialogue
                      | Location
                    --> "FORWARD" Expression
Step
                      | "BACKWARD" Expression
                    --> "RRIGHT"
Rotate
                      | "RLEFT"
                    --> "PEN" "UP"
Mode
                      | "PEN" "DOWN"
Dialogue
                    --> Say_Statement
                      | Ask_Statement
                    --> "GOTO" number "," number
Location
                      | "WHERE" "?"
/****************** Assignment Statement
                   --> Variable "=" Expression
Assignment
Variable
                   --> identifier Array_Acess
Array_Acess
                    | "[" Single_Expression "]"
/***** Expression
                   --> Single_Expression
Expression
                     | Expression Rel_Oper Single_Expression
/***** Single_Expression
Single_Expression
                   --> Term
                     | Single_Expression Add_Op Term
/***** Term
```

```
--> Factor
Term
                    | Term Mul_Op Factor
/***** Factor
Factor
                  --> Constant
                    | Variable
                    | SuccOrPred
                    | "!" Expression
                    | "+" Expression
                    | "-" Expression
                    | "(" Expression ")"
/****** Operators
Add_Op
                  --> "+"
                    | "-"
                    1 "11"
Mul_Op
                  --> "*"
                   | "/"
                    | "&&"
                    | "**"
                  --> "=="
Rel_Op
                    | "!="
                    | "<"
                    | ">"
                    | "<="
                    | ">="
                    | "in"
/***** SuccOrPredd
SuccOrPred
                  --> SuccPred identifier
                  --> "SUCC"
SuccPred
                    | "PRED"
/******************* IO Statements
                 --> "SAY" "(" Expression ")"
Say_Statement
                --> "ASK" "(" string "," Variable ")"
Ask_Statement
Conditional_Statement --> IfThenElse_Stat
```