Decrição da Linguagem

Linguagens Formais e Autómatos(LFA)

Daniel Marques, Francisco Morgado, Jorge Catarino,
Óscar Pimentel, Paulo Vasconcelos

universidade de aveiro
theoria poiesis praxis

DETI - Universidade de Aveiro

Daniel Marques, Francisco Morgado, Jorge Catarino,
Óscar Pimentel, Paulo Vasconcelos
(85070) danielfmarques@ua.pt, (85009) fmpfmorgado@ua.pt,
(85028) jorge.catarino@ua.pt, (80247) oscarpimentel@ua.pt
(84987) paulobvasconcelos@ua.pt

11 de junho de 2018

Conteúdo

1 Gramáticas

1.1 BaseGrammar

1.1.1 Instrução

Nesta regra *instruction* pode-se observar que a gramática aceita a definição das variáveis, podendo esta ser apenas uma variável simples (*simpVar*) ou uma variável com uma unidade associada (*unitVar*), aceita também um comando para imprimir uma variável que esteja dentro de parêntesis ou uma atribuição a uma variável de uma operação que façamos.
A variável é composta por letras maiúsculas e minúsculas de ***A*** a ***Z*** e eventualmente números compreendidos entre ***0*** e ***9***.

```
instruction returns[String varName]:
    // Variable declaration
    varType NAME                                    #varDec
    // Print/Read variable
    | COMMAND '(' NAME ')'                          #command
    // Value atribution to variable
    // (This also accepts values that are not the result of
        an operation)
    | NAME '=' operation                            #assignment
    // Operation without storing result or (most common)
        variable increment/decrement
    | operation                                     #soloOp
    ;
```

```
// Variable Types
varType:
    'simpVar'       #simple
    | 'unitVar'     #unit
    ;
```

```
// Commands
COMMAND: 'Print';
```

```
// Variable
// (Must start with a letter and may have digits)
NAME: [a-zA-Z] [a-zA-Z_0-9]*;
```

1.1.2 Operação

A regra *operation* dá prioridade à realização de operações que estejam dentro de parêntesis, depois acrescentou-se a possibilidade de realizar incrementos e decrementos a variáveis, de seguida temos a operação propriamente dita que aceita dois operandos em que cada um pode ser também uma operação. Estes operandos são separados por um operador numérico, que é definido na *BaseLexerRules*, que pode ser, por ordem de prioridade, uma potência, uma divisão ou multiplicação e por fim uma adição ou uma subtração.

```
operation returns [tipo ty] :
    '(' n=operation ')'                                  #par
    | NAME '++'                                          #increment
    | NAME '--'                                          #decrement
    | left=operation NUMERIC_OPERATOR right=operation    #op
    | NAME                                               #assignVar
    | value                                              #val
    ;
```

```
// Numeric Operators
NUMERIC_OPERATOR: OP01
                | OP02
                | OP03
                ;
OP01: '^';
OP02: '*'|'/';
OP03: '+'|'-';
```

1.1.3 Condições if-else e ciclos for/while/do-while

```
//CONDITIONALS SECTION
if_else:
    'if' '(' condition ')' (('{'statList'}')|stat?)
    ('else' (if_else|('{'statList'}')|stat?))?;
//LOOPS SECTION
loop:
    // FOR LOOP
    'for' '(' var=NAME ';' min=INT ';' max=INT ')'
    '{' statList '}'                                #loopFor
    // WHILE LOOP
    | 'while' '(' condition ')' '{' statList '}'#loopWhile
    // DO-WHILE LOOP
    | 'do' '{' statList '}'
    'while' '(' condition ')'                       #loopDoWhile
    ;
```

```
condition:
    left=conditionE CONDITIONAL_OPERATOR right=conditionE
                                                    #compare
    |conditionE                                      #soloCond
    ;
```

```
// Conditional Operators
CONDITIONAL_OPERATOR:
    '=='  // Equal
    | ('>'|'<') ('=')?  // Greater (or equal) OR Smaller (or
        equal)
    ;
```

```
conditionE:
    value           #condiEValue
    |NAME           #condiEVar
    ;
```

1.1.4 Potência

```
// Equivalent to "*10^"
pow: 'e' ('-')? (INT|REAL);
```

1.2 Unidades

```
create: 'create' 'unit' uname=unit 'named' NAME;

pow: 'raise' NAME 'to power of' INT;

compose:'compose' composedUnit 'named' NAME;
```

```
unit returns[String varName]:
                            NAME                #unitUNIT
                            ;

composedUnit returns[String varName]:
              NAME                              #cUnitName
              |'(' p=composedUnit ')'           #cUnitParents
              |left=composedUnit op=(':'|'*')
              right=composedUnit                #cUnitDivMult
              ;
```

1.3 Mais exemplos de testes