

Projeto de Linguagens Formais e Autómatos

Universidade de Aveiro

Jacinto Luflakio - 89162

Márcia Pires - 88747

Pedro Almeida - 89205

Ruben Menino - 89185

Samuel Duarte - 89222



Projeto de Linguagens Formais e Autómatos

Departamento de Eletrónica, Telecomunicações e
Informática

Universidade de Aveiro

Jacinto Lufilakio - 89162

Márcia Pires - 88747

Pedro Almeida - 89205

Ruben Menino - 89185

Samuel Duarte - 89222

2 de julho de 2019

Conteúdo

1	Introdução	1
2	Linguagem	2
2.1	VarGrammar	2
2.1.1	Entradas	2
2.1.2	Instruções	3
2.1.3	Instruções condicionais e ciclos	3
2.1.4	Expressões	4
2.1.5	Tipos	4
2.2	Dimensões	5
3	Manual de instruções	6
3.1	Compilação	6
3.2	Execução	7
4	Testes e Gestão de Erros	8

Capítulo 1

Introdução

O projeto de Linguagens Formais e Autômatos (LFA) consiste na criação de uma linguagem para análise dimensional (física). Pretende-se estender o sistema de tipos de uma linguagem de programação com a possibilidade de definir dimensões distintas a expressões numéricas. Isto implica que apenas se podem efetuar operações sobre variáveis da mesma dimensão, podendo essa operação criar uma nova dimensão.

Este documento está dividido em quatro capítulos. Depois desta introdução, no Capítulo 2 são apresentadas as gramáticas desenvolvidas (*VarGrammar* e *Dimensoes*). De seguida, no Capítulo 3 é o manual de instruções dividido em duas secções: Seção 3.1 onde são apresentados os comandos para a realização da compilação; Seção 3.2 onde apresentados os comandos para a execução. Finalmente, no Capítulo 4 são apresentados alguns testes realizados assim como a gestão de erros.

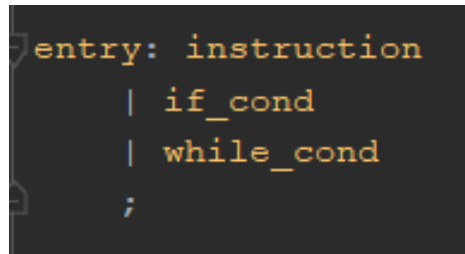
Capítulo 2

Linguagem

Neste capítulo serão apresentadas as gramáticas desenvolvidas *VarGrammar.g4* (Seção 2.1) e *Dimensoes.g4* (Seção 2.2).

2.1 VarGrammar

2.1.1 Entradas



```
entry: instruction
    | if_cond
    | while_cond
    ;
```

Figura 2.1: VarGrammar.g4 - Entry

A gramática *VarGrammar.g4* permite usar 3 tipos de entradas, sendo elas: uma entrada do tipo instrução e duas do tipo condição (*if* e *while*).

2.1.2 Instruções

```
21  instruction:
22      declaration
23      | assignment
24      | increment
25      | show
26      ;
```

Figura 2.2: VarGrammar.g4 - Instruções

Como se pode observar na imagem acima:

- **declaration** - com ou sem tipo (posteriormente explicado) e atribuir-lhes um valor ou uma expressão;
- **assignment** - isto é, atribuir um valor ou expressão a uma variável previamente declarada;
- **increment** - aumentar ou decrementar o valor de uma variável inteira em 1;
- **show** - imprime no terminal um valor ou o valor da expressão de uma variável;

2.1.3 Instruções condicionais e ciclos

```
if_cond: 'if(' condition ')' (' instIf=instruction ')? ('else(' instElse=instruction ')')?';
while_cond: 'while(' condition ')' (' instruction ');
```

Figura 2.3: VarGrammar.g4 - Instruções

Como instruções condicionais, é possível usar um *if..else* e como ciclos o *while*.

Ambas as instruções são usadas como qualquer outra linguagem de programação (exemplo: java, c, etc...). Na instrução *if*, se a condição for verdadeira são executadas as instruções desse *if* e, de seguida, salta as instruções dentro do *else*. No caso de a condição ser falsa, as instruções dentro do *if* não são executadas e passa para as instruções dentro do *else*. No caso *while*, as instruções dentro do "corpo" do *while* serão executadas até que a condição inicial seja falsa.

Na imagem seguinte, estão presentes os operadores condicionais que se podem utilizar.

```
COND_OPER: '<' | '>' | '<=' | '>=' | '==' | '!=';
```

Figura 2.4: VarGrammar.g4 - Operadores condicionais

2.1.4 Expressões

Uma expressão pode ser:

- identificadores (variáveis);
- números reais, inteiros com ou sem identificador (dimensão);
- operações entre expressões (soma, diferença, produto e quociente);
- expressões com parênteses (essencialmente para prioridade entre operações).

```
expression returns[Type tipo, String varName]:
    '(' expression ')' #ExpressParent
    | left=expression op=('*' | '/') right=expression #ExpressMulDiv
    | left=expression op=('+' | '-') right=expression #ExpressAddSub
    | num=(REAL|INT) IDENTIFIER #ExpressDimensao
    | REAL #ExpressReal
    | INT #ExpressInt
    | IDENTIFIER #ExpressID
    ;
```

Figura 2.5: VarGrammar.g4 - Expressões

2.1.5 Tipos

A regra *type* é utilizada para definir o tipo das variáveis. Uma variável pode ser do tipo *integer* (números inteiros), *real* e *unidade*.

```
type returns[Type t_tipo] :
    'integer' #TypeInt
    | 'real' #TypeReal
    | 'unidade' #TypeUnit
    ;
```

Figura 2.6: VarGrammar.g4 - Tipos

2.2 Dimensões

A gramática *Dimensoes-g4* serve exclusivamente para a criação de novas unidades e para a conversão entre as mesmas.

```
11 main: stat* EOF;
12
13 stat:  declaration
14      | convert;
15
16 declaration: ID '=' expression;
17
18 convert: id1=ID '->' NUM id2=ID;
```

(a) Dimensoes.g4 - parte 1

```
20 expression: ARRAY;
21 ID: [A-Za-z][A-Za-z]*('/' ?[A-Za-z])*;
22 NUM: [-+]?[0-9]+ ('.' ?[0-9]+)*;
23 ARRAY: '[' NUM ',' NUM ',' NUM ',' NUM ']' ;
24 WS: [ \t\r\n]+ -> skip;
25 COMMENT: '//' .*? '\n' -> skip;
26 ERROR: .;
```

(b) Dimensoes.g4 - parte 2

Figura 2.7: Dimensoes.g4

Como se pode observar a declaração de uma nova unidade é do tipo:

`ID = expression`

onde a *expression* é um *ARRAY*. Cada posição do *ARRAY* corresponde a uma medida *SI*. Neste caso, [distância, tempo, massa, temperatura]. O valor de cada posição do *ARRAY* é o grau do expoente. Exemplo: o *ARRAY* [1,-2,0,0] produzirá m/s^2 , ou seja, aceleração.

A conversão de unidades é do tipo:

`ID -> NUM ID`

onde o primeiro *ID* é no nome da unidade de destino, *NUM* é o fator de conversão e, finalmente, o segundo *ID* é o nome de uma unidade conhecida, ou seja, *SI*. Por exemplo, para a conversão de metros para centímetros seria: "cm -> 0.01 m". A partir deste momento, é possível fazer operações com a unidade "cm".

No Capítulo 4 será apresentado resultados e testes sobre esta gramática.

Capítulo 3

Manual de instruções

Antes de começar a escrever o programa principal, o ficheiro *progDimen.txt* deverá ser programado com as unidades e conversões necessárias para o programa principal. Este ficheiro encontra-se já com algumas unidades e conversões. É da responsabilidade do utilizador a verificação da declaração das unidades que pretende utilizar e, caso não se verifique, faça a declaração prévia das mesmas. O mesmo se aplica para as conversões entre as respetivas unidades.

```
1 // [metro, segundo, kilograma, kelvin]
2 m = [1, 0, 0, 0]
3 s = [0, 1, 0, 0]
4 kg = [0, 0, 1, 0]
5 k = [0, 0, 0, 1]
6
7 m/s = [1, -1, 0, 0] // velocidade
8 m/ss = [1, -2, 0, 0] // aceleracao
9 n = [1, -2, 1, 0] // newton
10 Pa = [-1, -2, 1, 0] // pascal
11 J = [2, -2, 1, 0] // joule
12 W = [2, -3, 1, 0] // watt
```

(a) Declaração de unidades

```
14 // Conversoes distancia
15 km -> 1000 m
16 hm -> 100 m
17 dam -> 10 m
18 dm -> 0.1 m
19 cm -> 0.01 m
20 mm -> 0.001 m
```

(b) Exemplo de conversões

Figura 3.1: Ficheiro *progDimen.txt*

3.1 Compilação

Os comandos para executar a compilação são apenas dois e são os seguintes:

O ficheiro *progDimen.txt* contém as unidades, podendo este ficheiro ser alterado para a criação de mais ou menos unidades consoante a necessidade do utilizador.

O ficheiro *prog0.txt* é o programa em si, isto é, o programa que utiliza as unidades criadas no ficheiro anterior.

```
antlr4-build  
java -ea VarGrammarMain progs/progDimen.txt progs/prog0.txt
```

Figura 3.2: Comandos para Compilar

3.2 Execução

Como em qualquer outra linguagem de programação, não pode ser efetuada a execução do programa sem antes ter efetuado a sua compilação (Seção 3.1).

Para a execução é necessário executar os seguintes comandos:

```
gcc Output.c Utils.c  
./a.out
```

Figura 3.3: Comandos para Executar

Capítulo 4

Testes e Gestão de Erros

```
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ java -ea VarGrammarMain progs/progDimen.txt progs/prog6.txt
progs/progDimen.txt is correct
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ gcc Output.c Utils.c
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ ./a.out
604800.000 s
1814400.000 s
0.999 s
1.000 s^2
1000000000.000
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$
```

Figura 4.1: Compilação e execução do *progs/prog6.txt*

```
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ java -ea VarGrammarMain progs/progDimen.txt progs/prog8.txt
progs/progDimen.txt is correct
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ gcc Output.c Utils.c
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ ./a.out
240
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$
```

Figura 4.2: Compilação e execução do *progs/prog8.txt*

```
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ java -ea VarGrammarMain progs/progDimen.txt progs/err1.txt
progs/progDimen.txt is correct
[ERROR at line 3] Variable "b" does not exists
[ERROR at line 5] Variable "f" already declared
[ERROR at line 6] Variable "d" does not exist!
[ERROR at line 7] Expression type does not conform to variable "f" type
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$
```

Figura 4.3: Mensagem de erro ao compilar o do *progs/erro1.txt*

```

utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ antlr4-build
Processing ./VarGrammar
Processing ./Dimensoes
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ java -ea VarGrammarMain progs/progDimen.txt progs/err2.txt
progs/progDimen.txt is correct
[ERROR at line 3] Different types!
[ERROR at line 6] Can't sum/sub different types
[ERROR at line 7] Variable "l" does not exist!
[ERROR at line 8] Variable "l" does not exist!
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ █

```

Figura 4.4: Mensagem de erro ao compilar o do *progs/err2.txt*

```

utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ java -ea VarGrammarMain progs/progDimen.txt progs/err3.txt
progs/progDimen.txt is correct
[ERROR at line 1] Can't sum/sub different types
[ERROR at line 5] Can't sum/sub different types
[ERROR at line 6] Can't divide numeric by unit
utilizador@Menino:~/Desktop/LFA/TrabalhoLFA/lfa-1819-g02$ █

```

Figura 4.5: Mensagem de erro ao compilar o do *progs/erro3.txt*

Contribuições dos autores

O Samuel fez a gramática das dimensões, o compilador e trabalhou no Utils. O Pedro fez o string template group e trabalhou no Utils. A Márcia trabalhou na análise semântica do VarGrammar e no Utils e fez os programas de teste. O Jacinto fez a gramática VarGrammar e trabalhou na análise semântica da mesma. O Ruben fez a análise semântica da Dimensoes e o interpretador da mesma.

O trabalho foi feito de modo a obter uma maior produtividade de todo o grupo e conseguir organizar um trabalho justo e em equipa. Por esses mesmos motivos, todos os membros realizaram cerca de 20 % do trabalho.

- Jacinto Lufilakio - 20%
- Márcia Pires - 20%
- Pedro Almeida - 20%
- Ruben Menino - 20%
- Samuel Duarte - 20%

Acrónimos

LFA Linguagens Formais e Autómatos