Processamento de Linguagens

Ficha Prática Nº2 (GIC)

2019/2020

1 Objetivos da Ficha

Servem os exercícios desta ficha para introduzir as **Gramáticas Independentes de Contexto** (GIC) como formalismo para definição de **linguagens de programação** (LP) ou **linguagens de domínio específico** (DSL).

Essas GIC serão usadas em exercícios posteriores para gerar automaticamente os **Analisadores Sintáticos** (Parsers) para reconhecer as frases das respetivas linguagens e até para gerar automaticamente os **Tradutores** ou **Processadores** finais para essas frases.

Serve também esta folha para praticarem o uso de **Expressões Regulares** (ER) para definir os **símbolos terminais** (T) da Linguagem e assim gerar automaticamente os **Analisadores Léxicos** (AL) que irão reconhecer os ditos símbolos terminais, ou **Tokens**, para os devolver ao Parser.

2 Exercício modelo resolvido

Para resolver os exercícios desta ficha é importante recordar que:

Texto é uma sequência de carateres;

Frase é uma sequência de símbolos que pertencem a um conjunto designado por **Vocabulário** ou **Alfabeto**;

Linguagem é um conjunto de frases;

Gramática é o formalismo matemático para especificar uma linguagem[1] formado por 4 elementos

$$< T,\ N,\ S,\ P >$$

- **T** o Alfabeto ou conjunto dos símbolos Terminais, vulgarmente designados por **Tokens**, que irão aparecer nas frases válidas;
- **N** o conjunto dos símbolos Não-terminais que não aparecem nas frases nas que definem os conceitos que se pretendem descrever com as frases da linguagem;
- **S** o **Axioma** ou **Símbolo Inicial**[2] que é um elemento singular de N pelo qual começa todo o processo de derivação das frase[3];
- **P** o conjunto de **Produções** ou **Regras de Derivação** em que cada regra é da forma

$$p:\ X_0\ \rightarrow\ X_1 \ldots X_i \ldots X_n,\quad X_0 \in N\ \wedge\ X_i \in (N \cup T)$$

[1] Isto é, para definir todas as frases válidas da linguagem, ou seja, estabelecer as sequências de símbolos que pertencem à linguagem, rejeitando todas as sequências inválidas, que não estão no conjunto.

[2] Em inglês, *S(tart Symbol)*.

[3] Relembre que uma frase dada só pertence à linguagem se se provar que deriva do Axioma e que também se só é válida a frase obtida por derivação a partir do Axioma.

Recorde que, dado um Alfabeto T, a transformação de um *texto* numa *frase* é conseguida através de um processador chamado **Analisador Léxico** (AL) que lê o texto e identifica as subsequências de carateres que correspondem a cada Terminal retornando o respetivo código[4].

2.1 Linguagem de Listas Mistas

Neste exercício pretende-se que defina formalmente uma linguagem para escrita de listas de elementos (um ou mais) escritos entre as palavras-reservadas INICIO e FIM, em que cada elemento da lista pode ser uma palavra ou um número inteiro.

Para especificar, ou definir formalmente, uma linguagem será necessário escrever uma GIC como a que se mostra a seguir, a qual tem:

- 6 produções conforme se pode ver mais abaixo,
- 4 símbolos não-terminais: `Frase`, o Axioma da gramática; `Lista`, para dizer que esse conceito é formado por um elemento ou por um elemento seguido de uma cauda separados por uma virgula; `Cauda`, para dizer que a cauda que continua o primeiro elemento é novamente (recursivamente) uma lista; e `Elem`, para dizer que cada elemento da lista poder ser uma palavra ou um número,
- 5 símbolos terminais: as palavras reservadas `INICIO` e `FIM`, o sinal `VIRG` (a vírgula ','), e os símbolos variáveis `pal` e `num` para designar respetivamente qualquer sequência de letras ou qualquer sequência de dígitos.

As regras de derivação, que dizem como combinar os símbolos Terminais e por que ordem para se obterem as frases válidas da linguagem, são:

```
P= {
   1: Frase --> INICIO Lista FIM
   2: Lista --> Elem
   3:         |  Elem VIRG Cauda
   4: Cauda --> Lista
   5: Elem  --> pal
   6:         |  num
   }
```

De acordo com esta gramática, pertencem à *linguagem das listas mistas* frases como:

```
INICIO 1 FIM
INICIO exemplo FIM
INICIO 1, 2, 3 FIM
INICIO  e, um ,  exemplo FIM
INICIO  e  ,  1, exemplo , com , 6,palavras FIM
```

Note que os dois primeiros exemplos mostram as duas frases mais curtas que se podem derivar da GIC acima.

Para construir um AL para ler um texto e identificar os símbolos Terminais é necessários escolher os seus códigos. Considere para isso a tabela seguinte (uma das atribuições possíveis):

```
INIC   5
FIM    6
VIRG   7
pal    8
num    9
```

[4] O número inteiro que identifica univocamente cada Token.

Feita esta associação, o AL pretendido pode ser especificado em Flex da seguinte forma:

```
%{
#define INIC   5
#define FIM    6
#define VIRG   7
#define pal    8
#define num    9
%}
%option noyywrap
%%
[,]           { return(VIRG); }
(?i:inicio)   { return(INIC); }
(?i:fim)      { return(FIM); }
[a-zA-Z]+     { return(pal); }
[0-9]+        { return(num); }
.|\n          { ; }
%%
```

3 Exercícios para resolver

3.1 Linguagem Lisp

A linguagem de programação funcional LISP, a primeira que surgiu e que foi usada muitos anos antes de aparecer a linguagem funcional ML sucedida depois pela Haskell, é formada exclusivamente por *Symbolic Expressions* (vulgarmente denotadas por SExp) e tem uma sintaxe incrivelmente simples como se ilustra nos exemplos abaixo de 4 programas Lisp válidos:

```
12
nil
(add 1 2)
(mul (add 3 2)(sub (power 2 5) 10))
```

para definir essa famosa linguagem de programação pode escrever-se a GIC seguinte:

```
1: Lisp --> SExp
2: SExp --> pal
3:       |  num
4:       | "(" SExpLst ")"
5: SExpLst --> SExp SExpLst
6:          |  &
```

Nesta gramática os símbolos `Lisp`, `SExp` e `SExpLst` são não-terminais e os símbolos `pal` e `num` são terminais variáveis ou classes. Nesta GIC só há 2 símbolos terminais do tipo sinal, os parêntesis curvos esquerdo e direito, e não há palavras-reservadas.

Analise então com cuidado a gramática apresentada e explique por palavras suas o que é um programa Lisp válido.

Escolha o código dos símbolos a seu gosto e especifique em Flex um AL para a linguagem Lisp.

3.1.1 Resolução

A GIC acima define a Linguagem de Programação Funcional LISP dizendo que um programa nesta linguagem é uma *Symbolic Expression* (SExp) que é uma de três coisas: uma palavra; um número; ou uma lista de *Symbolic Expressions* entre parêntesis curvos. Por sua vez a dita lista pode ser

vazia ou então tem uma cabeça que é uma *Symbolic Expression* e uma cauda que é de novo uma lista.

O Analisador Léxico pretendido pode ser especificado em Flex como se mostra abaixo. Porém neste caso e como o AL vai ser usado em combinação como o *Parser* que vai ser gerado pelo Yacc o código dos símbolos terminais é gerado automaticamente pelo referido Yacc e os define respetivos que associam os tokens aos códigos são inserido no ficheiro `y.tab.h` que terá de ser incluído na especificação abaixo. Além disso os sinais da linguagem, os parêntesis curvos, serão representados pelo respetivo código ASCII.

```
%{
#include "y.tab.h"
%}
%option noyywrap
%%
[()]                  { return(yytext[0]); }
[a-zA-Z]+             { return(pal); }
[0-9]+(\.[0-9]+)?     { return(num); }
.|\n                  { ; }
%%
```

Para que se interliguem as várias partes, lista-se abaixo a versão da GIC em notação Yacc e indica-se no fim como gerar um Reconhecedor para a linguagem LISP a partir das 2 especificaçõs e dos dois geradores de código C.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
%}

%token pal
%token num

%%
Lisp : SExp
     ;
SExp : pal
     | num
     | '(' SexpList ')'
     ;
SexpList : SExp SexpList
         |
         ;
%%
#include "lex.yy.c"

int yyerror(char *s) { printf("ERRO: %s\n",s); return(0); }

int main(){
    yyparse();
    return 0;
}

//---------como gerar e invocar o Processador 'lisp' :
```

```
> yacc -d lisp.y
> flex lisp.l
> gcc -o lisp y.tab.c
> lisp < prg1.lisp
```

O Parser pode agora ser estendido para processar cada frase de entrada, isto é, para produzir resultados enquanto faz a análise léxico-sintática.
Por exemplo para calcular o comprimento da lista (contar os seus elementos) e para contar o número de listas que aparecem
Para obter esse resultado basta acrescentar Ações Semânticas de contagem e adição às produções da gramática, como se vê abaixo.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

int conta=0;
int comp=0;
%}

%token pal
%token num

%%
Lisp : SExp
     ;
SExp : pal
     | num
     | '(' SexpList ')'   { conta++; }
     ;
SexpList : SExp SexpList  { comp++; }
         |
         ;
%%
#include "lex.yy.c"

int yyerror(char *s) { printf("ERRO: %s\n",s); return(0); }

int main(){
    yyparse();
       printf("Comprimento da Lista : %d\n",comp);
       printf("Número de Listas encontradas : %d\n",conta);
    return 0;
}
```

Mais interessante ainda é somar os valores dos números que aparecem nas listas. Para isso é preciso passar o valor de cada número, a partir do Analisador Léxico, antes de retornar o respetivo código 'num', como se mostra abaixo.

```
[0-9]+(\.[0-9]+)?     { yylval.val=atof(yytext); return(num); }
```

feito isto, é preciso adaptar a Gramática Tradutora para associar ao símbolo terminal 'num' o respetivo valor léxico e depois então já se pode usar esse valor na Ação Semântica. O resultado é a GT abaixo.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

int comp=0;
float soma=0;
%}

%union{float val;}

%token pal
%token <val>num

%%
Lisp : SExp
     ;
SExp : pal
     | num                { soma += $1; }
     | '(' SexpList ')'
     ;
SexpList : SExp SexpList  { comp++; }
         |
         ;
%%
#include "lex.yy.c"

int yyerror(char *s) { printf("ERRO: %s\n",s); return(0); }

int main(){
    yyparse();
       printf("Comprimento da Lista : %d\n",comp);
       printf("Somatório total dos números das Listas : %f\n",soma);
    return 0;
}
```

3.2 Linguagem dos parêntesis

Todas as linguagens de programação incluem os parêntesis algures na liguagem como símbolos terminais. Normalmente, são usados para agrupar coisas. Neste exercício, considera uma linguagem muito simples constituída apenas por dois símbolos terminais: '(' e ')'. Especifica uma GIC para esta linguagem atendendo aos seguintes requisitos:

- As frases desta liguagem são constituídas por um número arbitrário de parêntesis;
- Qualquer que seja a posição numa frase, o número de ')' nunca poderá ser superior ao número de '(' até esse ponto;
- No fim, o número de '(' e ')' tem de ser igual.

Apresentam-se a seguir algumas frases válidas:

```
()
(()(()))
()()(())
```

3.3 Listas em Haskell

Especifica uma GIC para listas de inteiros em Haskell. Lembra-te que uma lista em Haskell pode ser vazia, singular, conter uma lista de inteiros em compreensão ou extensão e ainda aninhar sublistas. Apresentam-se a seguir algumas frases válidas desta linguagem:

```
[]
[17]
[1..10]
[1,2,[3,4],5,6]
[1..10,21..30]
[1,3,5,7,9,10..20,[23,25,30..40]]
```

3.4 Linguagem SQL

Como sabe a linguagem standard para interrogação e gestão de Bases de Dados é a SQL. Esta linguagem, criada para interrogar BDs, permite escrever instruções da forma:

```
SELECT * FROM table WHERE ((chv = 1) AND (tipo="AAA"))
ORDER BY nome GROUP BY regiao
```

Neste contexto e ignorando todos os outros comandos que podem ser escritos para inserir, remover e actualizar bases de dados, tabelas e registos, pretende-se que escreva uma GIC para definir essa linguagem que aceite frases como a ilustrada acima.

Adicionalmente e usando ERs, desenvolva um Analisador Léxico (AL) para reconhecer todos os símbolos terminais das frases interrogação da linguagem SQL acima exemplificada e devolver os respectivos códigos.

Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

3.5 Documento anotado em XML

Como sabe um Documento XML é um texto vulgar semeado de anotações, ou marcas, que são identificadores especiais (designados por *elementos XML*) intercalados entre os caracteres "<" e ">". Num documento XML bem formado, a cada *marca de abertura* corresponderá uma *marca de fecho*, que tem o mesmo identificador, mas que começa por "</" terminando na mesma em ">".

Dentro de cada *marca de abertura*, além do identificador do elemento, ainda podem aparecer triplos formados por um outro identificador (de atributo), pelo sinal "=" e pelo respectivo valor que é qualquer texto entre aspas.

Cada fragmento do documento (texto livre) entre marcas deve ser considerado em bloco como sendo o símbolo PCDATA.

Escreva a GIC que define a linguagem XML conforme acima explicado.

Desenvolva ainda um AL que receba um documento XML e devolva todos os símbolos terminais encontrados, a seguir resumidos: "<", ">", "</", "=", identificador (qualquer palavra formada por letras), valor, PCDATA.

3.6 Anuário dos Medicamentos brancos

Para auxiliar o Instituto Farmacêutico do Ministério da Saúde na gestão do novo lote de medicamentos brancos, pretende-se criar um sistema de consulta a esses medicamentos acessível a qualquer farmácia via um browser HTML. Esse sistema deve mostrar a informação agrupada por: classe de medicamentos no Symposium Terapêutico (uma página por classe, com os medicamentos

ordenados alfabeticamente); ou por fabricante (uma página única, com os medicamentos agrupados por fabricante).
Sobre cada medicamento é fornecida a seguinte informação: nome, código, classe, composição química, preço recomendado, fabricantes disponíveis e lista de medicamentos de marca equivalentes (respectivo nome e fabricante). Neste contexto o que se pretende é que:

- defina uma linguagem para descrever a informação envolvida no lote de medicamentos a considerar (essa linguagem terá que permitir definir inicialmente o ano a que o Symposium Terapêutico diz respeito e a lista das classes de medicamentos);
- desenvolva um AL para reconhecer todos os símbolos terminais dessa linguagem e devolver os respectivos códigos.
  Melhore o seu AL suportando cada *palavra-chave*, ou *palavra-reservada*, da linguagem em *maiúsculas* ou *minúsculas* e permitindo a inserção de *linhas de comentário* no meio de uma frase válida.

3.7 Linguagem de Programação infantil 1

Para ensinar crianças a programar construiu-se, com base num microprocessador e alguns sensores baratos, um brinquedo móvel, um tanque de guerra, comandável por software. O dito tanque é capaz de andar até estar próximo de um obstáculo, parar definitivamente todo o programa, avançar ou recuar um determinado número de passos (o tamanho em centímetros correspondente a 1 passo também é programável), virar à esquerda ou direita um determinado número de graus, reagir com uma das ações de avanço ou viragem anteriores se ouvir um som estranho ou se ficar escuro, disparar 1 vez a arma 1 ou disparar N vezes a arma 2.

Neste contexto, Escreva uma GIC que defina uma linguagem simples para programar o dito tanque de guerra. Essa linguagem, além de ativar as ações descritas, deve ainda permitir repetir ações (uma ou mais).

3.8 Linguagem de Programação infantil 2

Para ensinar às crianças o pensamento computacional que está na base da programação pretende-se criar um interpretador simples que permita descrever inicialmente uma base de conhecimento (BC) na forma de um conjunto de triplos <`sujeito, predicado, objeto`> e depois permita fazer uma ou mais perguntas sobre essa BC. As perguntas podem ser de vários tipos de modo a permitir saber todas as relações com um dado `sujeito` ou um dado `predicado`, ou então saber todos os pares <`sujeito, objeto`> relacionados através de um dado `predicado`. Os resultados das perguntas devem poder ser imprimidos de imediato ou ficar guardados em variáveis (que à moda do AWK podem ser usadas sem ser declaradas)

Neste contexto, Imagine uma linguagem simples e intuitiva que possa ser usada por uma criança para fazer a programação da BC e respetivas perguntas e, então, escreva uma GIC que defina a linguagem que conceber.

3.9 Lista de Compras

Pretende-se criar uma aplicação para gerir listas de compras. Um dos seus componentes é uma linguagem com a qual se poderão descrever essas listas.

Neste contexto, especifique uma GIC para descrever listas de compras atendendo aos seguintes requisitos:

1. Uma lista de compras está dividida em secções em que cada secção corresponde a uma categoria de compra: limpeza, frescos, peixe, carne, padaria, ...
2. Cada secção tem, por sua vez, uma lista de produtos a comprar;

3. Cada produto é caraterizado por: código, designação, preço (de referência) e quantidade a comprar;

4. A quantidade a comprar é um tuplo formado por tipo (unidade, peso, volume líquido, ...) e valor.

3.10 Tarefas dos funcionários de uma secretaria

Pretende-se uma linguagem de Domínio Específico que permita descrever as tarefas de cada funcionário de uma secretaria.

Para tal deve-se indicar, no início, a lista de funcionários, indicando para cada um o nome completo e o respetivo código, bem como a função. Depois então surge a lista de tarefas agrupadas por funcionário (agora já só identificado pelo respetivo código). Por cada tarefa indique o dia, hora, a prioridade (normal, urgente ou baixa), e a descrição da tarefa.

Escreva então uma Gramática Independente de Contexto, GIC, que especifique a Linguagem pretendida (note que o estilo da linguagem (mais ou menos verbosa) e o seu desenho são da sua responsabilidade).

Especifique em Flex um Analisador Léxico para reconhecer todos os símbolos terminais da sua linguagem e devolver os respetivos códigos.

3.11 Linguagem BibTeX

BibTeX é uma linguagem de Domínio Específico para descrever diferentes tipos (atualmente 20 variantes) de referências bibliográficas que podem ser citadas em documentos LaTeX, conforme se mostram 3 exemplos a seguir.

```
@incollection{MHL2015a,
  author = {Ricardo Martini and Pedro Rangel Henriques and Giovani Libreloto},
  title={Storing Archival Emigration Documents to Create Virtual Exhibition Rooms},
  booktitle={New Contributions in Information Systems and Technologies},
  series={Advances in Intelligent Systems and Computing},
  editor={Rocha, Alvaro and Correia, Ana and Costanzo, S. and Reis, Luis Paulo},
  volume={353},
  pages={403-409},
  year = {2015},
  month =   {April}
}
@InProceedings{MHC2015,
  author = {Vitor T. Martins and Pedro Rangel Henriques and Daniela da Cruz},
  title = {An AST-based tool, Spector, for Plagiarism Detection},
  booktitle = {Proceedings of SLATE'15},
  pages = {173--178},
  ISBN = {},
  year = {2015},
  month =   {},
  publisher = {Fundación General UCM},
  annote = {Keywords: software, plagiarism, detection, comparison, test}
}
@book{Oli91a,
    author = "José Nuno Oliveira",
    title = "Especificação e Semântica",
    year = 1991,
    edition = "1.st",
```

```
    publisher = "Departamento de Informática, Univ. do Minho"
}
```

Baseando-se nesta descrição responda às seguintes alíneas:

a) Escreva uma gramática independente de contexto (GIC) para a linguagem apresentada considerando que o tipo de cada registo (que aparece logo no início depois de `'@'`) e o nome dos campos dos registos são variáveis;

b) Especifique o respetivo analisador léxico usando a notação do *Flex*.