Processamento de Linguagens

3º Trabalho Prático

2017/2018

Vitor Peixoto (a79175)
Francisco Oliveira (a78416)
Raul Vilas Boas (a79617)

Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Processamento de Linguagens, foi-nos pedido para desenvolver algo capaz de descrever parte da rede semântica do Museu da Emigração.

Para tal desenvolvemos uma linguagem capaz de descrever os nodos pedidos e um processador capaz de pegar nessa linguagem e criar um grafo para visualização da rede.

Neste relatório será explicado o processo de desenvolvimento e as decisões tomadas ao longo da realização do trabalho.

Capítulo 2

Rede do Museu da Emigração

Neste trabalho desenvolvemos uma linguagem, que mais tarde seria convertida para Dot (*GraphViz*), usando um reconhecedor léxico e sintático (*Flex/Yacc*) por nós também desenvolvido.

Esta linguagem é capaz de suportar os 3 nodos esperados, *emigrante*, *obra* e *evento*, bem como os 2 arcos esperados, *fez* e *participou*.

Como extra, a nossa linguagem também aceita um nodo *wild* e arco *liga* simples, que tem alguma utilidade por não estarem tão limitados em alguns aspetos como iremos falar mais á frente. Temos também possibilidade de criar *subgraphs* (do tipo cluster).

2.1 Flex - Analisador Léxico

A nível léxico, devem-se referir algumas palavras importantes da nossa linguagem.

Sempre que queremos definir um nodo, colocamos o nome que pretendemos dar ao nodo, com um prefixo que indica o tipo de nodo. Os prefixos disponíveis são:

- **P_** para Emigrantes,
- **O_** para Obras,
- **E_** para Eventos,
- **W_** para um nodo indefinido.

Para alem dos nodos temos os arcos, que colocariamos associados a 2 nodos e seriam:

- **fez** para Obras feitas por um Emigrante;
- **participou** para Eventos visitados por um Emigrante;
- **liga** para ligações entre nodos indefinidos.

Temos também atributos que serão usados em vários sítios mais tarde, como:

- **label** para alterar o texto que irá aparecer dentro de um nodo, arco ou grafo;
- **href** para adicionar um hiperlink a um nodo;
- **tooltip** para alterar o texto da tooltip de um nodo.

Para alem de tudo isto temos tambem subgrafos, marcados com o prefixo *subgraph_* seguido do nome que o cluster terá.

Sempre que o analisador léxico encontra esta palavras ele irá substitui-las pelos respetivos *tokens* que serão recebidos pelo analisador sintático mais tarde.

Todos os referidos *tokens* podem ser vistos abaixo nos returns no código apresentado.

Código Flex

```
%{
    #include "y.tab.h"
%}

TEXT    [a-zA-Z][a-zA-Z0-9\-\_]*
WORD    [a-zA-Z0-9]+

%%

[ \t\r]          ;
[\n]             yylineno++;
[{}\[\]]         return *yytext;

digraph          {yylval.string = strdup(yytext); return GRAPH;}
subgraph_{WORD} {yylval.string = strdup(yytext); return SUBGRAPH;}

P_{TEXT}         {yylval.string = strdup(yytext); return EMIGRANTE;}
O_{TEXT}         {yylval.string = strdup(yytext); return OBRA;}
E_{TEXT}         {yylval.string = strdup(yytext); return EVENTO;}
W_{TEXT}         {yylval.string = strdup(yytext); return WILD;}

fez              return FEZ;
participou       return PARTICIPOU;
liga             return LIGA;

label=["][^"]*["]   {yylval.string = strdup(yytext); return LABEL;}
href=["][^"]*["]    {yylval.string = strdup(yytext); return HREF;}
tooltip=["][^"]*["] {yylval.string = strdup(yytext); return TOOLTIP;}

.                return ERRO;
```

2.2 Yacc - Analise Sintática

A nível de gramática, esta linguagem não foge muito do estilo de linguagem do *Dot*, contudo tem alguns detalhes que ajudam a prevenir erros na estruturação da tarefa apresentada, bem como melhorar a apresentação rapidamente e com menos esforço.

Passando á explicação, a nossa linguagem começa por indicar o tipo de grafo a desenhar, representado pelo token GRAPH. Apesar de apenas aceitarmos um tipo de grafo (digraph) no nosso trabalho, esta parte poderia ser expandida para atender ao requisito de mais tipos de grafos. De seguida delimitamos o conteúdo do grafo com chavetas { }.

Dentro destas encontramos os comandos que constituem o grafo. Estes incluem nodos e arcos, as principais componentes de um grafo, mas também a possibilidade de colocar uma *label* no grafo.

Temos também subgrafos que se comportam de forma igual ao grafo original, pelo que abrem chavetas, colocam os comandos pretendidos e fecham chavetas.

O nodo aceita qualquer um dos tipos vistos anteriormente (Emigrante, Obra, Evento, Wild) seguidos dos seus atributos dentro de parênteses retos [ ]. Os atributos que aceitamos são labels (LABEL) para alterar o texto que irá aparecer dentro do nodo, links (HREF) que permite que quando um nodo seja clicado abra o link inserido e tooltips (TOOLTIP) que permite alterar a tooltip de qualquer nodo. Referir que o texto da label irá ficar azul quando o nodo tiver um link associado, para facilmente saber quais os nodos com links. A nível de atributos qualquer ordem pode ser inserida, bem como repetir atributos e todos serão processados por nós, contudo aquando da compilação pelo *Dot* apenas o último irá prevalecer sobre os restantes repetidos.

O arco aceita qualquer um dos tipos vistos anteriormente (Fez, Participou, Liga) e apenas aceita labels como atributos, para alterar o texto que irá aparecer sobre o arco.

As produções criadas satisfazem a condição LR.

Produções Yacc

```
NotDot  : Graph
        ;

Graph   : GRAPH '{' Comands '}'
        ;

Comands : Comands Comando
        |
        ;

Comando : Nodo
        | Arco
        | LABEL
        | SUBGRAPH '{' Comands '}'
        ;

Nodo    : EMIGRANTE '[' AtribsN ']'
        | OBRA '[' AtribsN ']'
        | EVENTO '[' AtribsN ']'
        | WILD '[' AtribsN ']'
        ;

AtribsN : AtribsN AtribN
        |
        ;

AtribN  : LABEL
        | HREF
        | TOOLTIP
        ;

Arco    : EMIGRANTE FEZ OBRA '[' AtribsA ']'
        | EMIGRANTE PARTICIPOU EVENTO '[' AtribsA ']'
        | WILD LIGA WILD '[' AtribsA ']'
        ;

AtribsA : AtribsA AtribA
        |
        ;

AtribA  : LABEL
        ;
```

2.3 Exemplos

Aqui podemos ver um pequeno exemplo com o código original na linguagem por nós criada, a versão gerada em *Dot* depois de processado e finalmente a imagem gerada.

Código original

```
digraph {
    subgraph_TheWilds{
        label="Exemplo Subgraph"
        W_GOOGLE [label="GOOGLE" href="http://www.google.com"]
        W_MUSEU  [label="Museu\nEmigrantes"
                  href="http://www.museu-emigrantes.org"]
    }
    P_Pedro   [label="Pedro"]
    P_Antonio [label="Antonio|Canalizador"]
    O_MIEI [label="MIEI"]
    E_paredes [label="Paredes de Coura"
               href="https://www.paredesdecoura.com/"]
    E_SaoJoao [label="São João"]

    W_GOOGLE liga W_MUSEU           [label="arco"]
    P_Pedro fez O_MIEI              [label="fez"]
    P_Pedro participou E_paredes    [label="participou"]
    P_Antonio participou E_SaoJoao  []
}
```

Dot gerado

```
digraph{
    subgraph cluster_TheWilds{
    label="Exemplo Subgraph"
    W_GOOGLE [shape=circle label="GOOGLE" tooltip="GOOGLE"
              href="http://www.google.com" fontcolor="blue"]
    W_MUSEU  [shape=circle label="Museu\nEmigrantes"
              tooltip="Museu\nEmigrantes"
              href="http://www.museu-emigrantes.org" fontcolor="blue"]
    }
    P_Pedro   [shape=Mrecord penwidth=4 label="Pedro" tooltip="Pedro"]
    P_Antonio [shape=Mrecord penwidth=4 label="Antonio|Canalizador"
               tooltip="Antonio|Canalizador"]
    O_MIEI    [shape=doublecircle label="MIEI" tooltip="MIEI"]
    E_paredes [shape=Msquare label="Paredes\nde Coura"
               tooltip="Paredes\nde Coura"
               href="https://www.paredesdecoura.com/" fontcolor="blue"]
    E_SaoJoao [shape=Msquare label="São João" tooltip="São João"]
    W_GOOGLE  -> W_MUSEU   [label="arco"]
    P_Pedro   -> O_MIEI    [style=bold label="fez"]
    P_Pedro   -> E_paredes [style=dashed label="participou"]
    P_Antonio -> E_SaoJoao [style=dashed]
    label="Exemplo de um Grafo"
}
```

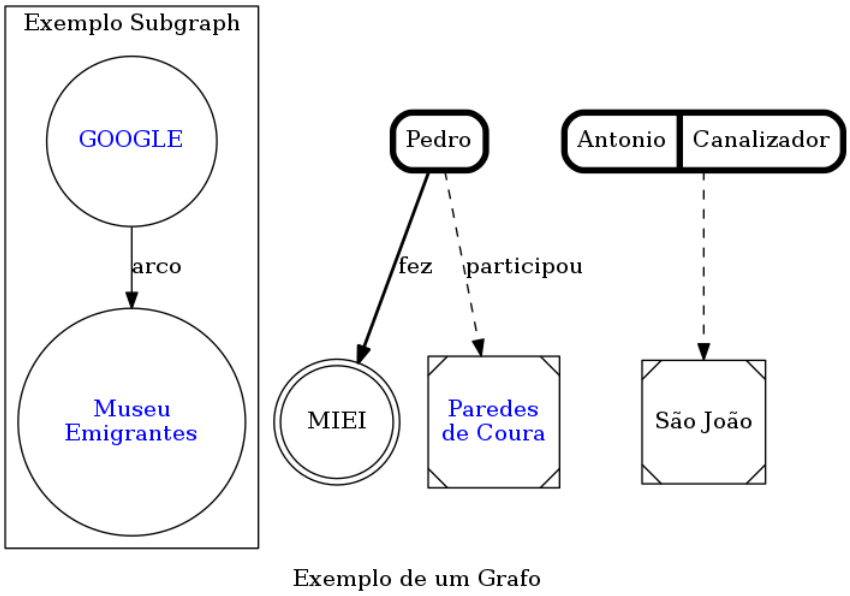

Figura 2.1: Grafo gerado

Capítulo 3

Conclusão

Este trabalho permitiu-nos melhor entender o par *Flex/Yacc* e como o utilizar para criar um compilador de uma linguagem por nós criada, para uma mais conhecida, neste caso *Dot*.

O objetivo do trabalho foi atingido, contudo algumas partes podiam ser melhoradas como adicionar atributos nos arcos ou até mais atributos em geral.

Avaliamos assim a nossa prestação como positiva.