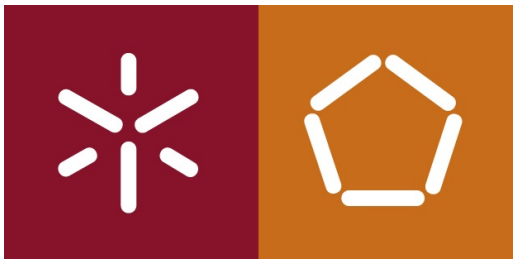

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens e Compiladores

2017/2018 - 2ºsemestre

Trabalho Prático nº 3 - YACC

Autores :
Diana Costa (A78985)

Marcos Pereira (A79116)

Vitor Castro (A77870)

Braga, 13 de Junho de 2018

Resumo

Perante a proposta de realizar um exercício sobre gramáticas independentes do contexto e processadores de linguagens segundo o método da tradução dirigida pela sintaxe (suportado numa gramática tradutora), utilizando o par **lex/yacc**, houve um impasse inicial, devido à necessidade de uma boa estruturação dos problemas. O nível de dificuldade e complexidade deste projeto, quando comparada com o primeiro exercício sobre GAWK, ou o segundo sobre FLEX, é bastante superior, e levou a que o grupo se debatesse e fizesse um esforço conjunto para perceber como aplicar os conhecimentos. Tudo isto requeriu a escolha e desenvolvimento acertado da gramática, expressões regulares e respetivas ações semânticas, estruturas de dados globais para armazenar os dados ao longo do processamento, e uma criação/análise aprofundada de extratos de texto (*datasets*), de forma a que a resolução do exercício fosse a mais clara e simples possível.

Depois de algum tempo e trabalho, o resultado encontrado foi satisfatório, e os objetivos e respostas às questões do enunciado proposto cumpridos.

Conteúdo

1 Introdução

O objetivo deste trabalho prende-se com o aumento da experiência em ambiente *Linux*, em rever e aumentar a capacidade de escrever gramáticas I.C. que satisfaçam a condição LR(), desenvolvimento de processadores de linguagens(segundo o método da tradução dirigida pela sintaxe, suportado numa gramática tradutora) e utilização do par *flex/yacc* como geradores de compiladores de texto. De entre os enunciados disponibilizados, o grupo ficou com a tarefa de desenvolver o número 1, - **Rede Semântica do Museu da Emigração**. Essencialmente, pretende-se descrever a rede semântica que suporta o Museu da Emigração e da Luso-descendência e gerar, posteriormente, um grafo (usando GraphViz / WebDot) que permita uma navegação conceptual sobre esse repositório de conhecimento: percorrer os caminhos do grafo, selecionar um nodo e saltar para uma página com info sobre esse nodo.

Assim, ao nível deste enunciado, era requerido que se procedesse inicialmente à especificação da gramática concreta da linguagem de entrada. Posto isto, seria necessário desenvolver um reconhecedor léxico e sintático para a mesma linguagem (utilizando as ferramentas geradoras flex/yacc), e, finalmente, construir o gerador de código para produzir a resposta solicitada. Este gerador de código é construído associando ações semânticas de tradução às produções da gramática, recorrendo mais uma vez ao gerador yacc.

Em suma, a Secção 2 descreve os preliminares necessários ao projeto, enquanto que a Secção 3 descreve a resolução do enunciado 1 do trabalho, com todos os pormenores da gramática desenvolvida(3.2.1 a 3.2.4). Apresentam-se, na Secção 4 alguns testes e resultados obtidos, e o relatório termina com uma breve conclusão na Secção 5, onde é feito um balanço do trabalho realizado, tendo em conta as dificuldades ao longo do desenvolvimento do mesmo. Em anexos (6) apresenta-se o código desenvolvido pelo grupo.

2 Preliminares/Contextualização

Considera-se que, para o desenvolvimento deste exercício prático e para uma compreensão inicial do esquema utilizado no mesmo, é necessária uma breve e simples explicação do que faz o par lex/yacc e de como o grupo modelou o trabalho.

Assim, o yacc e o gerador de analisador léxico lex são geralmente usados em conjunto. O Yacc usa uma gramática formal para analisar sintaticamente uma entrada, algo que o lex não consegue fazer somente com expressões regulares (o lex é limitado a simples máquinas de estado finito). Entretanto, o yacc não consegue ler a partir duma simples entrada de dados, ele requer uma série de tokens, que são geralmente fornecidos pelo lex. O lex age como um pré-processador do yacc.

Desta forma, dividiu-se o trabalho em duas fases: na primeira, o código em flex faz o reconhecimento de cada símbolo encontrado no ficheiro, e, posteriormente, o yacc, dependendo do que é passado no flex, efetua uma ação, que terá funções/estruturas que serão utilizadas de um módulo auxiliar. Mais concretamente, as estruturas de dados utilizadas no ficheiro "emigrantes.y"pertencem à biblioteca *Glib*. Resume-se abaixo os três módulos constituintes deste modelo, e o encadeamento entre eles.

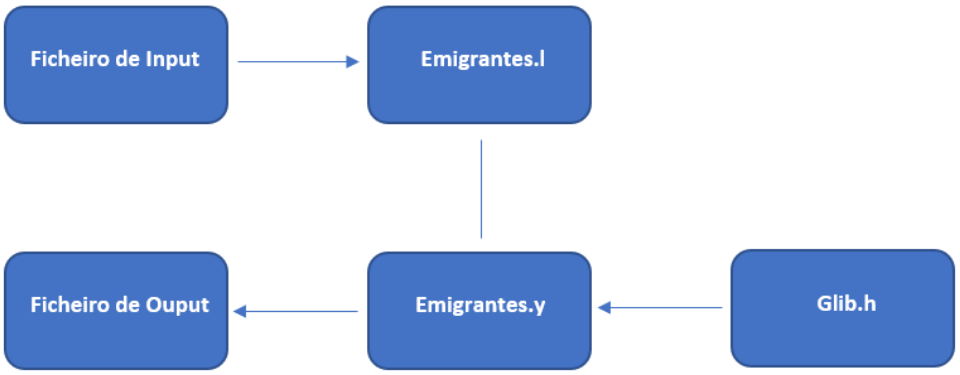

Figura 1: Módulos constituintes do modelo de trabalho utilizado pelo grupo

É de notar que, para o caso deste exercício em concreto, o ficheiro de output será um grafo, por razões que o grupo explicará mais à frente.

3 Rede Semântica do Museu da Emigracão - Análise e especificação da linguagem

Este tema visa a descrição de parte da rede semântica que suporta o Museu da Emigração, das Comunidades e da Luso-descendência [1], e a geração de um grafo (usando GraphViz / WebDot) de modo a permitir a navegação conceptual sobre este repositório de conhecimento. Deste modo, deverá ser possível, para além de percorrer os caminhos do grafo, selecionar um nodo e saltar para uma página com info sobre esse elemento. Para o efeito, a linguagem definida deve processar três tipos de nodos, ou vértices, da rede:

- emigrante - nodo com dados pessoais e dados relativos ao processo de emigração;
- obra - nodo com identificação de palácios, fábricas, hospitais, escolas, capelas, etc;
- evento - nodo com informação de bailes, saraus literários, saraus musicais, etc.

E os seguintes arcos:

- fez - liga emigrante a obra;
- participou - liga emigrante a evento;

3.1 Criação de datasets

Para o presente exercício, uma vez que não foram fornecidos quaiquer extratos de texto, o grupo definiu um ficheiro com o que seria a linguagem do exercício, como se apresenta abaixo:

```
emigrante joao:
  nome: "Antonio Joao Joaquim"
  nascimento: "1890-11-24"
  morte: "1972-03-14"
  natural: "Portugal"
  partida: "1904-09-13"
  destino: "Brasil"
  url: "https://www.google.pt/search?q=antonio+joao+joaquim&source=
       lnms&tbm=isch&sa=X&ved=0ahUKEwiMn8Sb6cPbAhUIuhQKHazRDJMQ_AU
       ICigB&biw=1920&bih=974#imgrc=6TU1D4xJKAN5cM:"

obra hospitalassis:
  nome: "Hospital Manuel de Assis"
  data: "1940-02-01"
  local: "Sao Paulo"

obra escolapotassio:
  nome: "Escola Cloreto de Potassio"
  data: "1943-04-20"
  local: "Para"

obra capelamoniz:
  nome: "Capela Martim Manhas"
  data: "1964-06-07"
  local: "Belo Horizonte"

evento baile real47:
  nome: "Baile Real da Realeza do Brasil numero 47"
  data: "1952-03-29"
  local: "Recife"

  (...)

joao fez hospitalassis
joao fez escolapotassio
joao fez capelamoniz

joao participou baile real47
joao participou inauguracaocapelamoniz

joana participou baile real47
joana fez orfanatoteixeira
```

Desta forma, são identificados emigrantes com a tag "emigrante", e declaradas as suas informações dentro dessa tag. O mesmo acontece para obras e eventos, onde cada uma tem um identificador e detalhes. Todos estes nodos do grafo serão correlacionados através das tags "fez"e "participou", de modo a unir emigrantes e eventos ou obras.

3.2 Gramática

Nesta secção define-se a linguagem criada pelo grupo, começando-se por analisar os símbolos terminais e não terminais, partindo depois para as produções. Como visto nas aulas da unidade curricular, uma gramática G deve ser definida da seguinte maneira:

- T representa o conjunto de símbolos terminais;
- N indica o conjunto de símbolos não-terminais;
- S descreve o axioma da gramática;
- P é o conjunto de regras de produções.

Assim sendo, a gramática ficará definida pela igualdade G={ T, N, S, P }.

3.2.1 Terminais

Os terminais são símbolos constituintes de uma gramática, podendo aparecer quer na entrada, quer na saída de uma produção. Estes mesmo não podem ser derivados em "unidades menores", e, por decisão do grupo, os terminais serão designados por caracteres maiúsculos.

T = { OBJECT_TYPE, STRING, OBJECT_ID, FIELD_ID, FEZ, PARTICIPOU, ERR }

3.2.2 Não-terminais

Os não-terminais são símbolos que pertencem a uma gramática, podendo aparecer apenas na saída de uma produção, ou mesmo ser derivados noutras produções, que podem ser compostas por mais símbolos terminais e não-terminais. O grupo definiu que estes símbolos seriam escritos todos com letra maiúscula.

N = { OBJECTS, OBJECT, FIELDS, FIELD, CONNECTION }

3.2.3 Axioma

O axioma é a raíz da árvore de derivação, e por onde se começa a primeira produção. Neste caso, designou-se o axioma por "OBJECTS", o qual derivar-se-á em objetos singulares e conexões.

S = { OBJECTS }

3.2.4 Produções

Depois de analisados todos os conjuntos acima, relacionam-se agora o conjunto de símbolos, usando produções. Estas produções (regras) são consideradas as strings válidas que se podem formar, a partir do alfabeto da linguagem. Assim, começando pelo axioma "objects", que pode ser descrito por o conjunto de nodos do grafo, é fácil deduzir que este pode derivar em vazio, num "object"simples seguido de mais "objects", ou em "objects"e uma "connection"entre dois "object". Considera-se que um "object"representa apenas um nodo do grafo, e pode ser derivado no seu tipo ("object_type"), id("object_id") e campos ("fields"). Por sua vez, "fields"pode ser derivado apenas num "field"ou em vários ("fields field"). Como é sabido, o campo será composto pelo seu id e por uma string. Por fim, e generalizando, uma conexão ("connection") será sempre entre dois "object_id", e pode ser do modo "fez"ou "participou", que explica o relacionamento entre um emigrante e os eventos e obras que ele participou ou elaborou, respetivamente.

```
P = {
    p0: OBJECTS -> OBJECTS OBJECT
    p1: OBJECTS -> OBJECTS CONNECTION
    p2: OBJECTS -> E
    p3: OBJECTS -> ;
    p4: OBJECT -> OBJECT_TYPE OBJECT_ID FIELDS
    p5: OBJECT -> ;
    p6: FIELDS -> FIELDS FIELD
    p7: FIELDS -> FIELD
    p8: FIELDS -> ;
    p9: FIELD -> FIELD_ID STRING
    p10: FIELD -> ;
    p11: CONNECTION -> OBJECT_ID FEZ OBJECT_ID
    p12: CONNECTION -> OBJECT_ID PARTICIPOU OBJECT_ID
    p13: CONNECTION -> ;
}
```

Apresenta-se, finalmente, a gramática elaborada pelo grupo num formato mais simples e legível.

```
OBJECTS : OBJECTS OBJECT
        | OBJECTS CONNECTION
        |
        ;

OBJECT : OBJECT_TYPE OBJECT_ID FIELDS
       ;

FIELDS : FIELDS FIELD
       | FIELD
       ;

FIELD : FIELD_ID STRING
      ;

CONNECTION : OBJECT_ID FEZ OBJECT_ID
           | OBJECT_ID PARTICIPOU OBJECT_ID
           ;
```

3.3 Ações semânticas e expressões regulares

Antes de declarar as expressões regulares e respetivas ações semânticas, é de bom senso explicar os padrões mais usados pelo grupo, como forma a simplificar a sua leitura.

Um ID poderá ser constituído por uma ou mais ocorrências de letras e números.

```
id          [a-zA-Z0-9]+
```

Um number será uma ou mais ocorrências de um número inteiro positivo. É de notar que o grupo acabou por não utilizar este padrão, mas manteve-o inalterado, pois achou útil num desenvolvimento futuro.

```
number      [0-9]+
```

Uma string será tudo o que aparecer, até chegar a uma aspa ou newline (uma ou mais ocorrências).

```
string      [^"\r\n]+
```

Um wspace ignora espaços em branco entre caracteres. Inclui o \r para compatibilidade com Windows.

```
wspace      [ \r\n]*
```

Por fim, um newline define o que são considerados caracteres indicativos de uma nova linha, e é compatível com Windows e Linux.

```
newline     (\r\n|\n)
```

Finalmente, declaram-se e explicam-se as expressões regulares e as respetivas ações semânticas.

Esta expressão regular apanha as palavras reservadas "emigrante", "obra", ou "evento", e comunica ao yacc que foi detetado um token do tipo OBJECT_TYPE. (Para além disso, passa o valor do tipo dentro de yylval.str, mas neste momento o yacc não faz nada com o valor do OBJECT_TYPE). Chama-se à atenção para, uma vez detetado isto, a entrada no contexto object_id.

```
(emigrante|obra|evento)     { printf("flex: Object type: %s\n", yytext);
                              BEGIN object_id;
                              yylval.str = strdup(yytext);
                              return OBJECT_TYPE; }
```

A expressão abaixo é responsável por apanhar um ID, dentro do contexto inicial, o que significa que se segue uma conexão entre dois nodos. Exemplificando, em "alberto fez hospital", tem-se que o emigrante "alberto"é um object_id de um object, e está ligado a um hospital pela conexão "fez". É de salientar que se entra no contexto "connection".

```
{id}                        { printf("flex: Link ID: %s\n", yytext);
                              BEGIN connection;
                              yylval.str = strdup(yytext);
                              return OBJECT_ID; }
```

Dentro do contexto "object_id", é necessário eliminar os espaços entre o tipo de um objeto e o seu ID. É isso que a próxima expressão regular faz.

```
<object_id>[ ]              { /* Eating the space between object type
                                 and its ID */ }
```

Ainda no mesmo contexto, é preciso apanhar o ID de um objeto.

```
<object_id>{id}              { printf("flex: Object ID: %s\n", yytext);
                               yylval.str = strdup(yytext);
                               return OBJECT_ID; }
```

No mesmo contexto, é mandatório que se apanhe o espaço em branco no fim de um object_id, e que se entre no contexto "field", uma vez que a seguir a um ID de um objeto (que acaba com dois pontos) vem um campo.

```
<object_id>:{wspace}         { BEGIN field; }
```

Mudando de contexto para "field", é necessário apanhar o ID de um campo. Por exemplo, em "emigrante alberto: nome "Abreu", está-se a apanhar "nome", e é dito ao yacc que este token é um "field_id".

```
<field>{id}                  { printf("flex: Field ID: %s\n", yytext);
                               yylval.str = strdup(yytext);
                               return FIELD_ID; }
```

Ainda dentro do contexto anterior, a expressão regular seguinte apanha os dois pontos no fim de um "field_id"(seguido de white spaces), e entra no contexto "field_value", que vai apanhar o valor do campo.

```
<field>\:{wspace}\"          { BEGIN field_value; }
```

Mudando de contexto para "field_value", a próxima expressão regular captura o valor do campo. Tomando como exemplo "emigrante alberto: nome "Abreu", estaria-se a capturar o nome "Abreu". É passada a informação ao yacc que o que foi capturado é um token do tipo string.

```
<field_value>{string}        { printf("flex: String: %s\n", yytext);
                               yylval.str = strdup(yytext);
                               return STRING; }
```

A próxima expressão regular captura, dentro do mesmo contexto, as duas new lines ou mais no fim de um campo de um objeto. Quando isto acontece, é sinalizador de que não existem mais campos a seguir e volta-se ao contexto inicial.

```
<field_value>\"[ ]*{newline}{2,} {
                               printf("flex: Two newlines after field value\n");
                               BEGIN INITIAL;
                               }
```

Por fim, neste contexto, a expressão abaixo apanha whitespaces no fim de um campo, até ao ponto de duas new lines seguidas, ou entrava-se na expressão regular acima. Isto acontece porque o flex escolhe o padrão que apanha a string mais comprida, e, em caso de empate, escolhe a primeira expressão regular que aparecer. Para além disto, sabe-se que, neste caso, aproxima-se um novo campo e é necessário voltar ao contexto field.

```
<field_value>\"{wspace}      { printf("flex: End of string (not two newlines)\n");
                               BEGIN field; }
```

Agora para o contexto connection, temos strings no formato "antonio fez asneiras". Desta forma, a próxima expressão captura o "fez"e retorna um token "FEZ".

```
<connection>{wspace}fez{wspace} { printf("flex: fez\n");
                                  return FEZ; }
```

A próxima expressão é semelhante à acima designada, com a exceção de que apanha "participou"em vez de "fez", e retorna um token "PARTICIPOU".

```
<connection>{wspace}participou{wspace}  { printf("flex: participou\n");
                                          return PARTICIPOU; }
```

No mesmo contexto, captura-se agora um ID de uma connection, seja a da esquerda ou a da direita. Num caso concreto, em "antonio fez hospital", "antonio"e "hospital"são dois object_id.

```
<connection>{id}            { printf("flex: Link ID string: %s\n", yytext);
                              yylval.str = strdup(yytext);
                              return OBJECT_ID; }
```

Ainda na mesma situação, é necessário apanhar os whitespaces no fim de um "antonio participou baile"e voltar ao contexto inicial.

```
<connection>[ ]*{newline} { printf("flex: End of connection\n");
                            BEGIN INITIAL; }
```

Quase a terminar, a próxima expressão regular irá capturar os new lines, para que possam existir no input.

```
{newline}                   { printf("flex: Eating newline.\n");
                              yylineno++; }
```

Por fim, é retornado o token "ERRO"por qualquer caractere que seja apanhado na próxima expressão regular.

```
.                           { printf("flex: Unexpected character (return ERR).\n");
                              return ERR; }
```

3.4 Estruturas de dados

No yacc (ficheiro "emigrantes.y"), usaram-se os arrays dinâmicos da *Glib* por uma questão de simplicidade. Desta forma, os dois arrays usados pelo grupo foram:

- node_data = g_array_new(FALSE, TRUE, sizeof(char*));
- edge_data = g_array_new(FALSE, TRUE, sizeof(char*)).

Começando por explicar os argumentos de g_array_new, tem-se que:

- FALSE - primeiro argumento, que indica que não se pretende que o array tenha um campo a 0 no fim;
- TRUE - segundo argumento, sinalizador de que se quer inicializar o array a zeros;
- sizeof(char*) - tamanho de cada elemento, em bytes.

Desta forma, o array node_data vai receber todos os tokens que forem capturados relacionados a nodos, pela ordem que aparecem. Já o array edge_data recebe todos os tokens que forem apanhados relacionados a arestas, também pela ordem em que aparecem. No fim, os arrays têm o formato:

```
node_data:
  1: nome
  2: Joao Oliveira
  3: idade
  4: 300
  5: emigrante
  6: joao

edge_data:
  1: antonio
  2: hospital
  3: fez
```

A ordem dos tokens no array edge_data é a ideal para imprimir a linha do dot que faz uma aresta.

```
joao -> hospitalassis[label="fez"]
```

No entanto, a ordem dos argumentos do array node_data parece bizarra, porque se apanha, em primeiro lugar, os campos e só depois o tipo do objeto e o seu ID. Por esta razão, ao percorrer este array será necessário, inicialmente, avançar até ser encontrado "emigrante", "obra"ou "evento"e com o auxílio de um índice secundário percorre-se tudo o que se ignorou neste processo. Ao percorrer o array com este índice secundário, vai-se escrevendo a linha do dot que cria um nó.

```
escolapotassio [label="{Escola Cloreto de Potassio |
      Data: 1943-04-20 | Local: Para}"];
```

Chama-se à atenção que também foi necessário código extra para inserir o caractere | entre os campos, e também para colocar o URL nos objetos que têm esse campo como último.

3.5 Como executar

Para executar o programa, o grupo criou a seguinte makefile, sendo assim necessário apenas digitar "make"na linha de comandos.

```
LIBS=‘pkg-config --cflags --libs glib-2.0‘

emigrantes: y.tab.o lex.yy.o
	gcc -D_XOPEN_SOURCE=700 -std=c99 -o emigrantes y.tab.o
	    lex.yy.o -ll -lm -lfl $(LIBS)

y.tab.o: y.tab.c
	gcc -D_XOPEN_SOURCE=700 -std=c99 -c y.tab.c $(LIBS)

lex.yy.o: lex.yy.c
	gcc -D_XOPEN_SOURCE=700 -std=c99 -c lex.yy.c $(LIBS)

y.tab.c y.tab.h: emigrantes.y
	yacc -d emigrantes.y

lex.yy.c: emigrantes.l y.tab.h
	flex emigrantes.l
```

4 Testes e Resultados

Apresentam-se de seguida dois testes para o programa elaborado, juntamente com o desenho do grafo já processado a partir de uma ferramenta de Graphviz online[2].

4.1 Caso 1

4.1.1 Input

```
emigrante joao:
  nome: "Antonio Joao Joaquim"
  nascimento: "1890-11-24"
  morte: "1972-03-14"
  natural: "Portugal"
  partida: "1904-09-13"
  destino: "Brasil"
  url: "https://www.google.pt/search?q=antonio+joao+joaquim&source=
        lnms&tbm=isch&sa=X&ved=0ahUKEwiMn8Sb6cPbAhUIuhQKHazRDJMQ_
        AUICigB&biw=1920&bih=974#imgrc=6TU1D4xJKAN5cM:"

obra hospitalassis:
  nome: "Hospital Manuel de Assis"
  data: "1940-02-01"
  local: "Sao Paulo"

obra escolapotassio:
  nome: "Escola Cloreto de Potassio"
  data: "1943-04-20"
  local: "Para"

obra capelamoniz:
  nome: "Capela Martim Manhas"
  data: "1964-06-07"
  local: "Belo Horizonte"

evento bailereal47:
  nome: "Baile Real da Realeza do Brasil numero 47"
  data: "1952-03-29"
  local: "Recife"

evento inauguracaocapelamoniz:
  nome: "Inauguracao da Capela Martim Manhas"
  data: "1964-06-09"
  local: "Belo Horizonte"

emigrante joana:
  nome: "Joana Maria"
  nascimento: "1892-10-25"
  morte: "1972-03-15"
  natural: "Portugal"
  partida: "1904-10-19"
  destino: "Brasil"
  url: "https://www.google.pt/search?q=joana+maria&source=lnms&tbm=
        isch&sa=X&ved=0ahUKEwiMn8Sb6cPbAhUIuhQKHazRDJMQ_AUICigB&biw
```

```
        =1920&bih=974#imgrc=6TU1D4xJKAN5cM:"

obra orfanatoteixeira:
  nome: "Ofanato Mendes Teixeira"
  data: "1947-02-10"
  local: "Belo Horizonte"

joao fez hospitalassis
joao fez escolapotassio
joao fez capelamoniz

joao participou bailereal47
joao participou inauguracaocapelamoniz

joana participou bailereal47
joana fez orfanatoteixeira
```

4.1.2 Output

```
digraph D {
    node [shape=Mrecord fontname="Arial"];
    edge [fontname="Arial"];
    joao [label="{Antonio Joao Joaquim | Nascimento: 1890-11-24 |
                  Morte: 1972-03-14 | Natural: Portugal | Partida:
                  1904-09-13 | Destino: Brasil}", URL="http://google.com"];
    hospitalassis [label="{Hospital Manuel de Assis | Data: 1940-02-01 |
                           Local: Sao Paulo}"];
    escolapotassio [label="{Escola Cloreto de Potassio | Data: 1943-04-20 |
                            Local: Para}"];
    capelamoniz [label="{Capela Martim Manhas | Data: 1964-06-07 | Local:
                         Belo Horizonte}"];
    baile real47 [label="{Baile Real da Realeza do Brasil Numero 47 |
                          Data: 1952-03-29 | Local: Recife}"];
    inauguracaocapelamoniz [label="{Inauguracao da Capela Martim Manhas |
                            Data: 1964-06-09 | Local: Belo Horizonte}"];

    joao -> hospitalassis[label="fez"]
    joao -> escolapotassio[label="fez"]
    joao -> capelamoniz[label="fez"]
    joao -> baile real47[label="participou"]
    joao -> inauguracaocapelamoniz[label="participou"]

    joao2 [label="{Joana Joaquim | Nascimento: 1890-11-23 | Morte: 1972-03-15 |
                   Natural: Portugal | Partida: 1904-09-13 | Destino: Brasil}",
                   URL="http://google.com"];
    escolapotassio2 [label="{Escola Cloreto de Potassio | Data: 1943-04-20 |
                             Local: Para}"];
    capelamoniz2 [label="{Capela Martim Manhas | Data: 1964-06-07 |
                          Local: Belo Horizonte}"];
    baile real472 [label="{Baile Real da Realeza do Brasil Numero 47 |
                           Data: 1952-03-29 | Local: Recife}"];
    inauguracaocapelamoniz2 [label="{Inauguracao da Capela Martim Manhas |
                                     Data: 1964-06-09 | Local: Belo Horizonte}"];

    joao2 -> hospitalassis[label="fez"]
    joao2 -> escolapotassio2[label="fez"]
    joao2 -> capelamoniz2[label="fez"]
    joao2 -> baile real472[label="participou"]
    joao2 -> inauguracaocapelamoniz2[label="participou"]
}
```

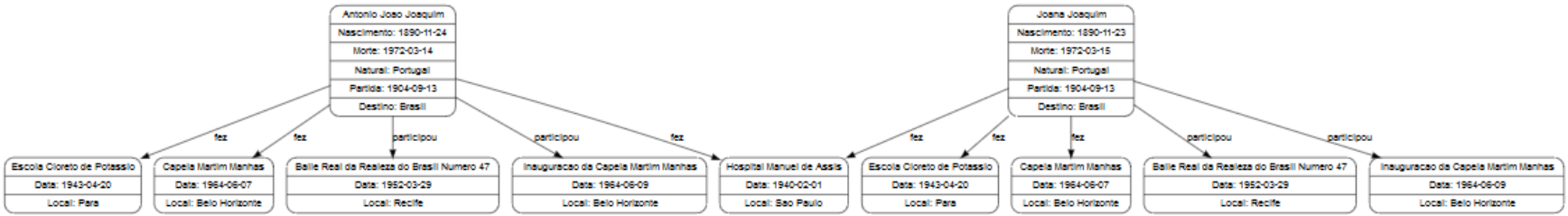

Figura 2: Desenho do grafo para o caso 1

4.2 Caso 2

4.2.1 Input

```
emigrante alberto:
  nome: "Alberto Faria"
  nascimento: "1920-11-22"
  morte: "2002-05-24"
  natural: "Portugal"
  partida: "1950-01-13"
  destino: "Franca"
  url: "https://www.google.pt/search?q=alberto+faria&source=
        lnms&tbm=isch&sa=X&ved=0ahUKEwiMn8Sb6cPbAhUIuhQKHaz
        RDJMQ_AUICigB&biw=1920&bih=974#imgrc=6TU1D4xJKAN5cM:"

obra hospitalgomes:
  nome: "Hospital Gomes de Sa"
  data: "1974-02-01"
  local: "Paris"

obra restaurantebras:
  nome: "Restaurante Bras"
  data: "1989-04-20"
  local: "Marseille"

obra capelaforte:
  nome: "Capela Fortalecimento"
  data: "1997-06-07"
  local: "Aix En Provence"

evento inauguracaocapelaforte:
  nome: "Inauguracao da Capela Fortalecimento"
  data: "1952-03-29"
  local: "Aix En Provence"

alberto fez hospitalgomes
alberto fez restaurantebras
alberto fez capelaforte

alberto participou inauguracaocapelaforte
```

4.2.2 Output

```
digraph D {
  node [shape=Mrecord fontname="Arial"];
  edge [fontname="Arial"];
  alberto [label="{Nome: Alberto Faria | Nascimento: 1920-11-22 |
                   Morte: 2002-05-24 | Natural: Portugal |
                   Partida: 1950-01-13 | Destino: Franca}",
                   URL="https://www.google.pt/search?q=alberto+faria
                   &source=lnms&tbm=isch&sa=X&ved=0ahUKEwiMn8Sb6cPb
                   AhUIuhQKHazRDJMQ_AUICigB&biw=1920&bih=974#imgrc=
                   6TU1D4xJKAN5cM:"];
  hospitalgomes [label="{Nome: Hospital Gomes de Sa | Data: 1974-02-01
                         | Local: Paris}"];
  restaurantebras [label="{Nome: Restaurante Bras | Data: 1989-04-20 |
                           Local: Marseille}"];
  capelaforte [label="{Nome: Capela Fortalecimento | Data: 1997-06-07 |
                       Local: Aix En Provence}"];
  inauguracaocapelaforte [label="{Nome: Inauguracao da Capela
                                  Fortalecimento | Data: 1952-03-29 |
                                  Local: Aix En Provence}"];
  alberto -> hospitalgomes[label="fez"]
  alberto -> restaurantebras[label="fez"]
  alberto -> capelaforte[label="fez"]
  alberto -> inauguracaocapelaforte[label="participou"]
}
```

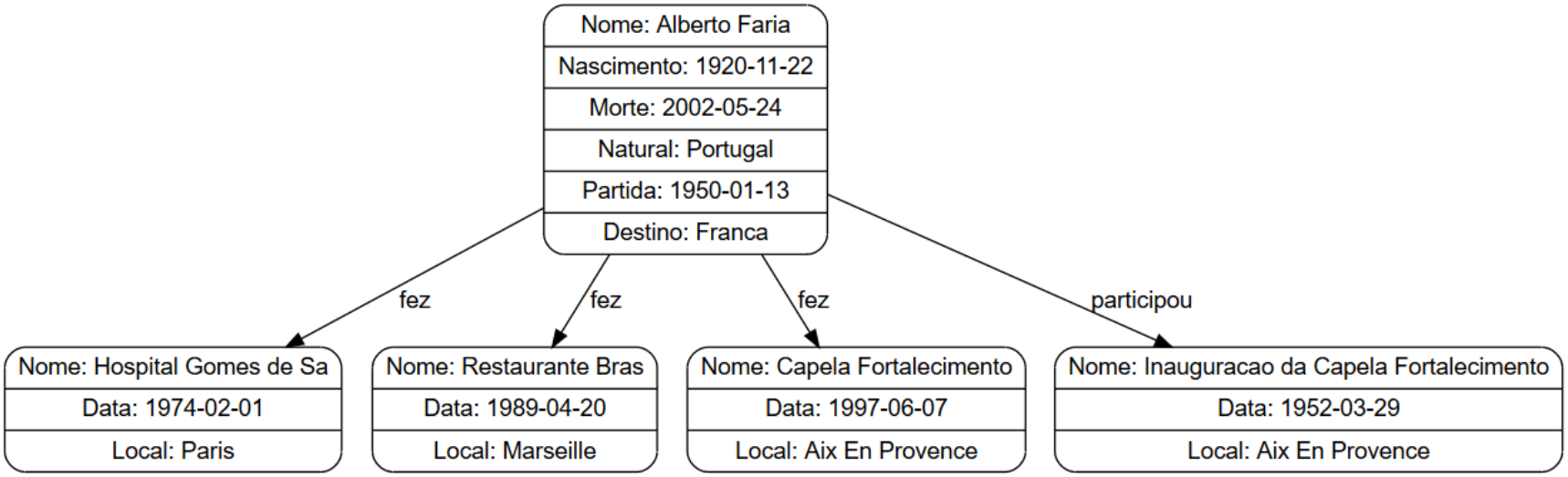

Figura 3: Desenho do grafo para o caso 2

5 Conclusões e Sugestões

A resolução deste último exercício prático foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados nas aulas práticas da Unidade Curricular de Processamento de Linguagens. Deste modo, foram adquiridos conhecimentos mais sólidos acerca de expressões regulares e ações semânticas, e da utilidade o par *lex/yacc* pode ter num contexto real. De facto, foi possível perceber que a exigência de programar algo equivalente ao desenvolvido em Java (linguagens de maior domínio do grupo), por exemplo, seria muito maior. Assim, o grupo ficou satisfeito por ter desenvolvido este trabalho, de complexidade bastante superior aos anteriores exercícios da unidade curricular, em GAWK e FLEX.

No que toca ao enunciado 1, em geral, uma das dificuldades residiu, em primeira instância, no desenvolvimento/escolha acertada da gramática e expressões regulares, de modo a prever todos os casos de exceção, tornando os programas mais robustos. Ainda assim, o grupo considera que o maior obstáculo, e fase mais dispendiosa de tempo, consistiu na elaboração de algoritmos que processassem ou transformassem os *datasets* nos resultados desejados.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz.

6 Anexos

6.1 Flex

```
%{
#include <stdio.h>
#include "y.tab.h"
%}

id        [a-zA-Z0-9]+
number    [0-9]+
string    [^"\r\n]+
wspace    [ \r\n]*
newline   (\r\n|\n)

%x object_id field field_value connection

%%
(emigrante|obra|evento) { printf("flex: Object type: %s\n", yytext);
                          BEGIN object_id;
                          yylval.str = strdup(yytext);
                          return OBJECT_TYPE; }
{id}                    { printf("flex: Link ID: %s\n", yytext);
                          BEGIN connection;
                          yylval.str = strdup(yytext);
                          return OBJECT_ID; }
<object_id>[ ]          { /* Eating the space between object type
                             and its ID */ }
<object_id>{id}         { printf("flex: Object ID: %s\n", yytext);
                          yylval.str = strdup(yytext);
                          return OBJECT_ID; }
<object_id>:{wspace}    { BEGIN field; }
<field>{id}             { printf("flex: Field ID: %s\n", yytext);
                          yylval.str = strdup(yytext);
                          return FIELD_ID; }
<field>\:{wspace}\"     { BEGIN field_value; }
<field_value>{string}   { printf("flex: String: %s\n", yytext);
                          yylval.str = strdup(yytext);
                          return STRING; }
<field_value>\"[ ]*{newline}{2,}{printf("flex: Two newlines
                                after field value\n");
                                BEGIN INITIAL; }
<field_value>\"{wspace}              {printf("flex: End of
                                      string (not two newlines)\n");
                                      BEGIN field; }
<connection>{wspace}fez{wspace}      {printf("flex: fez\n");
                                      return FEZ; }
<connection>{wspace}participou{wspace}{printf("flex: participou\n");
                                      return PARTICIPOU; }
<connection>{id}                     {printf("flex: Link ID string:
                                      %s\n", yytext);
                                      yylval.str = strdup(yytext);
                                      return OBJECT_ID; }
<connection>[ ]*{newline}            {printf("flex: End of connection\n");
```

```
                                        BEGIN INITIAL; }
{newline}                               {printf("flex: Eating newline.\n");
                                        yylineno++; }
.                                       {printf("flex: Unexpected character
                                          (return ERR).\n");
                                        return ERR; }

%%
```

6.2 YACC

```
%{
#include <stdio.h>
#include <glib.h>
#include <string.h>
#include <ctype.h>
#include "y.tab.h"

extern int yylineno;
extern char* yytext;
extern int yylex();
int yyerror(char*);

/* node_data vai guardar todos os tokens relacionados com
   nodos do grafo, na ordem em que foram recebidos.
   edge_data vai guardar todos os tokens relacionados com
   ligacoes do grafo. O trabalho de interpretar os conteudos
   destes arrays (tendo em conta a ordem dos dados) e do for
   loop que itera sobre o array.
*/


/* Formato: ["nome", "joao alberto", "idade", "895", [...],
            "emigrante", "joao", [...]] */
GArray* node_data;
/* Formato: ["joao", "capela", "fez", "antonio", "baile", "participou"] */
GArray* edge_data;
%}

%token OBJECT_TYPE STRING OBJECT_ID FIELD_ID FEZ PARTICIPOU ERR

%union{
  char* str;
}

%type <str> STRING OBJECT_TYPE OBJECT_ID FIELD_ID

%start OBJECTS

%%

OBJECTS : OBJECTS OBJECT
        | OBJECTS CONNECTION
        |
        ;

OBJECT : OBJECT_TYPE OBJECT_ID FIELDS { char* one = strdup($1);
                                        char* two = strdup($2);
                                        g_array_append_val(node_data, one);
                                        g_array_append_val(node_data, two); }
       ;
```

```
FIELDS : FIELDS FIELD
       | FIELD
       ;

FIELD : FIELD_ID STRING { char* one = strdup($1);
                          char* two = strdup($2);
                          g_array_append_val(node_data, one);
                          g_array_append_val(node_data, two); }
      ;

CONNECTION : OBJECT_ID FEZ OBJECT_ID { char* one = strdup($1);
                                       char* three = strdup($3);
                                       char* f = "fez";
                                       g_array_append_val(edge_data, one);
                                       g_array_append_val(edge_data, three);
                                       g_array_append_val(edge_data, f); }
           | OBJECT_ID PARTICIPOU OBJECT_ID{char* one = strdup($1);
                                            char* three = strdup($3);
                                            char* p = "participou";
                                          g_array_append_val(edge_data,one);
                                          g_array_append_val(edge_data,three);
                                          g_array_append_val(edge_data,p);}
           ;
%%

int main() {
  node_data = g_array_new( FALSE, TRUE, sizeof(char*));
  edge_data = g_array_new( FALSE, TRUE, sizeof(char*));
  yyparse();

  printf("\n\n\n\n\n==================␣DOT␣OUTPUT␣==================\n\n");

  // Graph header
  printf("digraph␣D␣{\n␣␣␣node␣[shape=Mrecord␣fontname=\"Arial\"];\n
␣␣␣␣edge␣[fontname=\"Arial\"];\n");

  // Print every node
  // (unsigned int because node_data->len is a guint)
  unsigned int lastUsed = 0;
  for (unsigned int i = 0; i < node_data->len; i++) {

    // Look for emigrante, obra, or evento
    if (strcmp(g_array_index(node_data, char*, i), "emigrante") == 0 ||
        strcmp(g_array_index(node_data, char*, i), "obra") == 0 ||
        strcmp(g_array_index(node_data, char*, i), "evento") == 0
    ) {
      // Found!
      // Now pick back up on the "lastUsed" index and create the node

      // Start node
      printf("%s␣[label=\"{", g_array_index(node_data, char*, i+1));
      char* label;
      char* string;
      int startedWriting = 0;
```

```
      for (; lastUsed < i; lastUsed++) {

        label = g_array_index(node_data, char*, lastUsed);
        label[0] = toupper(label[0]); // Uppercase first char of label

        lastUsed++; // Go to next node_data token

        string = g_array_index(node_data, char*, lastUsed);

        if (strcmp(label, "Url") == 0) {
          break;
        }

        if (startedWriting) {
          printf(" | ");
        } else {
          startedWriting = 1;
        }

        printf("%s: %s", label, string);
      }

      // Finished wrting node
      // Print URL if we stopped on that field,
      // or just close the node if no URL found at the end
      if (strcmp(label, "Url") == 0) {
        printf("}\", URL=\"%s\"];\n", string);
        lastUsed += 3;//Move lastUsed 3steps forward because we stopped at url
      } else {
        printf("}\"];\n");
        lastUsed += 2; // Move lastUsed 2 steps forward
      }
    }
  }

  // Print every edge
  for (unsigned int j = 0; j < edge_data->len; j++) {
    char* doer = g_array_index(edge_data, char*, j);
    j++;
    char* done = g_array_index(edge_data, char*, j);
    j++;
    char* action = g_array_index(edge_data, char*, j);

    printf("%s -> %s[label=\"%s\"]\n", doer, done, action);
  }

  // Close graph
  printf("}\n");

  return 0;
}

int yyerror(char* err) {
  fprintf(stderr,"Error: %s\nyytext: %s\nyylineno: %d\n",err,yytext,yylineno);
```

```
  return 0;
}
```

Referências

[1] *Museu das Migrações e das Comunidades*. URL: www.museu-emigrantes.org. (accessed: 10.06.2018).

[2] *WebGraphviz is Graphviz in the Browser*. URL: http://www.webgraphviz.com/. (accessed: 09.06.2018).