Processamento de Linguagens (3º ano de Curso)

Trabalho Prático N 2

Relatório de Desenvolvimento

Filipe Costa Oliveira
a57816

29 de Maio de 2016

Conteúdo

Capítulo 1

Introdução

O presente trabalho prático foca-se no desenvolvimento de um compilador, que tem como fonte uma linguagem de alto nível (também esta desenvolvida especificamente para este trabalho prático) , gerando código para uma máquina de stack virtual.
Um compilador comum divide o processo de tradução em várias fases. Para o propósito específico desta unidade curricular iremos focar-nos nas seguintes:

- 1ª Fase de tradução – Análise Léxica, que agrupa sequências de caracteres em tokens. Recorreremos nesta fase à definição das expressões regulares que permitem definir os tokens.
- 2ª Fase de tradução – Reconhecimento(Parsing) da estrutura gramatical do programa, através do agrupamento dos tokens em produções. Recorreremos à definição de uma gramática independente de contexto por forma a definir as estruturas de programa válidas a reconhecer pelo parser. Denote que juntamente com o parsing é realizada a análise semântica, assim como a geração de código associando regras às produções anteriormente descritas.

Começaremos portanto por definir uma linguagem de programação imperativa simples, que chamaremos Algebra. A Algebra permitirá:

- declarar e manusear variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação. Aos arrays de duas dimensões, por se tratar de uma linguagem algébrica, chamaremos matrizes, dada a fácil associação a este tipo de variável à sua definição análoga da álgebra linear.
- efetuar instruções algorítmicas básicas como a atribuição de expressões a variáveis.
- ler do standard input e escrever no standard output.
- efetuar instruções para controlo do fluxo de execução – condicional e cíclica – que possam ser aninhadas.
- definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado atómico.

Na nossa linguagem de programação por questões de estruturação e percepção, teremos como premissa que as varíaveis deverão ser declaradas no início do programa, não podendo haver re-declarações, nem utilizações sem declaração prévia. Não será permitida a declaração e associação de um valor inteiro na mesma instrução. Achamos essa solução pouco elegante. Assim, todas as variáveis terão o valor zero após a declaração.
Será desenvolvido portanto o compilador para a Algebra, com base na GIC criada acima e recurso ao Gerador Yacc/ Flex. O compilador de Algebra irá gerar pseudo-código, Assembly da Máquina Virtual VM cuja documentação completa está disponibilizada em anexo.

Por forma a facilitar e validar o trabalho, à medida que as funcionalidades forem descritas serão apensados exemplos ilustrativos.
Por fim, serão apresentados um conjunto de testes mais complexos(programas-fonte diversos e respectivo código produzido), que tentam testar de uma forma mais alargadas as funcionalidades da Algebra, sendo estes:

- lidos 3 números, escrever o maior deles.
- ler N (valor dado) números e calcular e imprimir o seu somatório.
- contar e imprimir os números pares de uma sequência de N números dados.
- ler e armazenar os elementos de um vetor de comprimento N, imprimido os valores por ordem crescente após fazer a ordenação do array por trocas diretas.
- ler e armazenar os elementos de uma matriz NxM, calculando e imprimindo de seguida a média e máximo dessa matriz.
- invocar e usar num programa uma função.

Capítulo 2

Filtro de Texto com o Flex para ler uma ontologia descrita em OWL

2.1 Concepção/desenho da Resolução

Tal como descrito anteriormente o problema será resolvido com recurso à ferramenta Flex. Pretendemos então analisar uma ontologia descrita em OWL e desenhar um grafo que ligue os conceitos entre si. Denote que a ontologia descrita em OWL não passa de um dialecto XML, sendo portanto standardizada a forma como extraímos os dados.

Apesar da linguagem OWL ser extremamente extensa, apenas nos interessam a identificação das classes propriamente ditas e as relações entre estas, por forma a podermos desenhar o grafo que descreve a ontologia.

2.2 Padrões de frases a encontrar, através de ER

Podemos considerar que no contexto do problema apresentado temos especial interesse nos axiomas de propriedades de dados, objectos, e classes. Assim, os axiomas passíveis de serem incluídos no grafo serão:

- **Axiomas de Objetos**

```
ObjectPropertyAxiom :=
SubObjectPropertyOf | EquivalentObjectProperties |
DisjointObjectProperties | InverseObjectProperties |
ObjectPropertyDomain | ObjectPropertyRange |
FunctionalObjectProperty | InverseFunctionalObjectProperty |
ReflexiveObjectProperty | IrreflexiveObjectProperty |
SymmetricObjectProperty | AsymmetricObjectProperty |
TransitiveObjectProperty
```

- **Axiomas de Dados**

```
DataPropertyAxiom :=
SubDataPropertyOf | EquivalentDataProperties | DisjointDataProperties |
DataPropertyDomain | DataPropertyRange | FunctionalDataProperty
```

- **Axiomas de Classe**

```
ClassAxiom :=
SubClassOf | EquivalentClasses |
DisjointClasses | DisjointUnion
```

que serão convertidos nas seguintes definições de expressões regulares a serem utilizadas no Flex:

```
OBJ_PROP_AXIOM SubObjectPropertyOf|EquivalentObjectProperties|DisjointObjectProperties|
    InverseObjectProperties|ObjectPropertyDomain|ObjectPropertyRange|FunctionalObjectProper

DAT_PROP_AXIOM SubDataPropertyOf|EquivalentDataProperties|DisjointDataProperties|
    DataPropertyDomain|DataPropertyRange|FunctionalDataProperty

CLA_PROP_AXIOM SubClassOf|EquivalentClasses|DisjointClasses|DisjointUnion
```

2.2.1 Comutação de contextos em Flex através de start conditions

Por forma a realizar a correta análise dos padrões de texto foi necessário adicionar start conditions às regras das expressões regulares. Deste modo, apresenta-se a lista de todas as start conditions utilizadas e sua breve descrição:

```

%x IN_PROP IN_DATA RELATION RELATION_BEG_END DATA DATA_BEG_END VALUE IN_CLASS CLASS_BEG_END
```

- IN_PROP – define o contexto de leitura de Axiomas de Objetos
- RELATION – define o contexto de inicio da leitura de Relação de Objetos
- RELATION_BEG_END – define o contexto de inicio da leitura do primeiro e segundo objectos na Relação de Objetos

- IN_DATA – define o contexto de leitura de Axiomas de Dados
- DATA – define o contexto do início da leitura dos Dados
- DATA_BEG_END – define o contexto de inicio da leitura da classe envolvida no Axioma de Dados
- VALUE – define o contexto de inicio da leitura do datatype envolvido no Axioma de Dados

- IN_CLASS – define o contexto de leitura de Axiomas de Classes
- CLASS_BEG_END – define o contexto de inicio da leitura da primeira e segunda classes na Relação de Classes

2.2.2 Expressões Regulares e acções resultantes

Definidas as start conditions, resta-nos explicitar todas as expressões regulares, incluindo aquelas que dão início à comutação entre contextos do flex.

```
[^<> ]*<{DAT_PROP_AXIOM}> { BEGIN IN_DATA; }
[^<> ]*<{OBJ_PROP_AXIOM}> { BEGIN IN_PROP; }
[^<> ]*<{CLA_PROP_AXIOM}> { BEGIN IN_CLASS; }

<IN_CLASS>[^<>]*<Class[ \t\n]*IRI=\" {BEGIN CLASS_BEG_END;}
<IN_CLASS><\/{CLA_PROP_AXIOM} {BEGIN INITIAL;}
<IN_CLASS>.|\n {;}

<CLASS_BEG_END>[^">]+/\" {
if ( class_subclass == NULL ){
  class_subclass = strdup(yytext);
```

```
12 }
13 else{
14   class_class = strdup(yytext);
15   print_class();
16 }
17 }
18
19 <CLASS_BEG_END>\/> { BEGIN IN_CLASS;  }
20 <CLASS_BEG_END>.|\n {;}
21
22 <DATA>[^">]+/\" {data=strdup(yytext);}
23 <DATA>\/> {BEGIN IN_DATA;}
24 <DATA>.|\n {;}
25
26 <IN_DATA>[^<>]*<DataProperty[ \t\n]*IRI=\" { BEGIN DATA;}
27 <IN_DATA>[^<>]*<Class[ \t\n]*IRI=\" { BEGIN DATA_BEG_END;}
28 <IN_DATA>[^<>]*<Datatype[ \t\n]*abbreviatedIRI=\" {  BEGIN VALUE;}
29 <IN_DATA><\/{DAT_PROP_AXIOM} {BEGIN INITIAL;}
30 <IN_DATA>.|\n {;}
31
32 <VALUE>[^"<>]+/\" { value = strdup(yytext); print_data(); }
33 <VALUE>\/> {BEGIN IN_DATA;}
34 <VALUE>.|\n {;}
35
36 <DATA_BEG_END>[^">]+/\" { class = strdup(yytext); }
37 <DATA_BEG_END>\/> {BEGIN IN_DATA;}
38 <DATA_BEG_END>.|\n {;}
39
40 <IN_PROP>[^<>]*<ObjectProperty[ \t\n]*IRI=\" { BEGIN RELATION;  }
41 <IN_PROP>[^<>]*<Class[ \t\n]*IRI=\" {  BEGIN RELATION_BEG_END;}
42 <IN_PROP><\/{OBJ_PROP_AXIOM} { BEGIN INITIAL;}
43 <IN_PROP>.|\n {;}
44
45 <RELATION>[^"<>]+/\" { relation=strdup(yytext);}
46 <RELATION>\/> { BEGIN IN_PROP;}
47 <RELATION>.|\n {;}
48
49 <RELATION_BEG_END>[^"<>]+/\" {
50 if ( begin == NULL ){
51   begin = strdup(yytext);
52 }
53 else{
54   end = strdup(yytext);
55   print_prop();
56 }
57 }
58
59 <RELATION_BEG_END>\/> { BEGIN IN_PROP;}
60 <RELATION_BEG_END>.|\n { ;}
61
62 <INITIAL>.|\n {;}
```

2.3 Codificação e Testes

2.3.1 Estruturas de Dados e bibliotecas utilizadas

Como pode verificar, não existem estruturas de dados complexas para a resolução deste problema, sendo que são apenas utilizadas strings para guardar temporariamente os valores dos dados em relação. De seguida explicitam-se todas as variáveis e bilbliotecas utilizadas:

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

  char* relation;
  char* begin;
  char* end;
  char* data;
  char* class;
  char* value;
  char* class_class;
  char* class_subclass;

  void print_prop();
  void print_data();
  void print_class();
  %}
```

2.3.2 Métodos Adicionais

Como poderá constatar na seção **??** existem 3 métodos adicionais invocados no final da leitura completa dos axiomas de objectos, classes e dados, sendo eles de seguida explicitados:

```
  void print_class(){
    printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n",
        class_subclass);
    printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n",
        class_class);
    printf("\"%s\" -> \"%s\" [ label = \"SubClassOf\" , fontsize=8 , fontcolor=\"blue\",
        color=\"blue\" ]\n", class_subclass, class_class );
    class_subclass=NULL;
    class_class=NULL;
  }

void print_data(){
  printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n", class);
  printf("\"%s\" [shape = box, style=filled ,color=\"red\", fontsize=12 fontname=helvetica];\
      n", value);
  printf("\"%s\" -> \"%s\" [ label = \"%s\" ]\n", class, value, data );
  class=NULL;
  value=NULL;
  data=NULL;
}

void print_prop(){
  printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n", begin);
  printf("\"%s\" -> \"%s\" [ label = \"%s\" ]\n", begin, end, relation );
  begin=NULL;
  relation=NULL;
```

```
23   end=NULL;
24 }
```

Os métodos por si só são elucidativos, sendo que a impressão é realizada conforme a linguagem Dotty.

2.3.3 Main

Dada a simplicidade das estruturas de dados, o método main é também bastante simples, de seguida explicitado:

```
1 int main(int argc, char** argv){
2   graph_print();
3   yylex();
4   printf("}\n");
5   return (0);
6 }
```

O código completo da ferramenta é passível de consulta na seção **??** na página **??**.

2.4 Testes realizados e Resultados

Foi realizado um script da shell com vista a facilmente ilustrar as funcionalidades da ferramenta, sendo que essas mesma script lê o ficheiro exemplo test1.xml , criando o ficheiro ex1.gv e, a partir deste o ficheiro ex1.png, demonstrado na figura **??**.

2.4.1 Shell Script

```
1 #!bin/sh
2
3 echo "/*
4 ******************************************************************************
5 *Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
6 *   All Rights Reserved.
7 *
8 ******************************************************************************
9 *   Content : simple tool 1a) functionality script
10 *
11 *
12 ******************************************************************************/"
13
14 make clean
15 flex owl_graph.l
16 make
17 ./owl_graph < test1.xml > ex1.gv
18 dot ex1.gv -Tpng > ex1.png
19 echo "############################"
20 echo ">>>>>>>>> ex1 in ex1.png"
21 echo "          opening file"
22 echo "############################"
23 open ex1.png
24 echo "done"
```

2.4.2 Ficheiro Exemplo de input no formato OWL

```
<ObjectPropertyDomain>
<ObjectProperty IRI="#receives"/>
<Class IRI="#Laundry"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
<Annotation>
<AnnotationProperty abbreviatedIRI="owl:backwardCompatibleWith"/>
<IRI>#Laundry</IRI>
</Annotation>
<ObjectProperty IRI="#receives"/>
<Class IRI="#Order"/>
</ObjectPropertyRange>

<ObjectPropertyDomain>
<ObjectProperty IRI="#works"/>
<Class IRI="#Worker"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
<Annotation>
<AnnotationProperty abbreviatedIRI="owl:backwardCompatibleWith"/>
<IRI>#Worker</IRI>
</Annotation>
<ObjectProperty IRI="#works"/>
<Class IRI="#Laundry"/>
</ObjectPropertyRange>


<ObjectPropertyDomain>
<ObjectProperty IRI="#owns"/>
<Class IRI="#Client"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
<Annotation>
<AnnotationProperty abbreviatedIRI="owl:backwardCompatibleWith"/>
<IRI>#Client</IRI>
</Annotation>
<ObjectProperty IRI="#owns"/>
<Class IRI="#Order"/>
</ObjectPropertyRange>

<SubClassOf>
<Class IRI="#Laundry"/>
<Class IRI="#Stores"/>
</SubClassOf>
<SubClassOf>
<Class IRI="#Owner"/>
<Class IRI="#Person"/>
</SubClassOf>

<DataPropertyDomain>
<DataProperty IRI="#material"/>
<Class IRI="#Type"/>
</DataPropertyDomain>
<DataPropertyRange>
<DataProperty IRI="#material"/>
<Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>

```

```
59 <DataPropertyDomain>
60 <DataProperty IRI="#orderid"/>
61 <Class IRI="#Order"/>
62 </DataPropertyDomain>
63 <DataPropertyRange>
64 <DataProperty IRI="#orderid"/>
65 <Datatype abbreviatedIRI="xsd:string"/>
66 </DataPropertyRange>
67 <SubClassOf>
68 <Class IRI="#Client"/>
69 <Class IRI="#Person"/>
70 </SubClassOf>
71 <SubClassOf>
72 <Class IRI="#Worker"/>
73 <Class IRI="#Person"/>
74 </SubClassOf>
```

2.4.3 Grafo exemplo resultante

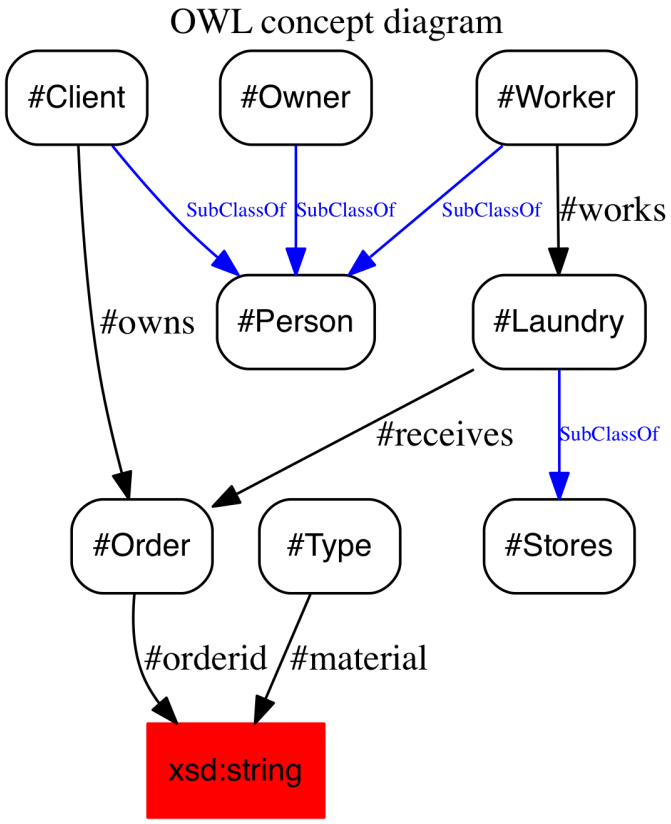

Figura 2.1: Grafo exemplo criado com as relações entre classes a partir da análise do ficheiro test1.xml

Capítulo 3

Normalizador de ficheiros BibTeX

3.1 Concepção/desenho da Resolução

Tal como descrito anteriormente o problema será resolvido com recurso à ferramenta Flex. Pretendemos então analisar documentos BibTeX e:

- fazer a contagem das categorias(phDThesis, Misc, InProceeding, etc.) que ocorrem no documento, produzindo um documento em formato HTML com o nome das categorias encontradas e respectivas contagens;
- Desenvolva uma ferramenta de normalização (sempre que um campo está entre aspas, trocar para chavetas e escrever o nome dos autores no formato "N. Apelido"), fazendo ainda uma ferramenta de pretty-printing que indente corretamente cada campo, escreva um autor por linha e colocando sempre no início os campos autor e título.
- Construir um Grafo que mostre, para um dado autor (escolhido pelo utilizador) todos os autores que publicam normalmente com o autor em causa, recorrendo uma vez mais à linguagem Dot do GraphViz2.

Para foram produzidos 3 ficheiros (bib_norm_1.l, bib_norm_2.l, bib_norm_3.l), um para cada alínea por forma a tornar de mais fácil compreensão o papel de cada expressão regular, start condition, ou estrutura de dados na solução geral. Facilmente se combinam os 3 ficheiros em 1, mas dado o propósito académico da ferramenta julgo ser fulcral a compreensão do anteriormente descrito em cada ficheiro e alínea do problema.

3.2 alínea a) ficheiro bib_norm_1.l

3.2.1 Padrões de frases a encontrar, através de ER

Podemos considerar que no contexto do problema apresentado temos interesse apenas na contagem das categorias, que serão facilmente encontradas com base nas seguintes definições de expressões regulares a serem utilizadas no Flex:

```
1 LETRA  [A–Za–z]
2 CATEG  \@{LETRA}+\{
3 LETRA_NUM  [0–9A–Za–z]
4 ID  ({LETRA_NUM}|:)+
```

Expressões Regulares e acções resultantes

Resta-nos explicitar todas as expressões regulares e respectivas acções resultantes:

```
1 %%
2 {CATEG}/[^=]*,          {
```

```
3   yytext++; yytext[yyleng-2]='\0';
4   char* key = g_ascii_strdown (yytext, yyleng-2);
5   if ( g_hash_table_contains ( table ,(void*) key ) ){
6     int value;
7     value = GPOINTER_TO_INT( g_hash_table_lookup ( table,(void*) key));
8     value++;
9     g_hash_table_replace (  table, (void*) key,GINT_TO_POINTER(value) );
10  }
11  else {
12    int value = 1;
13    gboolean add_result = g_hash_table_insert (  table, (void*) key,  GINT_TO_POINTER(value)
        );
14  }
15                      }
16 .|\n|\t { ; }
17 %%
```

3.2.2 Codificação

Estruturas de Dados e bibliotecas utilizadas

Como pode verificar foi necessário recorrer a estruturas de dados complexas para a resolução deste problema, nomeadamente hash tables. Ora, segundo o conselho do professor José João, foi reutilizado código da biblioteca da GLib, biblioteca essa extremamente otimizada. De seguida explicitam-se todas as variáveis e bibliotecas utilizadas:

```
1 %{
2   /*
3    ****************************************************************************
4    *   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
5    *   All Rights Reserved.
6    *
7    ****************************************************************************
8    *   Content : Simple bibtex category counter (phDThesis, Misc, InProceeding,
9    *             etc.), that occur in a document
10   ****************************************************************************/
11
12 #include <stdio.h>
13 #include <glib.h>
14 #include <stdlib.h>
15 #include <string.h>
16
17   //HashTable
18   GHashTable *table;
19
20   %}
```

Métodos Adicionais

Como poderá constatar na seção **??** existe 1 método adicional invocado no final da leitura ficheiro, tendo por função imprimir o par (chave – valor) num formato de lista HTML, sendo a chave a categoria BibTeX e o valor o número de ocorrências da mesma no ficheiro:

```
1   static void print_key_value(gpointer key, gpointer value, gpointer userdata){
2     int val = (int) value;
3     char* ke = (char*) key;
4     printf("<li>%s : %d</li>\n", ke, val);
5   }
```

Main

Dada a simplicidade do analisador, o método main é também bastante simples, apenas imprimindo as tags HTML e inicializando a hashtable. Passaremos de seguida a explicitá-lo:

```
1
2 int main(){
3   table = g_hash_table_new(g_str_hash, g_str_equal);
4   yylex();
5   printf("<!DOCTYPE html>\n<html>\n<body>\n<ul>\n");
6   g_hash_table_foreach(table,print_key_value, NULL );
7   printf("</ul>\n</body>\n</html>\n");
8   return (0);
9 }
```

O código completo da ferramenta é passível de consulta na seção **??** na página **??**.

3.3 alínea b) ficheiro bib_norm_2.l

3.3.1 Padrões de frases a encontrar, através de ER

Podemos considerar que no contexto do problema apresentado temos interesse em todos os campos de cada entrada, sendo que teremos de tratar especialmente os campos author e title. Tal tarefa é facilitada com base nas seguintes definições de expressões regulares a serem utilizadas no Flex:

```
1 LETRA [A-Za-z]
2 LETRA_NUM [0-9A-Za-z]
3 NUM [0-9]
4 CATEG \@{LETRA}+\{
5 FIELD_ID ^[^,}"][^=]*=[ \t]*
6 FIELD_BREAK [^=]*[ \t\n]*(\"|\})
7 FIELD_BREAK_NUM [^=\}]*\}?
8 FIELD_START [\{\"]
9 AUTHOR_ID ^[ \t]*[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*
10 TITLE_ID ^[ \t]*[Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*
11 AUTHOR_SEP [ \t]and[ \t]
12 AUTHOR_BREAK [^=]+(\"|\})
```

Comutação de contextos em Flex através de start conditions

Por forma a realizar a correta análise dos padrões de texto foi necessário adicionar start conditions às regras das expressões regulares. Deste modo, apresenta-se a lista de todas as start conditions utilizadas:

```
1 %x INSIDE IN_AUTHOR IN_FIELD_TXT IN_FIELD_NUM AUTHOR_DIV START_AUTHOR
2 \end{itemize}
3
4 \subsubsection{Expressões Regulares e acções resultantes}
5 \label{2ber}
6 Definidas as start conditions, resta-nos explicitar todas as expressões regulares, incluindo
      aquelas que dão início à comutação entre contextos do flex.
7 \begin{lstlisting}
8 %%
9
10 {CATEG}[^=]*,                {
11   printf("%s\n",yytext);
12   BEGIN INSIDE;
13                              }
```

```
14
15 <INSIDE>{TITLE_ID}{FIELD_START}  {
16   yytext[yyleng-2]='\0';
17   title_key = strdup ( yytext );
18   title_key = g_strchomp ( title_key );
19   title_key = g_strchug ( title_key );
20   field_id = strdup(title_key);
21   BEGIN IN_FIELD_TXT;}
22
23   <INSIDE>{AUTHOR_ID}{FIELD_START}  {
24     yytext[yyleng-2]='\0';
25     author_key = strdup ( yytext );
26     author_key = g_strchomp ( author_key );
27     author_key = g_strchug ( author_key );
28     BEGIN START_AUTHOR;
29   }
30
31 <INSIDE>{FIELD_ID}/{LETRA_NUM}  {
32   yytext[yyleng-1]='\0';
33   field_id = strdup ( yytext );
34   field_id = g_strchomp ( field_id );
35   field_id = g_strchug ( field_id );
36   BEGIN IN_FIELD_NUM; }
37
38   <INSIDE>{FIELD_ID}{FIELD_START}  {
39     yytext[yyleng-2]='\0';
40     field_id = strdup ( yytext );
41     field_id = g_strchomp ( field_id );
42     field_id = g_strchug ( field_id );
43     BEGIN IN_FIELD_TXT;
44   }
45
46 <INSIDE>[ \n\t]*, { BEGIN INSIDE; }
47 <INSIDE>[ \n\t]*\} { BEGIN INITIAL; }
48 <INSIDE>.|\n { BEGIN INSIDE;}
49
50 <START_AUTHOR>.           {
51   author_initial = (char*) malloc ( 4* sizeof(char));
52   author_initial[0] = yytext[0];
53   author_initial[1] = '.';
54   author_initial[2]=' ';
55   author_initial[3]='\0';
56   BEGIN IN_AUTHOR;
57 }
58
59 <IN_AUTHOR>[^= \n\t]*/[ ]and[ ]     {
60   author_lastname = strdup ( yytext );
61   author_lastname = g_strchomp ( author_lastname );
62   author_lastname = g_strchug ( author_lastname );
63   char *initial_plus_lastname;
64   int size = strlen(author_lastname);
65   size += strlen( author_initial) + 1;
66   initial_plus_lastname = (char*) malloc ( size * sizeof(char));
67   strcpy(initial_plus_lastname, author_initial);
68   strcat(initial_plus_lastname, author_lastname);
69   g_hash_table_insert (  authors_table, (void*) initial_plus_lastname,   GINT_TO_POINTER (0)
       );
70   BEGIN AUTHOR_DIV;
71 }
```

```
72
73 <IN_AUTHOR>[^= \n\t]*[ \t]?(\"|\})/,     {
74 yytext[yyleng-1]='\0';
75 author_lastname = strdup ( yytext );
76 author_lastname = g_strchomp ( author_lastname );
77 author_lastname = g_strchug ( author_lastname );
78 char *initial_plus_lastname;
79 int size = strlen(author_lastname);
80 size += strlen( author_initial) + 1;
81 initial_plus_lastname = (char*) malloc ( size * sizeof(char));
82 strcpy(initial_plus_lastname, author_initial);
83 strcat(initial_plus_lastname, author_lastname);
84 g_hash_table_insert (  authors_table, (void*) initial_plus_lastname,   GINT_TO_POINTER (0) )
     ;
85 BEGIN AUTHOR_DIV;
86 }
87
88 <IN_AUTHOR>.                { ; }
89
90 <AUTHOR_DIV>,               { BEGIN INSIDE; }
91
92 <AUTHOR_DIV>[ ]and[ ]       { BEGIN START_AUTHOR; }
93
94
95 <IN_FIELD_NUM>{FIELD_BREAK_NUM}[ \n\t\r]*\} {
96   yytext[yyleng-1]='\0';
97   field = strdup(yytext);
98   field = g_strchomp ( field );
99   field = g_strchug ( field );
100  g_hash_table_insert (  num_fields_table, (void*) field_id,  (void*) field );
101  pretty_print();
102  BEGIN INITIAL;
103 }
104
105 <IN_FIELD_NUM>{FIELD_BREAK_NUM}[ \n\t\r]*(#[^,=]*)?, {
106  yytext[yyleng-1]='\0';
107  field = strdup(yytext);
108  field = g_strchomp ( field );
109  field = g_strchug ( field );
110  g_hash_table_insert (  num_fields_table, (void*) field_id,  (void*) field );
111  BEGIN INSIDE;
112 }
113
114 <IN_FIELD_TXT>{FIELD_BREAK}[ \n\t\r]*\} {
115  yytext[yyleng-2]='\0';
116  field = strdup(yytext);
117  field = g_strchomp ( field );
118  field = g_strchug ( field );
119  g_hash_table_insert (  txt_fields_table, (void*) field_id,  (void*) field );
120  BEGIN INITIAL;
121  pretty_print();
122 }
123
124 <IN_FIELD_TXT>{FIELD_BREAK}[ \n\t\r]*(#[^,=]*)?, {
125  yytext[yyleng-2]='\0';
126  field = strdup(yytext);
127  field = g_strchomp ( field );
128  field = g_strchug ( field );
129  g_hash_table_insert (  txt_fields_table, (void*) field_id,  (void*) field );
```

```
130   BEGIN INSIDE;
131 }
132
133 <IN_FIELD_TXT>.|\n {;}
134 <IN_FIELD_NUM>.|\n {;}
135
136 {CATEG}[^\n]*\}                 { printf("%s\n",yytext);}
137
138 <INITIAL>.|\n {;}
139
140 %%
```

3.3.2 Codificação

Estruturas de Dados e bibliotecas utilizadas

Como pode verificar foi necessário recorrer a estruturas de dados complexas para a resolução deste problema, nomeadamente hash tables. Ora, segundo o conselho do professor José João, foi reutilizado código da biblioteca da GLib, biblioteca essa extremamente otimizada. De seguida explicitam-se todas as variáveis e bibliotecas utilizadas:

```
1  %{
2    /*
3     ******************************************************************************
4     *   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
5     *   All Rights Reserved.
6     *
7     ******************************************************************************
8     *   Content : 2b) bibtex file normalizer and pretty-printer
9     ******************************************************************************/
10
11 #include <stdio.h>
12 #include <glib.h>
13 #include <stdlib.h>
14 #include <string.h>
15
16   //HashTable
17   GHashTable *txt_fields_table;
18   GHashTable *num_fields_table;
19   GHashTable *authors_table;
20   char* field_id;
21   char* field;
22   char* author_id = "author";
23   char* title_id = "title";
24   char* author_key;
25   char* title_key;
26   char* author_initial;
27   char* author_lastname;
28
29   //function sig
30   void pretty_print();
31
32   %}
```

Métodos Adicionais

Como poderá constatar na seção **??** existe 1 métodos adicional invocado no final da leitura de cada entrada do ficheiro, tendo por função imprimir o registo num formato normalizado:

```
%{
  void pretty_print(){
    int field_num = 0;
    char* current_value;

    GHashTableIter iter;
    gpointer key, value;

    if( title_key != NULL ){
      current_value =  g_hash_table_lookup ( txt_fields_table,(void*) title_key);
      g_hash_table_remove ( txt_fields_table, (void*) title_key);
      printf("\t%s {%s}", title_key, current_value);
      field_num++;
    }

    if ( author_key != NULL ){
      if(field_num > 0){ printf(",\n"); }
      printf("\t%s {", author_key);
      int number_authors = g_hash_table_size ( authors_table );
      int author_num = 1;
      g_hash_table_iter_init (&iter, authors_table );
      while (g_hash_table_iter_next (&iter, &key, &value)){
        if(author_num > 1 ){ printf("\n\t\tand "); }
        char* name = (char*) key;
        printf("%s", name);
        g_hash_table_iter_remove (&iter);
        author_num++;
      }
      printf("}");
      field_num++;
    }

    int size = g_hash_table_size ( txt_fields_table );
    size += g_hash_table_size ( num_fields_table );
    g_hash_table_iter_init (&iter, txt_fields_table );
    while (g_hash_table_iter_next (&iter, &key, &value)){
      if(field_num > 0 ){ printf(",\n"); }
      char* val = (char*) value;
      char* ke = (char*) key;
      printf("\t%s {%s}", ke, val);
      g_hash_table_iter_remove (&iter);
    }

    g_hash_table_iter_init (&iter, num_fields_table );
    while (g_hash_table_iter_next (&iter, &key, &value)){
      if(field_num > 0 ){ printf(",\n"); }
      char* val = (char*) value;
      char* ke = (char*) key;
      printf("\t%s %s", ke, val);
      g_hash_table_iter_remove (&iter);
    }
    printf("\n}\n\n");
  }
```

Main

O método main é também bastante simples, apenas inicializando as hashtables. Passaremos de seguida a explicitá-lo:

```
int main(){
  txt_fields_table = g_hash_table_new(g_str_hash, g_str_equal);
  num_fields_table = g_hash_table_new(g_str_hash, g_str_equal);
  authors_table = g_hash_table_new(g_str_hash, g_str_equal);
  yylex();
  return (0);
}
```

O código completo da ferramenta é passível de consulta na seção **??** na página **??**.

3.4 alínea c) ficheiro bib_norm_3.l

3.4.1 Padrões de frases a encontrar, através de ER

Podemos considerar que no contexto do problema apresentado temos interesse no campo author de cada entrada, ou seja, podemos reaproveitar grande parte do trabalho desenvolvido na alínea anterior. Tal tarefa é facilitada com base nas seguintes definições de expressões regulares a serem utilizadas no Flex:

```
LETRA [A-Za-z]
  LETRA_NUM [0-9A-Za-z]
  NUM [0-9]
  CATEG \@{LETRA}+\{
  FIELD_START [\{\"]
  AUTHOR_ID [ \t]*[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*
  AUTHOR_SEP [ \t]and[ \t]
  AUTHOR_BREAK [^=]+(\"|\})
```

Comutação de contextos em Flex através de start conditions

Por forma a realizar a correta análise dos padrões de texto foi necessário adicionar start conditions às regras das expressões regulares. Deste modo, apresenta-se a lista de todas as start conditions utilizadas:

```
%x INSIDE IN_AUTHOR IN_FIELD_TXT IN_FIELD_NUM AUTHOR_DIV START_AUTHOR
  \end{itemize}

\subsubsection{Expressões Regulares e acções resultantes}
\label{2cer}
Definidas as start conditions, resta-nos explicitar todas as expressões regulares,
    incluindo aquelas que dão início à comutação entre contextos do flex.
  \begin{lstlisting}
%%

{CATEG}[^=]*,              { BEGIN INSIDE; }

<INSIDE>{AUTHOR_ID}{FIELD_START}  {
  yytext[yyleng-2]='\0';
  author_key = strdup ( yytext );
  author_key = g_strchomp ( author_key );
  author_key = g_strchug ( author_key );
  BEGIN START_AUTHOR;
}

<INSIDE>[ \n\t]*,          { BEGIN INSIDE; }
<INSIDE>[ \n\t]*\}         { BEGIN INITIAL; }

<INSIDE>.|\n                 { BEGIN INSIDE;}
```

```
24
25      <START_AUTHOR>.             {
26        author_initial = (char*) malloc ( 4* sizeof(char));
27        author_initial[0] = yytext[0];
28        author_initial[1] = '.';
29        author_initial[2]=' ';
30        author_initial[3]='\0';
31        BEGIN IN_AUTHOR;
32      }
33
34      <IN_AUTHOR>[^= \n\t]*/[ ]and[ ]      {
35        author_lastname = strdup ( yytext );
36        author_lastname = g_strchomp ( author_lastname );
37        author_lastname = g_strchug ( author_lastname );
38        char *initial_plus_lastname;
39        int size = strlen(author_lastname);
40        size += strlen( author_initial) + 1;
41        initial_plus_lastname = (char*) malloc ( size * sizeof(char));
42        strcpy(initial_plus_lastname, author_initial);
43        strcat(initial_plus_lastname, author_lastname);
44        g_hash_table_insert (  authors_table, (void*) initial_plus_lastname,   GINT_TO_POINTER
              (0) );
45        BEGIN AUTHOR_DIV;
46      }
47
48      <IN_AUTHOR>[^= \n\t]*[ \t]?(\"|\})/,     {
49        yytext[yyleng-1]='\0';
50      author_lastname = strdup ( yytext );
51      author_lastname = g_strchomp ( author_lastname );
52      author_lastname = g_strchug ( author_lastname );
53      char *initial_plus_lastname;
54      int size = strlen(author_lastname);
55      size += strlen( author_initial) + 1;
56      initial_plus_lastname = (char*) malloc ( size * sizeof(char));
57      strcpy(initial_plus_lastname, author_initial);
58      strcat(initial_plus_lastname, author_lastname);
59      g_hash_table_insert (  authors_table, (void*) initial_plus_lastname,   GINT_TO_POINTER
            (1) );
60      BEGIN AUTHOR_DIV;
61    }
62
63    <IN_AUTHOR>.                { ; }
64    <AUTHOR_DIV>,               { check_authors(); BEGIN INSIDE; }
65    <AUTHOR_DIV>[ ]and[ ]               { BEGIN START_AUTHOR; }
66    <INITIAL>.|\n {;}
67
68    %%
```

3.4.2 Codificação

Estruturas de Dados e bibliotecas utilizadas

Como pode verificar foi necessário recorrer a estruturas de dados complexas para a resolução deste problema, nomeadamente hash tables. Ora, segundo o conselho do professor José João, foi reutilizado código da biblioteca da GLib, biblioteca essa extremamente otimizada. De seguida explicitam-se todas as variáveis e bibliotecas utilizadas:

```
1   %{
2
```

```
3       /*
4        ********************************************************************************
5        *    Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
6        *    All Rights Reserved.
7        *
8        ********************************************************************************
9        *    Content : 2c) bibtex co-authoring graph builder for a given normalized.
10       *              author name.
11       ********************************************************************************/
12
13  #include <stdio.h>
14  #include <glib.h>
15  #include <stdlib.h>
16  #include <string.h>
17
18      //HashTable
19      GHashTable *authors_table;
20      GHashTable *coauthors_table;
21      char* author_key;
22      char* author_initial;
23      char* author_lastname;
24      char* author_name;
25      //function sig
26      void print_graph();
27      void check_authors();
28
29      %}
```

Métodos Adicionais

Como poderá constatar na seção **??** existem 2 métodos adicionais. Um (check_authors()) invocado no final da leitura de cada entrada do ficheiro, tendo por função verificar se o autor em causa está referenciado no artigo e se sim, calcular o mapa dos seus co-autores. O outro método (graph_print()) é invocado aquando da impressão do grafo no format Dot:

```
1
2       void check_authors(){
3         if ( g_hash_table_contains ( authors_table ,(void*) author_name ) ){
4           g_hash_table_remove( authors_table, author_name );
5           GHashTableIter iter;
6           gpointer key, value;
7           g_hash_table_iter_init (&iter, authors_table );
8           while (g_hash_table_iter_next (&iter, &key, &value)){
9             char* coauthor_name = (char*) key;
10            int number_entries = 0;
11            if ( g_hash_table_contains ( coauthors_table ,key ) ){
12              number_entries = GPOINTER_TO_INT( g_hash_table_lookup ( coauthors_table , key));
13              number_entries++;
14              g_hash_table_replace (  coauthors_table, key, GINT_TO_POINTER( number_entries )
                    );
15            }
16            else{
17              number_entries++;
18              g_hash_table_insert (  coauthors_table, key,   GINT_TO_POINTER (number_entries)
                    );
19            }
20          }
21        }
```

```
22        g_hash_table_remove_all( authors_table );
23      }
24
25      void graph_print(){
26
27        GHashTableIter iter;
28        gpointer key, value;
29        g_hash_table_iter_init (&iter, coauthors_table );
30
31        printf( "digraph pl_2_2_a {\n//title\nlabelloc=\"t\";\nlabel=\"%s Document
              collaboration and co-authoring diagram\";rankdir=TB;\nresolution=300;size
              =\"8,5\";", author_name);
32        printf("\"%s\"[shape = box,style=filled,color=\"red\", style=rounded, fontsize=16
              fontname=helvetica];\n", author_name);
33        printf("node [shape = box, style=rounded, fontsize=12 fontname=helvetica]");
34        while (g_hash_table_iter_next (&iter, &key, &value))
35        {
36          char* coauthor_name = g_str_to_ascii ((char*) key, "C");
37          int number_entries = GPOINTER_TO_INT( value );
38          printf("\"%s\" -> \"%s\" [ label = \"%d\" ]\n", author_name, coauthor_name,
                number_entries );
39        }
40        printf("}\n");
41      }
```

Main

O método main é também bastante simples. Primeiramente inicializa as hashtables, sendo que de seguida coloca na variável author_name o valor do author a procurar artigos com co-autores. Após a leitura do ficheiro imprime no formato da linguagem Dot o grafo de co-autorias. Passaremos de seguida a explicitá-lo:

```
1     int main(int argc, char** argv){
2       authors_table = g_hash_table_new(g_str_hash, g_str_equal);
3       coauthors_table = g_hash_table_new(g_str_hash, g_str_equal);
4       author_name = strdup(argv[1]);
5       yylex();
6       graph_print();
7       return (0);
8     }
```

O código completo da ferramenta é passível de consulta na seção **??** na página **??**.

3.5 Testes realizados e Resultados

Foi realizado um script da shell com vista a facilmente ilustrar as funcionalidades da ferramenta, sendo que essas mesma script lê o ficheiro exemplo lp.bib , criando os ficheiros ex2a.html, ex2b.bib e ex2c.gv. A partir do ficheiro ex2c.gv cria ainda o ficheiro ex2c.png, demonstrado na figura **??**. Na figura **??** poderá ainda ver a página html exemplo resultante.

Shell Script

```
1 #!bin/sh
2
3     echo "/*
4       ******************************************************************************
```

```
5       *Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
6       *   All Rights Reserved.
7       *
8       ********************************************************************************
9       *   Content : 2a) bibtex category counter (phDThesis, Misc, InProceeding,
10        *             etc.), that occur in a document
11      *           : 2b) bibtex file normalizer and pretty-printer
12      *           : 2c) bibtex co-authoring graph builder for a given normalized
13      *             author name
14      ********************************************************************************/"
15
16      make clean
17      flex bib_norm_1.l
18      make a1
19      ./bib_norm_1 < lp.bib > ex2a.html
20      echo "############################"
21      echo ">>>>>>>>> ex2a in ex2a.html"
22      echo "          opening file"
23      echo "############################"
24      open ex2a.html
25
26      make clean
27      flex bib_norm_2.l
28      make a2
29      ./bib_norm_2 < lp.bib > ex2b.bib
30      echo "############################"
31      echo ">>>>>>>>> ex2b in ex2b.bib"
32      echo "############################"
33
34      make clean
35      flex bib_norm_3.l
36      make a3
37      ./bib_norm_3 "P. Henriques" < lp.bib > ex2c.gv
38      dot ex2c.gv -Tpng > ex2c.png
39      echo "############################"
40      echo ">>>>>>>>> ex2c in ex2c.png"
41      echo "          opening file"
42      echo "############################"
43      open ex2c.png
44      echo "done"
```

Página HTML exemplo resultante da alínea 2a

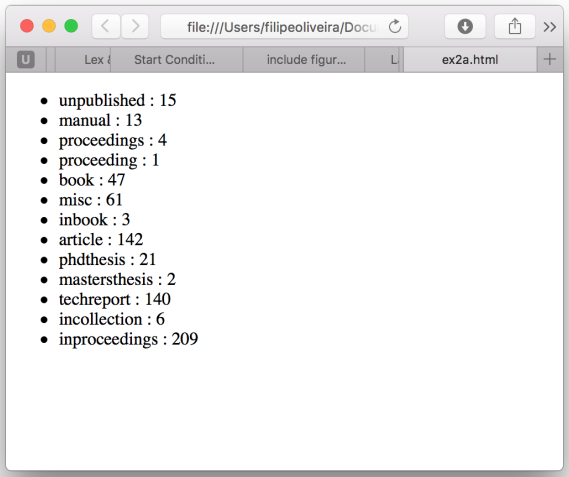

Figura 3.1: Página HTML exemplo resultante da alínea 2a a partir da leitura do ficheiro lp.bib

Grafo exemplo resultante da alínea 2c

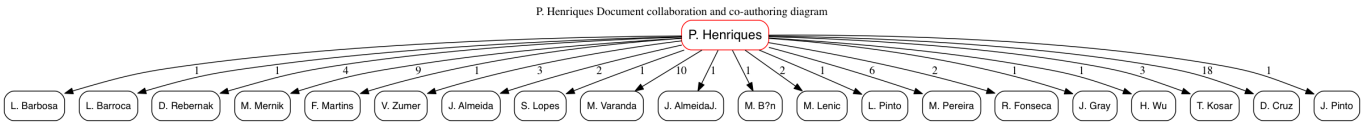

Figura 3.2: Grafo exemplo criado com as relações de co-autoria do autor P.Henriques a partir da leitura do ficheiro lp.bib

Capítulo 4

Conclusão

Relativamente ao estado final do projecto acredito que foram cumpridos todos os requisitos, sendo que o segundo exercício foi sem dúvida o mais desafiante dada a enorme quantidade de dados e o tipo de dados em si a serem analisados. Reconhecer por si só quais as sequências de caracteres válidas foi um desafio.

Naturalmente que a partir da alínea 2.2.b a alínea 2.2.c foi de extrema facilidade, uma vez que todo o trabalho de análise já estava realizado.

Foi ainda tido em conta a possiblidade de recuperar de erros de leitura na alínea 2.2.b o que facilitou o input correct de dados e posterior tratamento. O recurso à biblioteca Glib, recomendada pelo professor José João num aula laboratorial permitiu-me ambientar ainda mais com código desenvolvido por terceiros e sua correcta análise e integração nos meus projectos.

Faço um balanço positivo do trabalho prático, pois, apesar de ser extremamente "time consuming"retirei muito conhecimento no que da análise de dados e processamento de linguagens diz respeito.

Apêndice A

Código do Programa da alínea 1a

```
%{

/*
*************************************************************************
*   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
*   All Rights Reserved.
*
*************************************************************************
*   Content : 2.1) Simple OWL Ontology Graphic Viewer
*
*
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* relation;
char* begin;
char* end;
char* data;
char* class;
char* value;
char* class_class;
char* class_subclass;

void print_prop();
void print_data();
void print_class();
%}

LETRA [A-Za-z]
LETRA_NUM [0-9A-Za-z]
NUM [0-9]
OBJ_PROP_AXIOM SubObjectPropertyOf|EquivalentObjectProperties|DisjointObjectProperties|
    InverseObjectProperties|ObjectPropertyDomain|ObjectPropertyRange|
    FunctionalObjectProperty|InverseFunctionalObjectProperty|ReflexiveObjectProperty|
    IrreflexiveObjectProperty|SymmetricObjectProperty|AsymmetricObjectProperty|
    TransitiveObjectProperty
DAT_PROP_AXIOM SubDataPropertyOf|EquivalentDataProperties|DisjointDataProperties|
    DataPropertyDomain|DataPropertyRange|FunctionalDataProperty
CLA_PROP_AXIOM SubClassOf|EquivalentClasses|DisjointClasses|DisjointUnion
```

```
38
39 %x IN_PROP IN_DATA RELATION RELATION_BEG_END DATA DATA_BEG_END VALUE IN_CLASS CLASS_BEG_END
40
41 %%
42
43 [^<> ]*<{DAT_PROP_AXIOM}> { BEGIN IN_DATA; }
44 [^<> ]*<{OBJ_PROP_AXIOM}> { BEGIN IN_PROP; }
45 [^<> ]*<{CLA_PROP_AXIOM}> { BEGIN IN_CLASS; }
46
47 <IN_CLASS>[^<>]*<Class[ \t\n]*IRI=\" {BEGIN CLASS_BEG_END;}
48 <IN_CLASS><\/{CLA_PROP_AXIOM} {BEGIN INITIAL;}
49 <IN_CLASS>.|\n {;}
50
51 <CLASS_BEG_END>[^">]+/\" {
52                     if ( class_subclass == NULL ){
53                       class_subclass = strdup(yytext);
54                     }
55                     else{
56                       class_class = strdup(yytext);
57                       print_class();
58                     }
59                   }
60
61 <CLASS_BEG_END>\/> { BEGIN IN_CLASS;  }
62 <CLASS_BEG_END>.|\n {;}
63
64 <DATA>[^">]+/\" {data=strdup(yytext);}
65 <DATA>\/> {BEGIN IN_DATA;}
66 <DATA>.|\n {;}
67
68 <IN_DATA>[^<>]*<DataProperty[ \t\n]*IRI=\" { BEGIN DATA;}
69 <IN_DATA>[^<>]*<Class[ \t\n]*IRI=\" { BEGIN DATA_BEG_END;}
70 <IN_DATA>[^<>]*<Datatype[ \t\n]*abbreviatedIRI=\" {  BEGIN VALUE;}
71 <IN_DATA><\/{DAT_PROP_AXIOM} {BEGIN INITIAL;}
72 <IN_DATA>.|\n {;}
73
74 <VALUE>[^"<>]+/\" { value = strdup(yytext); print_data(); }
75 <VALUE>\/> {BEGIN IN_DATA;}
76 <VALUE>.|\n {;}
77
78 <DATA_BEG_END>[^">]+/\" { class = strdup(yytext); }
79 <DATA_BEG_END>\/> {BEGIN IN_DATA;}
80 <DATA_BEG_END>.|\n {;}
81
82 <IN_PROP>[^<>]*<ObjectProperty[ \t\n]*IRI=\" { BEGIN RELATION;  }
83 <IN_PROP>[^<>]*<Class[ \t\n]*IRI=\" {  BEGIN RELATION_BEG_END;}
84 <IN_PROP><\/{OBJ_PROP_AXIOM} { BEGIN INITIAL;}
85 <IN_PROP>.|\n {;}
86
87 <RELATION>[^"<>]+/\" { relation=strdup(yytext);}
88 <RELATION>\/> { BEGIN IN_PROP;}
89 <RELATION>.|\n {;}
90
91 <RELATION_BEG_END>[^"<>]+/\" {
92                                 if ( begin == NULL ){
93                                   begin = strdup(yytext);
94                                 }
95                                 else{
96                                   end = strdup(yytext);
```

```
97                                      print_prop();
98                                    }
99                                  }
100
101 <RELATION_BEG_END>\/> { BEGIN IN_PROP;}
102 <RELATION_BEG_END>.|\n { ;}
103
104 <INITIAL>.|\n {;}
105
106 %%
107
108 void print_class(){
109   printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n",
        class_subclass);
110   printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n",
        class_class);
111   printf("\"%s\" -> \"%s\" [ label = \"SubClassOf\" , fontsize=8 , fontcolor=\"blue\", color
        =\"blue\" ]\n", class_subclass, class_class );
112   class_subclass=NULL;
113   class_class=NULL;
114 }
115
116 void print_data(){
117   printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n", class);
118   printf("\"%s\" [shape = box, style=filled ,color=\"red\", fontsize=12 fontname=helvetica];\
        n", value);
119   printf("\"%s\" -> \"%s\" [ label = \"%s\" ]\n", class, value, data );
120   class=NULL;
121   value=NULL;
122   data=NULL;
123 }
124
125 void print_prop(){
126   printf("\"%s\" [shape = box, style=rounded, fontsize=12 fontname=helvetica];\n", begin);
127   printf("\"%s\" -> \"%s\" [ label = \"%s\" ]\n", begin, end, relation );
128   begin=NULL;
129   relation=NULL;
130   end=NULL;
131 }
132
133 void graph_print(){
134   printf( "digraph pl_2_1 {\n//title\nlabelloc=\"t\";\nlabel=\"OWL concept diagram\";rankdir
        =TB;\nresolution=300;size=\"8,5\";\n");
135 }
136
137 int yywrap(){return 1;}
138
139 int main(int argc, char** argv){
140   graph_print();
141   yylex();
142   printf("}\n");
143   return (0);
144 }
```

Apêndice B

Código do Programa da alínea 2a

```
%{

/*
*****************************************************************************
*   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
*   All Rights Reserved.
*
*****************************************************************************
*   Content : Simple bibtex category counter (phDThesis, Misc, InProceeding,
*             etc.), that occur in a document
*****************************************************************************/

#include <stdio.h>
#include <glib.h>
#include <stdlib.h>
#include <string.h>

//HashTable
GHashTable *table;

%}

LETRA [A-Za-z]
CATEG \@{LETRA}+\{
LETRA_NUM [0-9A-Za-z]
ID ({LETRA_NUM}|:)+

%%
{CATEG}/[^=]*,      {
                yytext++; yytext[yyleng-2]='\0';
                char* key = g_ascii_strdown (yytext, yyleng-2);
                if ( g_hash_table_contains ( table ,(void*) key ) ){
                  int value;
                  value = GPOINTER_TO_INT( g_hash_table_lookup ( table,(void*) key));
                  value++;
                  g_hash_table_replace (  table, (void*) key,GINT_TO_POINTER(value) );
                }
                else {
                  int value = 1;
                  gboolean add_result = g_hash_table_insert (  table, (void*) key,
                      GINT_TO_POINTER(value) );
                }
```

```
42                }
43 .|\n|\t { ; }
44 %%
45
46 int yywrap(){return 1;}
47
48 static void print_key_value(gpointer key, gpointer value, gpointer userdata)
49 {
50         int val = (int) value;
51         char* ke = (char*) key;
52         printf("<li>%s : %d</li>\n", ke, val);
53 }
54
55
56 int main(){
57   table = g_hash_table_new(g_str_hash, g_str_equal);
58   yylex();
59   printf("<!DOCTYPE html>\n<html>\n<body>\n<ul>\n");
60   g_hash_table_foreach(table,print_key_value, NULL );
61   printf("</ul>\n</body>\n</html>\n");
62   return (0);
63 }
```

Apêndice C

Código do Programa da alínea 2b

```
%{

/*
*****************************************************************************
*   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
*   All Rights Reserved.
*
*****************************************************************************
*   Content : 2b) bibtex file normalizer and pretty-printer
*****************************************************************************/

#include <stdio.h>
#include <glib.h>
#include <stdlib.h>
#include <string.h>

//HashTable
GHashTable *txt_fields_table;
GHashTable *num_fields_table;
GHashTable *authors_table;
char* field_id;
char* field;
char* author_id = "author";
char* title_id = "title";
char* author_key;
char* title_key;
char* author_initial;
char* author_lastname;

//function sig
void pretty_print();

%}

LETRA [A-Za-z]
LETRA_NUM [0-9A-Za-z]
NUM [0-9]
CATEG \@{LETRA}+\{
FIELD_ID ^[^,}"][^=]*=[ \t]*
FIELD_BREAK [^=]*[ \t\n]*(\"|\})
FIELD_BREAK_NUM [^=\}]*\}?
FIELD_START [\{\"]
```

```
AUTHOR_ID ^[ \t]*[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*
TITLE_ID ^[ \t]*[Tt][Ii][Tt][Ll][Ee][ \t]*=[ \t]*
AUTHOR_SEP [ \t]and[ \t]
AUTHOR_BREAK [^=]+(\"|\})

%x INSIDE IN_AUTHOR IN_FIELD_TXT IN_FIELD_NUM AUTHOR_DIV START_AUTHOR

%%

{CATEG}[^=]*,                {
                              printf("%s\n",yytext);
                              BEGIN INSIDE;
                             }

<INSIDE>{TITLE_ID}{FIELD_START}  {
                              yytext[yyleng-2]='\0';
                              title_key = strdup ( yytext );
                              title_key = g_strchomp ( title_key );
                              title_key = g_strchug ( title_key );
                              field_id = strdup(title_key);
                              BEGIN IN_FIELD_TXT;}

<INSIDE>{AUTHOR_ID}{FIELD_START}  {
                              yytext[yyleng-2]='\0';
                              author_key = strdup ( yytext );
                              author_key = g_strchomp ( author_key );
                              author_key = g_strchug ( author_key );
                              BEGIN START_AUTHOR;
}

<INSIDE>{FIELD_ID}/{LETRA_NUM}  {
                             yytext[yyleng-1]='\0';
                              field_id = strdup ( yytext );
                              field_id = g_strchomp ( field_id );
                              field_id = g_strchug ( field_id );
                              BEGIN IN_FIELD_NUM; }

<INSIDE>{FIELD_ID}{FIELD_START}  {
                              yytext[yyleng-2]='\0';
                              field_id = strdup ( yytext );
                              field_id = g_strchomp ( field_id );
                              field_id = g_strchug ( field_id );
                              BEGIN IN_FIELD_TXT;
                              }

<INSIDE>[ \n\t]*, { BEGIN INSIDE; }
<INSIDE>[ \n\t]*\} { BEGIN INITIAL; }
<INSIDE>.|\n { BEGIN INSIDE;}

<START_AUTHOR>.              {
                                 author_initial = (char*) malloc ( 4* sizeof(char));
                                 author_initial[0] = yytext[0];
                                 author_initial[1] = '.';
                                 author_initial[2]=' ';
                                 author_initial[3]='\0';
                                BEGIN IN_AUTHOR;
                              }

<IN_AUTHOR>[^= \n\t]*/[ ]and[ ]      {
```

```
102                             author_lastname = strdup ( yytext );
103                             author_lastname = g_strchomp ( author_lastname );
104                             author_lastname = g_strchug ( author_lastname );
105                             char *initial_plus_lastname;
106                             int size = strlen(author_lastname);
107                             size += strlen( author_initial) + 1;
108                             initial_plus_lastname = (char*) malloc ( size * sizeof(char));
109                             strcpy(initial_plus_lastname, author_initial);
110                             strcat(initial_plus_lastname, author_lastname);
111                             g_hash_table_insert (  authors_table, (void*)
                                    initial_plus_lastname,   GINT_TO_POINTER (0) );
112                            BEGIN AUTHOR_DIV;
113                            }
114
115 <IN_AUTHOR>[^= \n\t]*[ \t]?(\"|\})/,     {
116                             yytext[yyleng-1]='\0';
117                             author_lastname = strdup ( yytext );
118                             author_lastname = g_strchomp ( author_lastname );
119                             author_lastname = g_strchug ( author_lastname );
120                             char *initial_plus_lastname;
121                             int size = strlen(author_lastname);
122                             size += strlen( author_initial) + 1;
123                             initial_plus_lastname = (char*) malloc ( size * sizeof(char));
124                             strcpy(initial_plus_lastname, author_initial);
125                             strcat(initial_plus_lastname, author_lastname);
126                             g_hash_table_insert (  authors_table, (void*)
                                    initial_plus_lastname,   GINT_TO_POINTER (0) );
127                            BEGIN AUTHOR_DIV;
128 }
129
130 <IN_AUTHOR>.                { ; }
131
132 <AUTHOR_DIV>,               { BEGIN INSIDE; }
133
134 <AUTHOR_DIV>[ ]and[ ]       { BEGIN START_AUTHOR; }
135
136
137 <IN_FIELD_NUM>{FIELD_BREAK_NUM}[ \n\t\r]*\} {
138                               yytext[yyleng-1]='\0';
139                               field = strdup(yytext);
140                               field = g_strchomp ( field );
141                               field = g_strchug ( field );
142                               g_hash_table_insert (  num_fields_table, (void*) field_id,  (
                                      void*) field );
143                               pretty_print();
144                              BEGIN INITIAL;
145                              }
146
147 <IN_FIELD_NUM>{FIELD_BREAK_NUM}[ \n\t\r]*(#[^,=]*)?, {
148                               yytext[yyleng-1]='\0';
149                               field = strdup(yytext);
150                               field = g_strchomp ( field );
151                               field = g_strchug ( field );
152                               g_hash_table_insert (  num_fields_table, (void*) field_id,  (
                                      void*) field );
153                              BEGIN INSIDE;
154                              }
155
156 <IN_FIELD_TXT>{FIELD_BREAK}[ \n\t\r]*\} {
```

```
157                                 yytext[yyleng-2]='\0';
158                                 field = strdup(yytext);
159                                 field = g_strchomp ( field );
160                                 field = g_strchug ( field );
161                                 g_hash_table_insert ( txt_fields_table, (void*) field_id, (
                                        void*) field );
162                                 BEGIN INITIAL;
163                                 pretty_print();
164                                 }
165
166 <IN_FIELD_TXT>{FIELD_BREAK}[ \n\t\r]*(#[^,=]*)?, {
167                                 yytext[yyleng-2]='\0';
168                                 field = strdup(yytext);
169                                 field = g_strchomp ( field );
170                                 field = g_strchug ( field );
171                                 g_hash_table_insert ( txt_fields_table, (void*) field_id, (
                                        void*) field );
172                                 BEGIN INSIDE;
173                                 }
174
175 <IN_FIELD_TXT>.|\n {;}
176 <IN_FIELD_NUM>.|\n {;}
177
178 {CATEG}[^\n]*\}                 { printf("%s\n",yytext);}
179
180 <INITIAL>.|\n {;}
181
182 %%
183
184 void pretty_print(){
185   int field_num = 0;
186   char* current_value;
187
188   GHashTableIter iter;
189   gpointer key, value;
190
191   if( title_key != NULL ){
192     current_value = g_hash_table_lookup ( txt_fields_table,(void*) title_key);
193     g_hash_table_remove ( txt_fields_table, (void*) title_key);
194     printf("\t%s {%s}", title_key, current_value);
195     field_num++;
196   }
197
198   if ( author_key != NULL ){
199     if(field_num > 0){ printf(",\n"); }
200     printf("\t%s {", author_key);
201     int number_authors = g_hash_table_size ( authors_table );
202     int author_num = 1;
203     g_hash_table_iter_init (&iter, authors_table );
204     while (g_hash_table_iter_next (&iter, &key, &value)){
205       if(author_num > 1 ){ printf("\n\t\tand "); }
206       char* name = (char*) key;
207       printf("%s", name);
208       g_hash_table_iter_remove (&iter);
209       author_num++;
210     }
211     printf("}");
212     field_num++;
213   }
```

```
214
215   int size = g_hash_table_size ( txt_fields_table );
216   size += g_hash_table_size ( num_fields_table );
217   g_hash_table_iter_init (&iter, txt_fields_table );
218   while (g_hash_table_iter_next (&iter, &key, &value)){
219     if(field_num > 0 ){ printf(",\n"); }
220     char* val = (char*) value;
221     char* ke = (char*) key;
222     printf("\t%s {%s}", ke, val);
223     g_hash_table_iter_remove (&iter);
224   }
225
226   g_hash_table_iter_init (&iter, num_fields_table );
227   while (g_hash_table_iter_next (&iter, &key, &value)){
228     if(field_num > 0 ){ printf(",\n"); }
229     char* val = (char*) value;
230     char* ke = (char*) key;
231     printf("\t%s %s", ke, val);
232     g_hash_table_iter_remove (&iter);
233   }
234   printf("\n}\n\n");
235 }
236
237 int yywrap(){return 1;}
238
239 int main(){
240   txt_fields_table = g_hash_table_new(g_str_hash, g_str_equal);
241   num_fields_table = g_hash_table_new(g_str_hash, g_str_equal);
242   authors_table = g_hash_table_new(g_str_hash, g_str_equal);
243   yylex();
244   return (0);
245 }
```

Apêndice D

Código do Programa da alínea 3a

```
%{

/*
*****************************************************************************
*   Copyright(C) 2016 Filipe Oliveira, Universidade do Minho
*   All Rights Reserved.
*
*****************************************************************************
*   Content : 2c) bibtex co-authoring graph builder for a given normalized.
*             author name.
*****************************************************************************/

#include <stdio.h>
#include <glib.h>
#include <stdlib.h>
#include <string.h>

//HashTable
GHashTable *authors_table;
GHashTable *coauthors_table;
char* author_key;
char* author_initial;
char* author_lastname;
char* author_name;
//function sig
void print_graph();
void check_authors();

%}

LETRA [A-Za-z]
LETRA_NUM [0-9A-Za-z]
NUM [0-9]
CATEG \@{LETRA}+\{
FIELD_START [\{\"]
AUTHOR_ID [ \t]*[Aa][Uu][Tt][Hh][Oo][Rr][ \t]*=[ \t]*
AUTHOR_SEP [ \t]and[ \t]
AUTHOR_BREAK [^=]+(\"|\})

%x INSIDE IN_AUTHOR IN_FIELD_TXT IN_FIELD_NUM AUTHOR_DIV START_AUTHOR

%%
```

```
43
44 {CATEG}[^=]*,              { BEGIN INSIDE; }
45
46 <INSIDE>{AUTHOR_ID}{FIELD_START}  {
47                             yytext[yyleng-2]='\0';
48                             author_key = strdup ( yytext );
49                             author_key = g_strchomp ( author_key );
50                             author_key = g_strchug ( author_key );
51                             BEGIN START_AUTHOR;
52 }
53
54 <INSIDE>[ \n\t]*,          { BEGIN INSIDE; }
55 <INSIDE>[ \n\t]*\}         { BEGIN INITIAL; }
56
57 <INSIDE>.|\n               { BEGIN INSIDE;}
58
59 <START_AUTHOR>.            {
60                             author_initial = (char*) malloc ( 4* sizeof(char));
61                             author_initial[0] = yytext[0];
62                             author_initial[1] = '.';
63                             author_initial[2]=' ';
64                             author_initial[3]='\0';
65                             BEGIN IN_AUTHOR;
66                            }
67
68 <IN_AUTHOR>[^= \n\t]*/[ ]and[ ]     {
69                             author_lastname = strdup ( yytext );
70                             author_lastname = g_strchomp ( author_lastname );
71                             author_lastname = g_strchug ( author_lastname );
72                             char *initial_plus_lastname;
73                             int size = strlen(author_lastname);
74                             size += strlen( author_initial) + 1;
75                             initial_plus_lastname = (char*) malloc ( size * sizeof(char));
76                             strcpy(initial_plus_lastname, author_initial);
77                             strcat(initial_plus_lastname, author_lastname);
78                             g_hash_table_insert (  authors_table, (void*)
                                   initial_plus_lastname,   GINT_TO_POINTER (0) );
79                             BEGIN AUTHOR_DIV;
80                             }
81
82 <IN_AUTHOR>[^= \n\t]*[ \t]?(\"|\})/,     {
83                             yytext[yyleng-1]='\0';
84                             author_lastname = strdup ( yytext );
85                             author_lastname = g_strchomp ( author_lastname );
86                             author_lastname = g_strchug ( author_lastname );
87                             char *initial_plus_lastname;
88                             int size = strlen(author_lastname);
89                             size += strlen( author_initial) + 1;
90                             initial_plus_lastname = (char*) malloc ( size * sizeof(char));
91                             strcpy(initial_plus_lastname, author_initial);
92                             strcat(initial_plus_lastname, author_lastname);
93                             g_hash_table_insert (  authors_table, (void*)
                                   initial_plus_lastname,   GINT_TO_POINTER (1) );
94                             BEGIN AUTHOR_DIV;
95 }
96
97 <IN_AUTHOR>.               { ; }
98 <AUTHOR_DIV>,              { check_authors(); BEGIN INSIDE; }
99 <AUTHOR_DIV>[ ]and[ ]              { BEGIN START_AUTHOR; }
```

```
<INITIAL>.|\n {;}

%%

void check_authors(){
  if ( g_hash_table_contains ( authors_table ,(void*) author_name ) ){
    g_hash_table_remove( authors_table, author_name );
    GHashTableIter iter;
    gpointer key, value;
    g_hash_table_iter_init (&iter, authors_table );
    while (g_hash_table_iter_next (&iter, &key, &value)){
      char* coauthor_name = (char*) key;
      int number_entries = 0;
      if ( g_hash_table_contains ( coauthors_table ,key ) ){
        number_entries = GPOINTER_TO_INT( g_hash_table_lookup ( coauthors_table , key));
        number_entries++;
        g_hash_table_replace (  coauthors_table, key, GINT_TO_POINTER( number_entries ) );
      }
      else{
        number_entries++;
        g_hash_table_insert (  coauthors_table, key,   GINT_TO_POINTER (number_entries) );
      }
    }
  }
  g_hash_table_remove_all( authors_table );
}

void graph_print(){

  GHashTableIter iter;
  gpointer key, value;
  g_hash_table_iter_init (&iter, coauthors_table );

  printf( "digraph pl_2_2_a {\n//title\nlabelloc=\"t\";\nlabel=\"%s Document collaboration
      and co-authoring diagram\";rankdir=TB;\nresolution=300;size=\"8,5\";", author_name);
  printf("\"%s\"[shape = box,style=filled ,color=\"red\", style=rounded, fontsize=16 fontname
      =helvetica];\n", author_name);
  printf("node [shape = box, style=rounded, fontsize=12 fontname=helvetica]");
  while (g_hash_table_iter_next (&iter, &key, &value))
  {
    char* coauthor_name = g_str_to_ascii ((char*) key, "C");
    int number_entries = GPOINTER_TO_INT( value );
    printf("\"%s\" -> \"%s\" [ label = \"%d\" ]\n", author_name, coauthor_name,
        number_entries );
  }
  printf("}\n");
}

int yywrap(){return 1;}

int main(int argc, char** argv){
  authors_table = g_hash_table_new(g_str_hash, g_str_equal);
  coauthors_table = g_hash_table_new(g_str_hash, g_str_equal);
  author_name = strdup(argv[1]);
  yylex();
  graph_print();
  return (0);
}
```