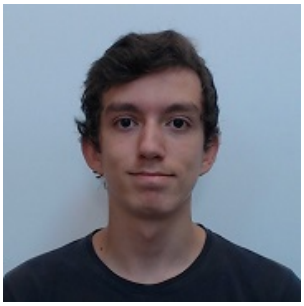
Universidade do Minho
Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Practical Assignment 2019/20
Enunciado 2.6 - Visualizador de árvores genealógicas

Junho 2020


Angélica Freitas
(A83761)


Jorge Mota
(A85272)


Rodrigo Pimentel
(A83765)

Conteúdo

1 Introdução

Foi nos proposto a criação de um visualizador de árvores genealógicas no âmbito da disciplina de Processamento de Linguagens. Para entender a linguagem produzida pelo API fornecido pelo "OWL", utilizámos o analisador léxico flex aliado com yacc que descrevem uma gramática de forma a produzir o output necessário para a representação de àrvores geneológicas.

1.1 Especificação

A gramática especificada segue formatos de representação de um sujeito, predicado e um objeto ao qual se refere (terminado com um `.`) com variações sintáticas.

```
<uri:subject> <uri:predicate> <uri:object> .
```

Estas variações incluem a utilização de `;` e `,` para reutilização de sujeito e sujeito/predicado respetivamente.

```
<uri:subject> <uri:predicate> <uri:object> ,
                    <uri:object> ;
                    <uri:predicate>   <uri:object> .
```

1.2 Formulação

Estruturar os dados interpretados para um grafo de relações e no fim imprimir o formato do ficheiro com a representação visual

A fim de produzir o respetivo grafo de relações familiares utilizamos o formato 'DOT'.

2 Flex

"Flex"feito pelo grupo com o objetivo de identificar predicados, objetos, comentários, sujeitos e operadores entre tripletos como a vírgula, ponto e ponto e vírgula. Para tal feito, froam usados vários estados para identificar cada elemento de um tripleto, todos os estados estão declarados na linha 13 no seguinte código:

```
1  %{
2  /* Declaracoes C diversas */
3
4  #include "yacc_aux.h"
5  #include "y.tab.h"
6  #include <ctype.h>
7  #include <string.h>
8
9  %}
10
11 %option stack noyywrap
12
13 %x NEXT_PREDICATE NEXT_OBJECT NEXT_OBJECT_NAME END COMMENT
14
15 %%
16
17 ^#                                      { BEGIN COMMENT; }
18 <COMMENT>.                              { ; }
19 <COMMENT>\n|\r                          { BEGIN INITIAL; }
20
21 [ ]+|\n                                 { ; }
22 :[A-Za-z0-9_]+                          { yylval.str_val = strdup(yytext+1);
       BEGIN NEXT_PREDICATE; return SUBJECT; }
23
24 <NEXT_PREDICATE>[ ]+|\n                 { ; }
25 <NEXT_PREDICATE>"rdf:type"              { BEGIN NEXT_OBJECT; return
       PREDICATE_INIT; }
26 <NEXT_PREDICATE>":gender"               { BEGIN NEXT_OBJECT; return
       PREDICATE_GENDER; }
27 <NEXT_PREDICATE>:[A-Za-z0-9]+          { yylval.str_val = strdup(yytext+1);
       BEGIN NEXT_OBJECT_NAME; return PREDICATE_RELATION; }
28
29 <NEXT_OBJECT>[ ]+|\n                    { ; }
30 <NEXT_OBJECT>[A-Za-z0-9]*:[A-Za-z0-9]+  { yylval.str_val = strdup(yytext);
       BEGIN END; return OBJECT; }
31
32 <NEXT_OBJECT_NAME>[ ]+|\n               { ; }
33 <NEXT_OBJECT_NAME>:[A-Za-z0-9_]+        { yylval.str_val = strdup(yytext+1);
       BEGIN END; return OBJECT; }
34
35 <END>[ ]+|\n                            { ; }
36 <END>"."                                { BEGIN INITIAL;        return yytext
       [0]; }
37 <END>","                               { BEGIN NEXT_OBJECT;    return yytext
       [0]; }
38 <END>";"                               { BEGIN NEXT_PREDICATE; return yytext
       [0]; }
39
40 %%
```

2.1 Tratar dos comentários

Com apenas 3 regras de "Flex"conseguimos identificar e tratar dos comentários.

```
1  ^#                                      { BEGIN COMMENT; }
```

```
<COMMENT>.                                  { ; }
<COMMENT>\n|\r                              { BEGIN INITIAL; }
```

Regra 1 - Identificar o ínicio de um comentário. Entrar no estado de COMMENT pois sabemos que todos os caracteres seguintes pertencem a um comentário.

Regra 2 - Filtrar todos os caracteres presentes dentro de um comentário.

Regra 3 - Identificar o fim de um comentário de linha singular. Voltar ao estado INITIAL.

2.2 Identificar um Sujeito

```
:[A-Za-z0-9_]+                              { yylval.str_val = strdup(yytext+1);
    BEGIN NEXT_PREDICATE; return SUBJECT; }
```

Regra 1 - Identificação de um URI (ATENÇÃO: Um URI poderá ser muito mais completo mas para efeitos deste trabalho simplificamos o REGEX). Depois de identificado um sujeito, sabemos que de seguida encontraremos um predicado, daí a mudança de estado para NEXT_PREDICATE.

2.3 Identificar Predicados

```
<NEXT_PREDICATE>[ ]+|\n                     { ; }
<NEXT_PREDICATE>"rdf:type"                  { BEGIN NEXT_OBJECT; return
    PREDICATE_INIT; }
<NEXT_PREDICATE>":gender"                   { BEGIN NEXT_OBJECT; return
    PREDICATE_GENDER; }
<NEXT_PREDICATE>:[A-Za-z0-9]+               { yylval.str_val = strdup(yytext+1);
    BEGIN NEXT_OBJECT_NAME; return PREDICATE_RELATION; }
```

Regra 1 - Ignorar espaços ou mudanças de linha entre Sujeito e Predicado.

Regra 2 - Identificar um Predicado de Inicialização de "NamedIndividual".

Regra 3 - Identificar um predicado de género. Estamos perante um predicado que se seguirá com um objeto do tipo "Male"or "Female".

Regra 4 - Identificar um predicado de relação (temPai, temMãe, temIrmão,...).

2.4 Identificar Objetos

```
<NEXT_OBJECT>[ ]+|\n                        { ; }
<NEXT_OBJECT>[A-Za-z0-9]*:[A-Za-z0-9]+      { yylval.str_val = strdup(yytext);
    BEGIN END; return OBJECT; }

<NEXT_OBJECT_NAME>[ ]+|\n                   { ; }
<NEXT_OBJECT_NAME>:[A-Za-z0-9_]+            { yylval.str_val = strdup(yytext+1);
    BEGIN END; return OBJECT; }
```

Regra 1 e 3 - Ignorar qualquer espaço ou mudança de linha enquanto não encontrarmos um objeto.

Regra 2 e 4 - Identificar um Objeto (URI).

2.5 Caracteres entre tripletos (Operadores)

```
<END>[ ]+|\n            { ; }
<END>"."                { BEGIN INITIAL;         return yytext[0]; }
<END>","                { BEGIN NEXT_OBJECT;     return yytext[0]; }
<END>";"                { BEGIN NEXT_PREDICATE; return yytext[0]; }
```

Regra 1 - Ignorar espaços e mudanças de linhas até encontrar um operador.

Regra 2 - Marcar o fim de um tripleto. Mudamos para o estado INITIAL.

Regra 3 - Marcar o fim de um tripleto e no próximo tripleto não haverá sujeito e predicado pois são os mesmo do tripleto anterior. Saltamos para o estado NEXT_OBJECT.

Regra 4 - Marcar o fim de um tripleto e no próximo tripleto não haverá sujeito pois é o mesmo do tripleto anterior. Mudamos para o estado NEXT_PREDICATE.

3 Estruturas auxiliares

Para guardar a informação para reproduzir o grafo de relações familiares, utilizamos a biblioteca 'glib' (Hashmaps **GHashTable** e Arrays dinâmicos **GPtrArray**).

Estruturamos os dados da seguinte forma:

```
typedef struct __relation
{
    const char* name;     // Name of the relation
    const char* target;   // Target of this relation
} relation;


typedef struct __person_data
{
    uint8_t* gender;      // Other data about this person
    GPtrArray* relations; // Dynamic array of relations
} person_data;


typedef GHashTable family_tree; // Key: char* , Value: person_data*
```

Juntamente, é fornecida uma interface com funções para realizar operações e manipular esta mesma estrutura localizada no mesmo cabeçalho (`family.h`).

4 Yacc

4.1 Generalização da gramática

A GIC (Gramática independente de contexto) especificada tem uma forma semelhante ao que se segue:

```
Line : Triplets '.'

Triplets : SUBJECT PREDICATE OBJECT
         | Triplets ',' OBJECT
         | Triplets ';' PREDICATE OBJECT
         ;
```

Sendo esta a base de toda a sintaxe de owl, notemos que a composição de tripletos utilizando os operadores `,` e `;` é suportada e tem um significado traduzivel para quando nao se aplicam os mesmos

Por exemplo:

```
:Subject :Predicate1 :Object1 ,
                     :Object2 ;
                     :Predicate2 :Object1 ,
                     :Object2 .
```

Seria equivalente a:

```
:Subject :Predicate1 :Object1 .
:Subject :Predicate1 :Object2 .
:Subject :Predicate2 :Object1 .
:Subject :Predicate2 :Object2 .
```

4.2 Implementação

De forma a que o tipo de informação seja mais fácil de manipular tendo em conta as diversas ações possíveis que podemos tomar, decidimos especializar a gramática a modo de realizar essas ações conforme predicados e objetos que vão aparecendo.

```
Lines : Lines Line
      |
      ;

Line : Triplets '.'

Triplets : SUBJECT Predicate OBJECT
         | Triplets ',' OBJECT
         | Triplets ';' Predicate OBJECT
         ;


Predicate : PREDICATE_INIT
          | PREDICATE_GENDER
          | PREDICATE_RELATION
          ;
```

De forma a reduzir a quantidade de código C presente junto com a gramática, incluimos ficheiros separados com esse código (`yacc_aux.c` e `yacc_aux.h`)

No fim do ficheiro de input são induzidas algumas relações familiares que são perceptíveis a partir da arvore até ao momento.

Esta opção está disponivel caso no executável seja fornecida a opção `-induced-relations`

5 Testes e Resultados

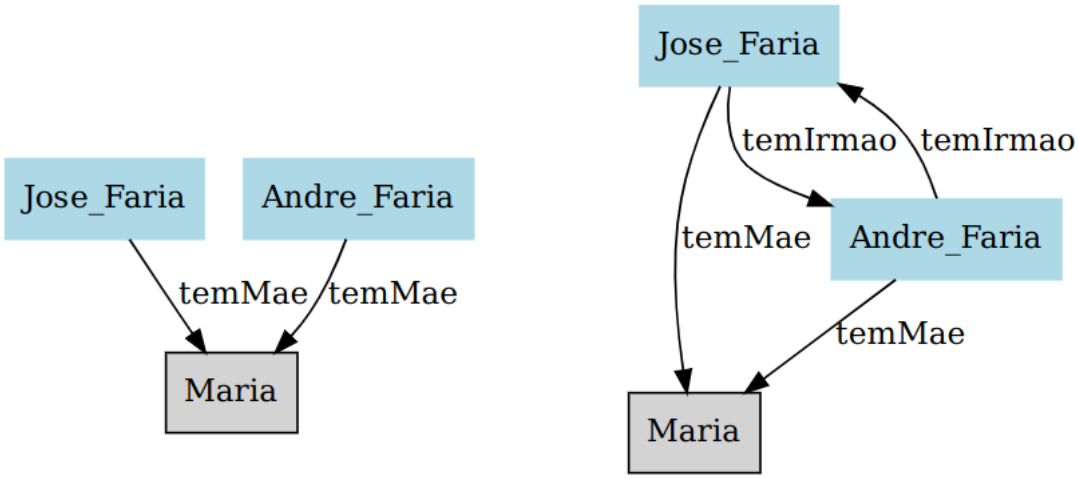


Figura 1: Teste simples com versão que deduziu os irmãos

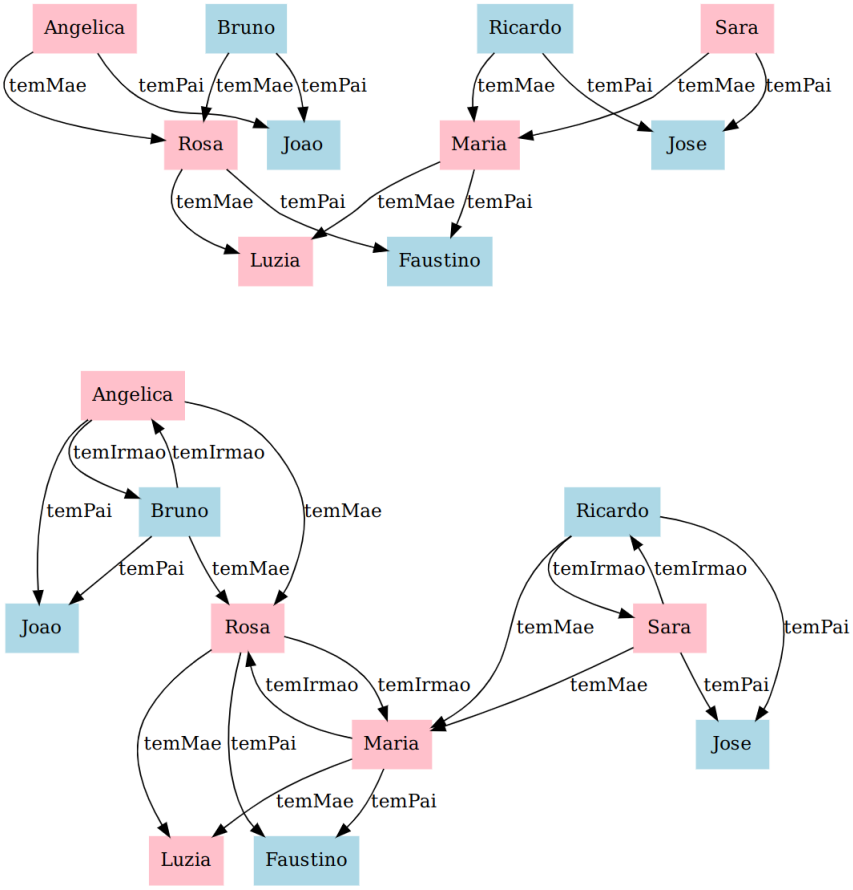


Figura 2: Teste com versão que deduziu os irmãos

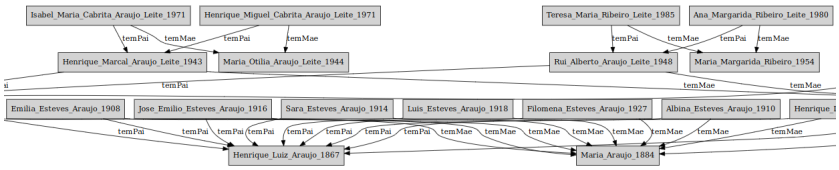


Figura 3: Parte do resultado do teste fornecido com o enunciado