Universidade do Minho
Departamento de Informática

Mestrado em Engenharia Informática

Processamento de Linguagens

Reverse Engineering dum Dicionário Financeiro

Grupo 6
A81765 Joana Matos
A82238 João Gomes

Braga, Portugal
28 de Junho de 2020

Resumo

Este trabalho consiste em desenvolver um processador de linguagem capaz de analisar um ficheiro de texto e, assim, passar esse mesmo texto para um formato mais "limpo"e percetível.

Para tal, será usado o par Flex-Yacc para alcançar este objetivo.

Conteúdo

Capítulo 1

Introdução

Um compilador ou um interpretador pode ser decomposto em duas partes:

- ler o programa fonte e descobrir a sua estrutura.
- processar esta estrutura para gerar um novo programa, neste caso um ficheiro.

No âmbito da disciplina de Processamento de Linguagens, este trabalho consiste em tirar o máximo partido de um ficheiro de texto relativo a um dicionário de negócios e finanças EN-PT aumentando a capacidade dos elementos do grupo de escrever gramáticas independentes de contexto (GIC) a partir de geradores de compiladores como é o caso do par flex/yacc. O objetivo é criar uma gramática que cubra o máximo de entradas possíveis deste dicionário financeiro convertendo-as para um formato mais tratável a partir da construção de um processador. Um exemplo poderá ser observado abaixo antes e depois do processamento.

```
dispute:
labour -            conflito (m) trabalhista
dissolution         dissolução (f)
distribution:
- costs             custos de distribuição
channels of -       anais (mpl) de distribuição
```

Figura 1.1: Excerto do dicionário antes do processamento.

```
EN labour dispute
+base dispute
PT conflito (m) trabalhista

EN dissolution
PT dissolução (f)

EN distribution costs
+base distribution:
PT custos de distribuição

EN channels of distribution
+base distribution:
PT canais (mpl) de distribuição
```

Figura 1.2: Excerto do dicionário pós processamento.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Tal como foi dito anteriormente, o objetivo deste trabalho é tornar um dicionário financeiro mais percetível. Para tal foi feita uma análise sucinta do ficheiro de texto disponibilizado. A partir das imagens 1.1 e 1.2 é possível ter a ideia de algumas complicações que foram encontradas ao logo do trabalho.

A entrada mais simples encontrada é quando existe a palavra em inglês e de seguida a sua tradução à frente, como se pode observar na figura 2.1.

```
winding up                    liquidação (f)
```

Figura 2.1: Excerto mais simples encontrado.

Na figura 2.2, observa-se como que uma variante do caso mais simples, mas que também pode existir no caso mais complexo explicado de seguida. Neste caso, as diferentes traduções são separadas por vírgulas.

```
walkout                       abandono (m) do trabalho,
                               entrando em greve
```

Figura 2.2: Excerto com mais que uma tradução.

Já quando uma palavra é incorporada numa expressão, como se pode observar na figura 2.3, o seu processamento torna-se mais complicado. Sempre que se está perante uma entrada deste género, a palavra em questão é seguida pelos dois pontos (":") e, por vezes, tem tradução na mesma linha o que não é o caso deste exemplo. Nas linhas seguintes, onde aparece o caracter "-" significa que esse é substituído pela palavra em questão, neste caso *year*. Portanto, ficaria *year of account*, *year base* ou *base year* e *financial year*.

```
year:
 - of account    exercício (m) contábil
 - base -        ano-base (m)
 financial -     exercício (m) financeiro, ano (m)
                  contábil
```

Figura 2.3: Excerto mais complexo.

Estas são as principais entradas e as prioritárias a serem apanhadas pelo processador. Sendo assim, sabe-se que o separador é o *new line*, logo, tem-se uma entrada diferente a cada mudança de linha.

Irá ter-se uma *string* que vai sendo construída à medida que as traduções são lidas, outra que terá a palavra base e outra que terá o resto da informação do que será traduzido, se for o caso. Esta palavra base só é guardada caso seja encontrado o caracter ":". Caso contrário, não será necessário porque se tratará do caso mais simples.

2.2 Especificação de Requisitos

2.2.1 Dados

Identificador do requisito: 1
Descrição: Todas as palavras devem ter tradução.
Motivação: Consistência no dicionário.
Data: 21/05/2020
Prioridade: Média

Identificador do requisito: 3
Descrição: Os únicos caracteres especiais permitidos são ",", "(", ")", "-" e ":".
Motivação: Permitir descartar outros caracteres.
Data: 22/05/2020
Prioridade: Elevada

2.2.2 Pedidos

Identificador do requisito: 4
Descrição: A gramática deve apanhar todas as entradas possíveis.
Motivação: Transformação para um dicionário mais percetível.
Data: 22/05/2020
Prioridade: Elevada

Identificador do requisito: 5
Descrição: O analisador léxico deve conhecer acentuação.
Motivação: Conhecimento da linguagem portuguesa.
Data: 22/05/2020
Prioridade: Elevada

2.2.3 Relações

Identificador do requisito: 2
Descrição: Sempre que uma palavra tem extensão de traduções, a palavra base deve ser seguida por ":".

Motivação: Permitir que a análise seja feita corretamente.
Data: 23/05/2020
Prioridade: Elevada

Identificador do requisito: 6
Descrição: O compilador deve conhecer o caracter ”-” e substituí-lo pela palavra correspondente, exceto quando se trata de uma palavra, por exemplo, ”guarda-chuva”.
Motivação: Conhecimento do texto do dicionário.
Data: 22/05/2020
Prioridade: Elevada

Capítulo 3

Concepção/Desenho da Resolução

3.1 Estruturas de Dados

3.1.1 *Tokens* e Produções

```
union{
   char* valueBase;
   char* correspondencia;
}
%token pal palT palC ERRO
%type <valueBase> pal
%type <correspondencia> palC Smth
```

Para captar a informação necessária, decidimos definir alguns símbolos terminais, cada um responsável por captar parte da informação.

- **Token pal**: Símbolo Terminal onde guardamos o valor base da expressão a ser traduzida.
- **Token palC**: Símbolo Terminal onde se guarda o valor tanto da tradução, como das sub-expressões a serem traduzidas nos casos em que existem **'-'**.

Foi necessário guardar à parte o valor da expressão base a traduzir, pois é necessário ter acesso a esta nos cenários em que existem nas linhas seguintes expressões compostas com **'-'** e é necessário subsitituir estes *hifens* pela expressão base.

```
Dicionario  : Traducao '\n'
            | Dicionario Traducao '\n'
            | Dicionario '\n'
            ;
```

Produção principal onde se define a estrutura do input que podemos receber. Aqui definimos que o input que recebemos pode ser uma Tradução ou uma lista de Traduções. O último caso foi usado para lidar com os casos em que existem linhas em branco a meio do input.

```
Traducao : Base Correspondencia
         | Base Delim
         | Intervalo Exp Correspondencia
         | Correspondencia
         ;
```

Esta produção é a mais complexa de todas e é a mais importante, pois define os tipos de input que uma tradução pode ter.

- **Caso 1: Base Correspondencia:** Usado para captar as linhas em que a expressão a traduzir e a tradução se encontram na mesma linha e não separadas pelo delimitador **':'**. Por exemplo:

```
    advertiser              anunciante (m)
```

- **Caso 2: Base Delim:** Usado para captar dois casos diferentes. Um primeiro em que a expressão a traduzir e a tradução se encontram na mesma linha mas estão separadas por um de limitador. Exemplo:

```
    administration:         administracao (f)
```

  E segundo, o caso em que na linha apenas tem a expressão a traduzir e o delimitador **':'**.

  Exemplo:

```
    abandonment:
```

- **Caso 3: Intervalo Exp Correspondencia:** Usado para captar os casos em que a expressão é traduzir será composta em função da palavra base. Exemplo:

```
    - department                    departamento (m) de contabilidade,
        - agent             agente (m) de publicidade
```

  Como se pode ver pelos exemplos, a indentação pode variar ao logo do input. Nalguns casos pode ter vários espaços ao começar uma linha, e noutros casos pode apenas ter um. Para resolver este problema, introduzimos a produção Intervalo, que será explicada a seguir.

- **Caso 4: Correspondencia** Usado para resolver o caso em que a tradução de uma dada expressão era dividida por , sendo necessário apanhar o resto da tradução na linha seguinte. Exemplo:

```
    ADP (automatic data processing)   processamento (m) automatico de
                                    dados
```

```
Intervalo : Intervalo ' '
          | ' '
          ;
```

Produção auxiliar usada para detetar as descrepâncias no número de espaços no começo de uma linha. Permite-nos aceitar linhas de expressões que começem por um número arbitrário de espaços.

```
Exp : Smth '-' Smth
    ;

Smth : palC
     |
     ;
```

Produção utilizada para captar os casos em que a expressão a ser traduzida contém uma palavra base e uma expressão com **'-'**. Existem 3 casos possíveis onde o **'-'** pode aparecer: no ínicio da expressão, no final da expressão ou a meio da expressão. Exemplos:

```
bank -                          conta (f) bancaria
- model                         modelo (m) contabil
value - tax (VAT)   imposto (m) sobre circulacao de
```

A produção Smth pode ter um símbolo terminal palC ou então *Empty*. A possibilidade de ser Empty é fundamental para conseguirmos captar os 3 casos possíveis explicados acima.

```
Correspondencia : Intervalo palC error
                | Intervalo palC
                ;
```

Esta produção é usada para captar a tradução. O uso do **token error** é muito importante para lidar com os casos em que antes do final da linha, havia um espaço.

```
Base    : pal
        ;
```

Por último esta produção é usada para captar o valor da expressão a traduzir, que será sempre guardada no símbolo terminal pal.

3.1.2 Estados

Como já foi referido em secções anteriores, o ficheiro de input pode ter vários formatos diferentes. Para facilitar no processamento do maior número de casos possíveis, criamos vários estados no Flex: **TRADUCAO**, **EXP** e **AUX**.

```
after sales service     serviço (m) pós-venda
```

Para este exemplo de input, o tratamento é feito da seguinte forma:

- **Passo 1:** No estado INITIAL, vai captar-se a expressão base **after sales service**, dando return ao símbolo terminal *pal* e entrando no estado TRADUCAO.
- **Passo 2:** No estado TRADUCAO, irá captar-se todos os espaços um a um, dando sempre return a cada espaço que encontramos, que vão ser apanhados pela produção INTERVALO na gramática.
- **Passo 3:** Após se ter apanhado todos os espaços, vai captar-se a expressão **serviço (m) pós-venda**, dando return ao símbolo terminal palC.
- **Passo 4:** Por fim, ainda no estado TRADUCAO, vai captar-se o \n, enviando-se para o yacc e retornando ao estado inicial para processar a próxima linha.

De seguida, demonstramos a relação entre o Flex e o Yacc, e como captamos os diferentes símbolos necessários.

```
' ': [ ]

'\n': \n

pal: [^ |\n|:]+([[ ][^ |\n|:]+)*

'-': \-

palC quando usada na produção Smth: [a-zA-Z()\[\]]+([[ -][a-zA-Z()\[\]]+)*

palC quando usado na produção Correspondencia:
[a-zA-Z()\[\]|acentos|\s]+([[ -][a-zA-Z()\[\]|acentos|\s]+)*
```

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Este projeto trouxe-nos muitas dificuldades que à partida não esperavamos. A nossa inexperiência a lidar com erros de sintaxe não nos permitiu ignorar certos padrões de input, obrigando-nos então a mudar e a alterar a nossa gramática de modo a poder suportar todos os tipos de input que poderão ser dados ao nosso programa, resultando numa gramática mais complexa do que necessária.

Um dos exemplos de erros com os quais não conseguimos lidar é o facto da nossa gramática necessitar de acabar num \n, e caso o input não o faça, o programa irá parar e dar erro. Outro erro com o qual conseguimos lidar é o facto de em algumas linhas, haver um espaço antes do \n.

4.2 Testes realizados e Resultados

Vendo todos os casos possíveis, foram feitos testes para todos mostrando de seguida os resultados obtidos.

Começando com uma das entradas mais simples do ficheiro como, por exemplo:

```
after sales service     serviço (m) pós-venda
```

Obtém-se o *output*:

```
EN after sales service
PT serviço (m) pós-venda
```

Dando como *input* o seguinte texto:

```
analysis:
 breakeven -                  análise (f) do ponto de equilíbrio
 competitor -                 análise (f) da concorrência
 contribution -               análise (f) de contribuições
 cost -                       análise (f) de custos
 cost-benefit - (CBA)         análise (f) de custos e benefícios
 cost-volume-profit -         análise (f) de custo, volume e lucro
 critical path - (CPA)        análise (f) do caminho crítico
```

Obteve-se o *output*:

```
EN analysis

EN breakeven analysis
+base analysis
PT análise (f) do ponto de equilíbrio

EN competitor analysis
+base analysis
PT análise (f) da concorrência

EN contribution analysis
+base analysis
PT análise (f) de contribuições

EN cost analysis
+base analysis
PT análise (f) de custos

EN cost-benefit analysis CBA
+base analysis
PT análise (f) de custos e benefícios

EN cost-volume-profit analysis
+base analysis
PT análise (f) de custo, volume e lucro

EN critical path analysis CPA
+base analysis
PT análise (f) do caminho crítico
```

De acordo com as instruções recebidas, este exemplo funciona corretamente com a implementação feita.
Por fim, temos mais um exemplo onde a tradução ficou dividida entre duas linhas. Dando como *input* o seguinte texto:

```
allowance:
 depreciation -                abatimento (m) de depreciação, reser
                                (f) paia depreciação
```

Obteve-se o *output*:

```
EN: allowance
EN: depreciation allowance
+base: allowance
PT Tradução: abatimento (m) de depreciação, reser

PT Tradução: (f) paia depreciação
```

4.3 *Setting up* e *Compilling*

Para ter este programa a funcionar, é necessário cumprir dois comandos.
Com a existência de uma Makefile, é muito mais simples de compilar este programa:

```
> make
> ./h.exe <input.txt >output.txt
```

Posteriormente, é expectável a existência de um ficheiro TXT que contém o processamento do ficheiro inicial TXT como pretendido.

Capítulo 5

Conclusão

Neste trabalho foi criado um processador de linguagens para o *input* do ficheiro de texto dado que se tratava de um dicionário financeiro, tal como foi explicado anteriormente.

Em suma, estamos satisfeitos com o trabalho apesar das dificuldades sentidas na gestão de erros sintáticos e na falta de capacidade que tivemos de executar direito esse requisito. Também durante o trabalho tivemos de lidar com alguns conflitos *reduce/shift* ou *reduce/reduce*, o que melhorou a nossa capacidade de trabalhar e manipular gramáticas no yacc. Aprendemos também a gerar o ficheiro *y.output* para perceber onde estes conflitos estavam a ocorrer.

A satisfação final com este trabalho é notável apesar das dificuldades enfrentadas e, possivelmente no futuro, irá tentar se corrigir e melhorar o projeto elaborado.

Apêndice A

Código do Programa

A.1 Código Lex/Flex

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

int numberSpaces = 0;

%}
%option yylineno
%option noyywrap

acentos Ç|ç|á|Á|à|À|ã|Ã|â|Â|é|É|è|È|ê|Ê|í|Í|ó|Ó|ô|Ô|ú|Ú
%x TRADUCAO EXP AUX
%%
[A-Z][ ]*\n                          { printf("Começa letra %c\n",yytext[0]);}
[ ]                                  { BEGIN AUX; return (yytext[0]); }
[^ |\n|:]+([[ ][^ |\n|:]+)*          { yylval.valueBase=strdup(yytext); BEGIN TRADUCAO;
                                      return pal;}
\n                                   { return (yytext[0]);}
.                                    {;}

<AUX>[ ]                             { numberSpaces++; return (yytext[0]); }
<AUX>\-                              { BEGIN EXP; numberSpaces = 0; return (yytext[0]); }
<AUX>[a-zA-ZÇçãÃ()\[\]|acentos|\s]+([[ -][a-zA-ZçÇãÃ()\[\]|acentos|\s]+)*
                                     { if(numberSpaces > 9){
                                         yylval.correspondencia=strdup(yytext); numberSpaces = 0;
                                         BEGIN INITIAL; return palC; } else
                                         yylval.correspondencia=strdup(yytext); numberSpaces = 0;
                                         BEGIN EXP; return palC;}}

<TRADUCAO>:                          { return (yytext[0]); }
<TRADUCAO>[ ]                        { return (yytext[0]); }
```

```
<TRADUCAO>[^ |\n]+([[ ][^ |\n]+)*   { yylval.correspondencia=strdup(yytext); return palC; }
<TRADUCAO>\n                        { BEGIN INITIAL;return (yytext[0]); }
<TRADUCAO>.                         { BEGIN INITIAL; }

<EXP>\-                             { return (yytext[0]); }
<EXP>[ ]                            { ; }
<EXP>[ ][ ]                         { BEGIN TRADUCAO; }
<EXP>[a-zA-Z()\[\]]+([[ -][a-zA-Z()\[\]]+)*
                                    { yylval.correspondencia=strdup(yytext); return palC; }
<EXP>\n                             { BEGIN INITIAL; return (yytext[0]); }
<EXP>.                              {;}

%%
```

A.2 Código Yacc

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

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

char * palavra_base;

%}

%union{
   char* valueBase;
   char* correspondencia;
}
%token pal palC ERRO
%type <valueBase> pal
%type <correspondencia> palC Smth
```

```
%%
Dicionario  : Traducao '\n'
            | Dicionario Traducao '\n'
            | Dicionario '\n'
            ;

Traducao : Base Correspondencia
         | Base Delim
         | Intervalo Exp Correspondencia
         | Correspondencia
         ;


Base    : pal                                   {printf("EN %s\n",$1); palavra_base = strdup($1)
        ;

Exp : Smth '-' Smth                             {printf("EN %s %s %s\n", $1, palavra_base, $3);
                                                 printf("+base %s\n", palavra_base); }
    ;

Smth : palC                                     {$$ = strdup($1);}
     |                                          {$$ = "";}
     ;

Delim : ':' Correspondencia
      | ':'
      ;


Correspondencia : Intervalo palC error           {printf("PT %s\n",$2); printf("\n");}
                | Intervalo palC                 {printf("PT %s\n",$2); printf("\n");}
                ;

Intervalo : Intervalo ' '
          | ' '
          ;
%%


int yyerror(){
    printf("Erro Sintático ou Léxico na linha: %d\n", yylineno);
}

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

A.3 Código Makefile

```
h.exe : y.tab.o lex.yy.o
  gcc -o h.exe y.tab.o lex.yy.o -ll

y.tab.o: y.tab.c
  gcc -c y.tab.c

lex.yy.o: lex.yy.c
  gcc -c lex.yy.c

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

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

Listing A.1: Código da Makefile.