



Universidade do Minho

Mestrado Integrado em Engenharia Informática

PROCESSAMENTO DE LINGUAGENS

Trabalho Prático nº2:
Reverse Engineering dum Dicionário Financeiro

João Nunes
(a82300)

Luís Braga
(a82088)

Shahzod Yusupov
(a82617)

Braga, Portugal
28 de Junho de 2020

Resumo

O presente relatório foi elaborado no âmbito da Unidade Curricular de Processamento de Linguagens do 3ºano do Mestrado Integrado em Engenharia Informática. Este documento regista e fundamenta o trabalho e as decisões tomadas ao longo da elaboração do *Trabalho Prático nº2*.

O trabalho prático consiste, de forma sucinta, em definir uma gramática em *Yacc* (*Yet Another Compiler-Compiler*) em conjunto com *Flex* (*Fast Lexical Analyzer Generator*) de forma a ser capaz de se captar o texto de um ficheiro de entrada procedendo-se à sua ulterior filtragem e validação gramatical. O texto de entrada consiste num dicionário de termos financeiros e o objectivo do trabalho passa por realizar o *reverse engineering* deste mesmo ficheiro.

Os resultados atingidos vão de encontro com o esperado e resolvem o problema proposto de uma forma que o grupo de trabalho pensa ser eficiente e robusta. Para além do processamento solicitado no enunciado o grupo de trabalho pensou também ser pertinente gerar um ficheiro *html* com o mesmo conteúdo do ficheiro de *output* para uma melhor visualização.

Conteúdo

1	Introdução	2
1.1	Reverse Engineering dum Dicionário Financeiro	2
2	Análise e Especificação	3
2.1	Enunciado	3
2.2	Descrição do problema	4
3	Concepção/desenho da Resolução	5
3.1	Gramática	5
3.2	Expressões Regulares	8
3.2.1	Abreviaturas	8
3.2.2	Expressões <i>FLEX</i>	9
3.3	Situações de Erro	11
4	Processamento extra	13
5	Codificação e Testes	15
5.1	Alternativas, Decisões e Problemas de Implementação	15
5.2	Testes realizados e resultados	15
6	Conclusão	17
A	Código do filtro de texto (tp2.fl)	18
B	Código do filtro de texto (tp2.y)	22

Capítulo 1

Introdução

1.1 Reverse Engineering dum Dicionário Financeiro

O tema proposto enquadra-se na Unidade Curricular de Processamento de Linguagens do 3ºano do Mestrado Integrado em Engenharia Informática. Como tal, o tema abordado ao longo deste relatório consiste em utilizar *Fast Lexical Analyzer Generator* em conjunto com expressões regulares e uma gramática escrita em *Yet Another Compiler-Compiler*.

O enunciado que foi atribuído ao grupo correspondeu ao enunciado número um, onde é necessário manipular um ficheiro *txt* de maneira a converter este ficheiro num ficheiro mais tratável. Para além disso, também é proposto que se crie uma gramática adicional, bem como a deteção de erros sintáticos que ocorram no ficheiro, alertando para a linha onde existe a ocorrência do erro, permitindo mesmo assim que se continue o processamento do ficheiro.

Estrutura do Relatório

No capítulo 2 introduz-se o que é dito no enunciado e descreve-se o problema. De seguida especifica-se a implementação da solução para o problema dado.

No capítulo 3 são explicadas as estruturas de dados usadas na resolução do problema e o que estas visam resolver.

No capítulo 4 dá-se a entender o processamento extra realizado.

No capítulo 5 faz-se referência às decisões mais importantes tomadas no decorrer do trabalho, às alternativas pelas quais se podia ter enveredado e imprevistos que surgiram durante a resolução do problema. Ou seja, conta-se, de uma forma geral, como foi o trabalho.

No capítulo 6 dá-se por concluído o relatório com um resumo do que foi dito.

Acrescentam-se ainda apêndices com o código do trabalho e a bibliografia.

Capítulo 2

Análise e Especificação

2.1 Enunciado

O problema apresentado, consiste em gerar um ficheiro de *output*, onde se apresenta o *reverse engineering* de um dicionário que contém termos específicos ao mundo dos negócios e finanças. O ficheiro contém, geralmente, a seguinte estrutura:

Listing 2.1: "Breve extrato do ficheiro a processar."

```
1 dispute:
2     labour -          conflito (m) trabalhista
3 dissolution          dissolucao (f)
4 distribution:
5     - costs           custos de distribuicao
6     channels of -     canais (mpl) de distribuicao
7     physical - management controle (m) da distribuicao fisica
8 distributor          distribuidor (m), atacadista (m)
9 diversification:
10    - strategy         estrategia (f) de distribui o
11    product -          diversifica o (f) de produtos
```

Partindo deste ficheiro de input é depois necessário gerar o ficheiro inverso, tratando da melhor maneira as ocorrências de erros estruturais nas traduções, que possuirá o seguinte aspeto, para o caso anterior.

Listing 2.2: "Breve extrato do ficheiro já processado."

```
1 EN labour dispute
2 +base dispute
3 PT conflito (m) trabalhista
4
5 EN dissolution
6 PT dissolucao (f)
7
8 EN distribution costs
9 +base distribution
10 PT custos de distribuicao
11
12 EN channels of distribution
13 +base distribution
14 PT canais (mpl) de distribuicao
15
16 EN physical distribution management
17 +base distribution
```

```

18 PT controle (m) da distribuicao fisica
19
20 EN distributor
21 PT distribuidor (m)
22 PT atacadista (m)
23
24 EN diversification strategy
25 +base diversification
26 PT estrategia (f) de distribuicao
27
28 EN product diversification
29 +base diversification
30 PT diversificacao (f) de produtos

```

2.2 Descrição do problema

Como é possível observar, passando de um tipo de ficheiro para o outro ficheiro de output, existem casos simples em que a tradução pode ser feita de maneira direta, como na linha 8 do *listing* 2.1, e outros casos mais complexos que envolvem a substituição do -" pelo termo base, como por exemplo na linha 2 do *listing* 2.2. Existem também casos em que a própria tradução do termo poderá conter mais que uma definição, como por exemplo na linha 8 do *listing* 2.1. Há também outros casos que têm de ser contemplados, mas que não se encontram no extrato do ficheiro a processar, e que serão abordados posteriormente, bem como situações de erro estrutural que também acontecem no ficheiro de *input*.

Capítulo 3

Concepção/desenho da Resolução

3.1 Gramática

Após analisar tanto o ficheiro de entrada, como a saída pretendida, o grupo debruçou-se primeiro com a escrita da gramática, que segue a estrutura pretendida para o ficheiro de saída. A própria gramática, com o avançar do projeto sofreu várias alterações, mas assentou na seguinte estrutura.

1. DicFinance -> Begin Capitulos

O dicionário de finanças é composto por vários capítulos desde $A \rightarrow Z$, daí a necessidade de construir a estrutura anterior.

2. Begin -> PALAVRAS

As traduções surgem após a expressão “*__BEGIN__*”, portanto, apenas o que vem após esta palavra reservada deve ser considerado como parte do dicionário.

3. Capitulos -> Capitulos Capitulo | &

Devido a existência de vários capítulos, tal como dito anteriormente, a primeira produção contempla a lista de capítulos, e a última produção abrange o caso em que é vazia a lista de capítulos, ou seja, é o caso de paragem.

4. Capitulo -> Letra Traducoes

Cada capítulo é de seguida identificado por uma letra, e dentro de cada capítulo existe uma série de traduções que começam pela letra que identifica o capítulo.

5. Letra -> CHAR

A letra identificadora do capítulo não é nada mais nada menos que um caractere.

6. Traducoes -> Original Traducao Traducoes | &

Nas traduções, existe sempre um termo que poderá ser identificado como o termo original de onde derivam as sucessivas traduções daí, e a este termo está associada uma lista de traduções, a segunda produção identifica o caso de paragem que é vazio.

7. Traducao -> TraducaoSimples | ':' TraducaoComplexa

Uma tradução, e tal como foi identificado no capítulo anterior, poderá ter dois formatos diferentes. O primeiro formato corresponde a uma tradução simples, onde é possível diretamente associar ao termo a traduzir a sua tradução.

Listing 3.1: "Exemplo de uma tradução simples."

```
1 dissolution dissolution (f)
```

Uma tradução complexa, e tal como o nome indica, envolve mais operações. Para além do termo base existem também outros termos associados a esse mesmo termo, como por exemplo:

Listing 3.2: "Exemplo de uma tradução complexa."

```
1 dispute:
2 labour - conflito (m) trabalhista
```

8. TraducaoSimples -> Significado

Numa tradução simples, pode-se imediatamente associar o significado ao termo a traduzir.

9. TraducaoComplexa -> Significado TraducoesIncompletas | TraducoesIncompletas

Numa tradução complexa, e como se pode observar pelo *listing* 3.1, a um significado, ou seja termo base a traduzir, no exemplo utilizado do *listing* 3.1 o termo *dispute*, existe uma série de traduções incompletas associadas ao termo, que terão de ser posteriormente completadas no ':' pelo termo base da tradução.

10. TraducoesIncompletas -> TraducoesIncompletas TraducaoIncompleta | TraducaoIncompleta

As traduções incompletas associadas ao termo base podem-se estender por várias linhas, daí a necessidade de definir uma lista de traduções incompletas bem como o caso de paragem na segunda produção de apenas conter uma tradução incompleta.

11. TraducaoIncompleta -> OriginalIncompleto Significado

Cada tradução incompleta contém o símbolo terminal *OriginalIncompleto* que representa o termo que está associado ao termo base com o '-'. A este termo incompleto está também associado de seguida a sua tradução.

12. OriginalIncompleto -> PALAVRASINC

Ao símbolo não terminal incompleto está associado um símbolo terminal denominado de *PALAVRASINC* que contém a *string* representativa do termo incompleto. Por exemplo, no caso do *listing* 3.1, o *PALAVRASINC* contém:

Listing 3.3: "Exemplo do PALAVRASINC."

```
1 OriginalIncompleto = "labour - "
```

13. Original -> PALAVRAS

No símbolo não terminal original, e tal como no caso anterior, nesta produção está representada a associação com o símbolo terminal *PALAVRAS* que contém o termo original da tradução, ou seja o termo base. Mais uma vez, e utilizando o *listing* 3.1, o *PALAVRAS* contém, por exemplo:

Listing 3.4: "Exemplo do PALAVRAS."

```
1 Original = "dispute"  
2 Original = "dissolution "
```

14. Significado -> PALAVRAS

O significado é portanto a tradução dos termos, podendo eles ser incompletos ou originais. E no caso dos dois casos anteriores será, respetivamente:

Listing 3.5: "Exemplo do PALAVRAS."

```
1 Significado = "Conflito (m) trabalhista"
2 Significado = "dissolucao (f)"
```

3.2 Expressões Regulares

De forma a complementar a gramática, foi necessário desenvolver um filtro de texto em *Flex* que permitisse abranger as situações descritas na secção anterior.

Para tal, e inicialmente, o grupo optou por escrever um conjunto de expressões regulares abreviadas, de maneira a depois utilizar estas abreviaturas nas consequentes definições completas de expressões regulares.

3.2.1 Abreviaturas

Listing 3.6: "Abreviatura acentos."

```
1 acentos    \xc3[\x80-\xbf]
```

A primeira abreviatura utilizada, e devido à existência de acentos no texto do dicionário, foi preciso definir uma expressão regular que permitisse apanhar os caracteres com acentos, tendo portanto utilizado a expressão regular anterior.

Listing 3.7: "Abreviatura hifen."

```
1 hifen1     {letra}\-(\ )?({letra}|[\r\n])+
2 hifen2     {letra}(\ )?-{letra}
```

As duas abreviaturas utilizadas anteriormente fazem uso da abreviatura letra, que será explicada posteriormente, e possui o intuito de fazer o filtro de termos (palavras) que possuem o sinal '-'. Para tal, e dado o ficheiro de *input* e as situações apresentadas nesse ficheiro, foram criadas duas abreviaturas similares cuja única diferença concentra-se na localização do espaço, que poderá ocorrer antes ou depois do '-'.

Listing 3.8: "Abreviatura aspas."

```
1 aspas1     \"[^\"]+\\"
```

Outra abreviatura também necessária foi a aspas, uma vez que existem termos (palavras) que possuem aspas na sua constituição, como tal foi criada a expressão regular anterior que apenas faz *match* com as palavras circundadas pelas aspas.

Listing 3.9: "Abreviatura letra e letras."

```
1 letra      [a-zA-Z\\(\\)\\+\\/','0-9,;\\.\\_]|{acentos}
2 letras     {letra}|{hifen1}|{hifen2}|{aspas1}
```

As duas seguintes abreviaturas são a *letra* e *letras*. A primeira abreviatura possui o intuito de fazer *match* com um único caractere, podendo este ser uma letra acentuada, uma letra normal, número ou um outro tipo de caractere especial como `'.`. A *ER letras* por sua vez, conjuga a expressão regular anterior com os outros casos também discutidos anteriormente dos *hifens* e o caso especial das *aspas*.

Listing 3.10: "Abreviatura palavras e palavrasinc."

```
1 palavras      {letras}{1,}(\ {letras}{1,})*
2 palavrasinc   ((\ {0,1}{letras}{1,})|(\ {0,2}(\ -(\ {0,1})))))*
```

As últimas abreviaturas, são referentes às próprias palavras que não são nada menos do que uma conjunção de *letras*, pelo menos uma, sendo que as palavras, no caso de serem palavras conjugadas, são separadas por um espaço. Por sua vez a *palavrasinc*, ou seja palavras incompletas, ocorrem no caso onde se pretende fazer *match* com o termo incompleto associado ao termo original. Por exemplo, e no caso do *listing 2.1* na linha 2 o *palavrasinc* possui o intuito de fazer *match* com *labour* -. Para tal, as palavras incompletas tanto podem aparecer com um espaço ou nenhum no início, sendo de seguida composta por uma conjunção de caracteres. Ou, as palavras incompletas podem também aparecer com 0 até 2 espaços no início, com um hífen e de seguida mais espaços.

3.2.2 Expressões *FLEX*

Listing 3.11: "Expressão regular começo do ficheiro."

```
1 ( __BEGIN__ ) [\r\n]+
```

A primeira expressão regular definida, para além das abreviaturas, foi referente à *tag* de início do dicionário. Ora, como o processamento do ficheiro só deverá ser efetuado partindo dessa *tag*, foi necessário capturar esta mesma de maneira a apenas executar as ações semânticas associadas à *ER* caso a *tag* `__BEGIN__` tenha sido capturada. Para além disso, mal esta *tag* seja capturada pela *ER*, é também criado o ficheiro de *logs* de erros, que são guardados num ficheiro a linha, código de erro e texto do erro.

Listing 3.12: "Expressão regular para a letra do capítulo."

```
1 [A-Za-z]\ *[\r\n]+
```

A expressão regular acima representada tem como finalidade capturar a letra que marca o início de Capítulo. É de relembrar que um capítulo, neste caso específico, de um dicionário começa por indicar qual é a letra pelas quais as palavras do capítulo vão passar.

Listing 3.13: "Expressão que captura mudanças de linha imprevistas."

```
1 [\r\n]
```

A expressão regular representada na figura tem o intuito de realizar o *garbage collecting* de mudanças de linha imprevistas.

Listing 3.14: "Expressão que marca início de uma tradução complexa."

```
1 [:]
```

Com a captura do símbolo ':', isto indicará então a entrada para uma tradução complexa, para tal, foi também criada uma *flag* que indica a entrada para uma tradução com o nome de *zonaComplexa*. Como tal, esta *flag* é colocada a 1, e o caracter é devolvido para ser tratado no *yacc*.

Listing 3.15: "Expressão que marca início de uma tradução complexa."

```
1 {palavras}\- /([\r\n]+\ {1,4}{palavrasinc}
```

Esta expressão regular foi desenvolvida para casos específicos em que o hífen que deve ser substituído por a palavra dita "base" se encontra imediatamente seguido de uma mudança de linha.

Listing 3.16: "ER para apanhar expressões simples multilinha."

```
1 ^{palavras}([\r\n]+\ {1,2}{palavras})*
```

A expressão anterior foi concebida de maneira a tratar dos casos em que as expressões simples se estendem por várias linhas. Depois de capturado o texto são tratados alguns aspectos como quebras de linha, que são substituídos por espaço, e os espaços múltiplos que são reduzidos apenas a um espaço. Sendo feito em fase ulterior o *return* de PALAVRAS.

Listing 3.17: "ER para apanhar expressões simples multilinha."

```
1 ^{palavras}([\r\n]+\ {1,2}{palavras})*
```

A expressão anterior foi concebida de maneira a tratar dos casos em que as expressões simples se estendem por várias linhas.

Listing 3.18: "ER para apanhar Significado."

```
1 {palavras}
```

Esta expressão bastante simples recolhe todas as PALAVRAS que se estendem apenas por uma linha.

Listing 3.19: "ER para apanhar expressões complexa multilinha."

```
1 {palavras}([\r\n]+[\ \t]{8,}){palavras}+)
```

De maneira análoga à expressão regular anterior, a expressão regular supracitada funciona de maneira semelhante à anterior mas apenas para o caso de ser multi linha. Onde é feito um tratamento semelhante ao *listing* 3.15, no sentido em que são tratadas as quebras de linha e os espaços múltiplos são também condensados. O valor do texto capturado é colocado na *union* e é de seguida retornado o símbolo terminal *PALAVRAS*.

Listing 3.20: "ER para apanhar expressões simples multilinha."

```
1 ^\ {1,4}{palavrasinc}([\r\n]+\ {1,4}{palavrasinc})*
```

Uma das expressões mais importantes do deste projecto é a expressão acima. Esta ER filtra TraducoesIncompletas, mas especificamente, o OriginalIncompleto, sendo por vezes multi-linha. Depois de capturado o texto apenas se retiram as mudanças de linha de forma a ser possível tratar o OriginalIncompleto e substituir o "-" pela palavras base.

3.3 Situações de Erro

Visto que no ficheiro fornecido para *input* existem várias gafes, o grupo de trabalho pensou ser pertinente classificá-las e documentá-las de forma a gerar um ficheiro de *logs de erro*, que foi algo que o grupo achou pertinente de maneira a melhor perceber e no futuro adaptar o filtro de texto para mais casos, gerado após cada processamento tenha mais sentido, relevância e utilidade. Tendo isto em conta, neste capítulo apresentar-se-ão os erros captados pelo grupo de trabalho.

O ficheiro de *logs* contém informação acerca da linha, texto e código do erro, o nome do ficheiro de *logs* é único no sentido em que é gerado na hora de acordo com a data e hora atual.














 ERRORLOG_2020-6-22T20_2_17.txt	22/06/2020 20:02	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T20_30_2.txt	22/06/2020 20:30	Documento de tex...	28 KB
 ERRORLOG_2020-6-22T20_30_37.txt	22/06/2020 20:30	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T21_21_13.txt	22/06/2020 21:21	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T21_41_52.txt	22/06/2020 21:41	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T21_51_31.txt	22/06/2020 21:51	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T21_53_3.txt	22/06/2020 21:53	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T21_59_52.txt	22/06/2020 21:59	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T22_4_31.txt	22/06/2020 22:04	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T22_9_24.txt	22/06/2020 22:09	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T22_59_26.txt	22/06/2020 22:59	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T23_0_16.txt	22/06/2020 23:00	Documento de tex...	29 KB
 ERRORLOG_2020-6-22T23_2_26.txt	22/06/2020 23:02	Documento de tex...	29 KB

Figura 3.1: Pasta com os logs de erro.

Foram desenvolvidas expressões regulares adicionais, de maneira a conseguir captar e ignorar o erro, permitindo que o processamento do ficheiro continuasse sem qualquer problema.

Listing 3.21: "Conteúdo do ficheiro de logs."

```
1 {ERRO NA LINHA:885
2 COM O TEXTO:vari veis
3 CODIGO DE ERRO:2}
4
5 {ERRO NA LINHA:886
6 COM O TEXTO:(mpl)
7 CODIGO DE ERRO:2}
```

A tabela abaixo visa mostrar, de uma forma compacta, a descrição do erro, um exemplo no ficheiro fonte, o seu respectivo código no ficheiro de *logs* e o local onde é tratado no ficheiro *LEX*.

Tabela de Erros			
Descrição	Exemplo	Código	Referência no Lex
Existe uma tradução complexa sem os ':' que a distinguem, seguida de uma mudança de linha, começando essa por uma palavra e não por hífen.	Da linha 1037 à 1038	1	Linha 50
Quando é captado um Significado não podem ser captados mais significados à sua direita.	Da linha 1850 à 1851	2	Sempre que é mencionado o <i>bool skipline</i> esta situação está a ser tratada.
Quando dentro de uma TraducaoIncompleta não existem OriginaisIncompletos.	Da linha 137 à 139	3	Linha 74
Quando ocorre um "Original:", contudo existem espaços antes do Original.	Linha 3389	4	Linha 80
Existe um "Original:", contudo de seguida não existem palavras por completar com a palavras base.	Linha 1920	5	Linha 85
Quando ocorre uma PALAVRASINC sem o respectivo espaço antes.	Linha 589	6	Linha 130
PALAVRASINC sem a respectiva tradução.	Linha 944	7	Linha 137
PALAVRA em início de linha sem nenhuma tradução.	Linha 546	8	Linha 144
Surgem OriginalIncompleto sem antes ter surgido o "Original:".	Linha 3386	9	Linha 125

Capítulo 4

Processamento extra

Finda a elaboração do objectivo primário deste projecto, decidiu-se que seria pertinente conceber, em adição ao que já foi construído, uma página *html* para melhor apresentar os resultados obtidos. Na figura abaixo demonstra-se a parte inicial da *interface* dessa mesma página *html*.

Reverse Engineering dum Dicionário Financeiro	
Original	Traduções
ADP (automatic data processing)	• processamento (m) automático de dados
absenteeism	• absenteísmo (m)
absorption costing	• custeio (f) de absorção
product abandonment	• retirada (f) de um produto
above par	• com ágio • acima da paridade
acceleration clause	• cláusula (f) de aceleração
acceptance	• aceitação (f)
brand acceptance	• aceitação (f) de uma marca

Figura 4.1: Interface do ficheiro html gerado.

Para o efeito "injectaram-se" escritas para ficheiro *html* nos mesmo locais/funções onde se escreve para o ficheiro de saída *txt*.

Na fase inicial do processamento, faz-se a iniciação do ficheiro *html*, aí escreve-se o início do *html* juntamente com o *css* que vai dar o aspecto que a tabela tem na figura acima. Quando surge uma *TraducaoSimples* ou uma *TraducaoComplexa* escreve-se a palavra original e seguida da sua lista de traduções. Aquando do fim do processamento fecha-se a tabela e as restantes *tags html*. O nome dado a este ficheiro é, de forma

análoga ao ficheiro *txt*, o nome do ficheiro dado como input como acréscimo de "SAIDA.html".

Capítulo 5

Codificação e Testes

5.1 Alternativas, Decisões e Problemas de Implementação

Como não poderia deixar de ser: inerente à produção de um trabalho é a tomada de decisões e, infelizmente, o aparecimento de problemas. Aqui falar-se-ão das decisões mais importantes tomadas ao longo do desenvolvimento do projecto.

A primeira decisão do projecto foi compreender o ficheiro de entrada de forma a ser-se capaz de planear o tomar os primeiros passos no seu processamento. Verificou-se que possuía um formato simples e uma gramática não muito complexa, não obstante, existem várias situações que fogem à regra (situações de erro) que se teriam de considerar e que sobem a dificuldade do projecto.

Após a análise do ficheiro de entrada, começou-se por elaborar as expressões regulares em conjunto com a gramática. Foram-se construindo versões cada vez mais acertadas, completas e que englobassem cada vez mais linhas do ficheiro de entrada, fazendo-se logo de início algum tratamento do *yytext*, isto é, eliminou-se espaços a mais e retiraram-se mudanças de linha. Foi precisamente nesta fase que o grupo encontrou as maiores dificuldades, pois, haviam situações de erro difíceis de se tratar e muitas vezes de se conjugar. Quando se atingiu o final do ficheiro acertaram-se as expressões regulares e deu-se por terminada esta fase. Numa terceira fase, construíram-se as acções no ficheiro *Yacc* que dão origem ao *output* final, ou seja, ao ficheiro de texto com o *reverse engineering* do dicionário financeiro e também o ficheiro com o que é descartado (erros) do ficheiro inicial.

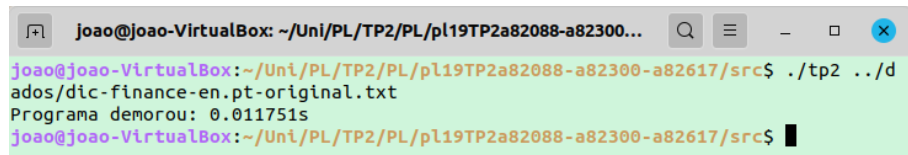
A quarta fase é uma fase extra, na medida em que se elaborou a saída para html.

No que diz respeito a alternativas, o colectivo de trabalho julga que a elaboração não poderia sair muito deste formato visto que o ficheiro de entrada não o permite. Poderiam sim, escrever-se as ERs de outras formas.

5.2 Testes realizados e resultados

De forma a avaliar o desempenho do filtro de texto concebido bem como a gramática definida pelo grupo, no ficheiro *yacc* na *main* foi medido o tempo desde o início desde o início do *parse* do ficheiro de *input* até ao fim da escrita para os ficheiros correspondentes, *txt* e *html*.

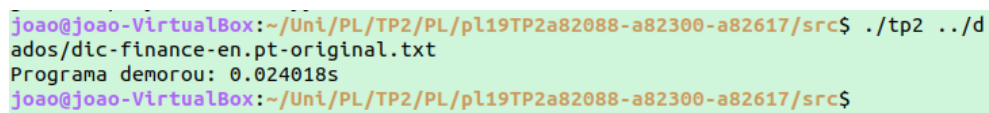
O ficheiro de *input* disponibilizado possui cerca de 3700 linhas para processar. Com esse ficheiro, foi possível obter o seguinte tempo de execução do programa.

A terminal window titled 'joao@joao-VirtualBox: ~/Uni/PL/TP2/PL/pl19TP2a82088-a82300...' with search, menu, and window control icons. The prompt is 'joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src\$'. The command './tp2 ../dados/dic-finance-en.pt-original.txt' is entered. The output is 'Programa demorou: 0.011751s'. The prompt is then 'joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src\$' followed by a cursor.

```
joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src$ ./tp2 ../dados/dic-finance-en.pt-original.txt
Programa demorou: 0.011751s
joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src$
```

Figura 5.1: Tempo de processamento do ficheiro de input.

De maneira a testar num ficheiro um pouco maior, o ficheiro anterior foi duplicado em tamanho, ao copiar todos os capítulos e colar novamente no final do ficheiro, o que gerou um novo ficheiro de *input* de 7400 linhas, sendo que foi possível obter os seguintes tempos. O tempo de processamento também duplicou de um ficheiro para outro passando de 0.01 segundos para 0.02 segundos.

A terminal window with the same title and icons as Figure 5.1. The prompt is 'joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src\$'. The command './tp2 ../dados/dic-finance-en.pt-original.txt' is entered. The output is 'Programa demorou: 0.024018s'. The prompt is then 'joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src\$' followed by a cursor.

```
joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src$ ./tp2 ../dados/dic-finance-en.pt-original.txt
Programa demorou: 0.024018s
joao@joao-VirtualBox:~/Uni/PL/TP2/PL/pl19TP2a82088-a82300-a82617/src$
```

Figura 5.2: Tempo de processamento do ficheiro de input com tamanho duplicado.

Capítulo 6

Conclusão

Ao longo da concepção deste primeiro trabalho prático o grupo deparou-se com alguns *stumbling blocks*, no tocante à estrutura do ficheiro de *input* uma vez que este possuía bastantes casos singulares que não poderiam ser apanhados de uma maneira geral, estes casos tinham de ser tratados como erros e consequentemente tinha de ser gerada uma *ER* para tratar desses casos, o que colocou um elevado nível de complexidade ao gerir estes erros, uma vez que não havia acesso a, por exemplo, contextos de maneira a ajudar no tratamento destes casos.

Não obstante, o trabalho revelou-se como sendo crucial para aprofundar os conhecimentos no que toca à utilização e criação das expressões regulares, bem como um aperfeiçoamento da linguagem *FLEX*. Para além disso, o projeto também se revelou como sendo bastante útil na consolidação dos conhecimentos relativos ao desenho, e implementação de uma gramática em *YACC*.

Como tal o grupo de trabalho faz uma avaliação positiva da solução concebida para o segundo trabalho prático, sendo que o grupo considera que esta solução atende e responde a todos os tópicos necessários, sendo que com o restante tempo disponível o grupo optou por melhorar a solução ao desenvolver uma interface por via *web* através da geração de um ficheiro *html*, bem como *logs* de erros.

Apêndice A

Código do filtro de texto (tp2.fl)

```
1 %{
2 #include "y.tab.h"
3 #include <ctype.h>
4 #include <stdio.h>
5 #include <time.h>
6
7 void strip_linebreak(char* str);
8 void strip_extra_spaces(char* str);
9 void createLogFile();
10 void printError(int cod);
11
12 int beginread = 0;
13 int skipline = 0;
14 int zonacomplexa = 0;
15
16 FILE *exceptions;
17 %}
18
19 %option yylineno
20
21acentos      \xc3[\x80-\xbf]
22hifen1       {letra}\-(\ )?({letra}|[\r\n])+
23hifen2       {letra}(\ )?-\{letra}
24aspas1       \',\^[^\,']*+\',
25letra        [a-zA-Z\(\)\+\,\,\'0-9,;\.\_]|{acentos}
26letras       {letra}|{hifen1}|{hifen2}|{aspas1}
27palavras     {letras}{1,}(\ {letras}{1,})*
28palavrasinc  ((\ {0,1}{letras}{1,})|(\ {0,2}(\-(\ {0,1}))))*
29
30%%
31
32(__BEGIN__)[\r\n]+      { beginread = 1; createLogFile(); return PALAVRAS; }
33
34[A-Za-z]\ *[\r\n]+     {
35                        if(beginread){
36                            yylval.cvalue = *yytext;
37                            return CHAR;
38                        }
39                    }
40
41[\r\n] ;
42
43[:]                {
44                    if(beginread){
45                        zonacomplexa = 1;
46                        return *yytext;
47                    }
48                }
49
```

```

50 ^{palavras}[\r\n]+\ {1,2}{palavras}{palavrasinc} { if(beginread){ skipline = 1; printError(1); } }
51
52 {palavras}\-/[r\n]+\ {1,4}{palavrasinc} {
53     if(beginread){
54         if(!skipline){
55             skipline = 1;
56             yylval.svalue = strdup(yytext);
57             return PALAVRAS;
58         }
59         else printError(2);
60     }
61 }
62
63 ^{palavras}([\r\n]+\ {1,2}{palavras})* {
64     if(beginread){
65         strip_linebreak(yytext);
66         strip_extra_spaces(yytext);
67         yylval.svalue = strdup(yytext);
68         zonacomplexa = 0;
69         skipline = 0;
70         return PALAVRAS;
71     }
72 }
73
74 ^{palavras}\:[r\n]+/{palavras} {
75     if(beginread){
76         printError(3);
77     }
78 }
79
80 ^\ {1,2}{palavras}\:(\ +{palavras}\ +{palavras})?[\r\n]+/\ {1,4}{palavrasinc} {
81     if(beginread)
82     printError(4);
83 }
84
85 ^{palavras}\:{palavras}\ +{palavras}[\r\n]+/{palavras} {
86     if(beginread)
87     printError(5);
88 }
89
90 {palavras} {
91     if(beginread){
92         if(!skipline){
93             skipline = 1;
94             yylval.svalue = strdup(yytext);
95             return PALAVRAS;
96         }
97         else printError(2);
98     }
99 }
100
101 {palavras}([\r\n]+[\ \t]{8,}{palavras})+ {
102     if(beginread){
103         if(!skipline){
104             skipline = 1;
105             strip_linebreak(yytext);
106             strip_extra_spaces(yytext);
107             yylval.svalue = strdup(yytext); return PALAVRAS;
108         }
109         else printError(2);
110     }
111 }
112
113
114
115 ^\ {1,4}{palavrasinc}([\r\n]+\ {1,4}{palavrasinc})* {
116     if(beginread){
117         if(zonacomplexa){

```

```

118                                     skipline = 0;
119                                     strip_linebreak(yytext);
120                                     yylval.svalue = strdup(yytext);
121                                     return PALAVRASINC;
122                                     }
123                                     else{
124                                     printError(10);
125                                     skipline = 1;
126                                     }
127                                     }
128                                     }
129
130 {palavrasinc} {
131                                     if(beginread){
132                                     skipline = 1;
133                                     printError(6);
134                                     }
135                                     }
136
137 ^\ {1,4}{palavrasinc}[\r\n]+/{palavras} {
138                                     if(beginread)
139                                     printError(7);
140                                     }
141
142 \ {palavras}[\r\n]+ {
143                                     if(beginread)
144                                     printError(8);
145                                     }
146
147 ^{palavras}[\r\n]+ {
148                                     if(beginread)
149                                     printError(9);
150                                     }
151
152
153 \ + ;
154
155 . { if(beginread){ return ERRO; } }
156
157 %%
158
159 void printError(int cod){
160     fprintf(exception, "{ERRO NA LINHA:%d\n", yylineno);
161     fprintf(exception, "COM O TEXTO:%s\n", yytext);
162     fprintf(exception, "CODIGO DE ERRO:%d\n\n", cod);
163 }
164
165
166 void strip_linebreak(char* str) {
167     int i;
168     for(i=0; str[i]; ++i)
169         if(i > 0 && str[i] == '\n' && str[i-1] == '\r'){
170             str[i] = ' ';
171             str[i-1] = ' ';
172         }
173     if(str[i] == '\n')
174         str[i] = ' ';
175 }
176
177 void strip_extra_spaces(char* str) {
178     int i, x;
179     for(i=x=0; str[i]; ++i)
180         if(!isspace(str[i]) || (i > 0 && !isspace(str[i-1])))
181             str[x++] = str[i];
182     str[x] = '\0';
183 }
184
185

```

```

186 void createLogFile(){
187
188     time_t t = time(NULL);
189     struct tm tm = *localtime(&t);
190     char nomeFich[100];
191     sprintf(nomeFich,"%d-%d-%dT%d_%d_%d", tm.tm_year + 1900, tm.tm_mon +1, tm.tm_mday, tm.
tm_hour, tm.tm_min, tm.tm_sec);
192
193     char* file = malloc(50);
194
195     strcpy(file, "../errorLogs/");
196     strcat(file, "ERRORLOG_");
197     strcat(file, nomeFich);
198     strcat(file, ".txt");
199
200     exceptions = fopen(file, "w");
201 }

```

Apêndice B

Código do filtro de texto (tp2.y)

```
1  %{
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <unistd.h>
7
8  extern int yylex();
9  extern int yylineno;
10 extern char* yytext;
11 extern int yy_flex_debug;
12 extern FILE *yyin;
13
14 void yyerror();
15 void erroSem(char*);
16 void printTradSimples(char* sig);
17 void splitSignificado(char* sig, char* simb);
18 int findSep(char * sig, char sep);
19 char *replaceWord(const char *s, const char *oldW,
20                  const char *newW);
21 void printTradIncompleto(char* orig, char* sig);
22 void help();
23 void initHTML(char* nomefic, int len);
24 void closeHTML();
25
26 char* originalstr;
27 FILE *yyout;
28 FILE *yyoutHTML;
29 %}
30
31 %union{
32     char* svalue;
33     char cvalue;
34 }
35
36 %token ERRO
37 %token <svalue> PALAVRAS
38 %token <svalue> PALAVRASINC
39 %token <cvalue> CHAR
40 %type <svalue> Original
41 %type <svalue> Significado
42 %type <svalue> OriginalIncompleto
43 %type <cvalue> Letra
44 %%
45
46 DicFinance
47     : Begin Capitulos
48     ;
49
```



```

50 Begin
51   : PALAVRAS
52   ;
53
54 Capitulo
55   : Capitulo Capitulo
56   |
57   ;
58
59 Capitulo
60   : Letra { fprintf(yyout, "%c\n\n", $1); } Traducoes
61   ;
62
63 Letra
64   : CHAR { $$ = $1; }
65   ;
66
67 Traducoes
68   : Original { originalstr = strdup($1); } Traducao Traducoes
69   |
70   ;
71
72 Traducao
73   : TraducaoSimples
74   | ':' TraducaoComplexa
75   ;
76
77 TraducaoSimples
78   : Significado { printTradSimples($1); }
79   ;
80
81 TraducaoComplexa
82   : Significado
83   { fprintf(yyout, "EN %s\n", originalstr);
84     fprintf(yyout, "PT %s\n\n", $1);
85     fprintf(yyoutHTML, "<tr><td>%s</td><td><ul><li>%s</li></ul></td></tr>", originalstr, $1); }
86   TraducoesIncompletas
87   | TraducoesIncompletas
88   ;
89
90 TraducoesIncompletas
91   : TraducoesIncompletas TraducaoIncompleta
92   | TraducaoIncompleta
93   ;
94
95 TraducaoIncompleta
96   : OriginalIncompleto Significado { printTradIncompleto($1, $2); }
97   ;
98
99 OriginalIncompleto
100  : PALAVRASINC { $$ = $1; }
101  ;
102
103 Original
104   : PALAVRAS { $$ = $1; }
105   ;
106
107 Significado
108   : PALAVRAS { $$ = $1; }
109   ;
110 %%
111 int main(int argc, char* argv[]) {
112
113     if(argc < 2){
114         help();
115         return 0;
116     }

```

```

117     if(access(argv[1], F_OK) != -1){
118
119         if(access(argv[1], R_OK) != -1){
120
121             int len = strlen(argv[1]);
122
123             // definir nome do ficheiro final
124             char* nome = (char *) malloc(9 + len + 1);
125             strcpy(nome, argv[1]);
126             nome[len-4] = '\0';
127             strcat(nome, "SAIDA.txt");
128
129             yyout = fopen(nome, "w");
130
131             // abrir o ficheiro html
132             initHTML(argv[1], len);
133
134             // abrir o ficheiro a ler
135             yyin = fopen(argv[1], "r");
136
137             clock_t start = clock();
138
139             // inicializar a leitura
140             //yy_flex_debug = 1;
141             yyparse();
142
143             closeHTML();
144             fclose(yyin);
145             fclose(yyout);
146
147             clock_t end = clock();
148             float seconds = (float)(end - start) / CLOCKS_PER_SEC;
149             printf("Programa demorou: %fs\n", seconds);
150
151             free(nome);
152         }
153         else{
154             printf("N o possui permiss o de leitura sobre o ficheiro fornecido!\n");
155         }
156     }
157     else{
158         printf("O ficheiro dado como argumento n o existe !\n");
159     }
160 }
161
162 return 0;
163 }
164
165
166 void help(){
167
168     printf("\n");
169     printf("*****Reverse Engineering dum Dicionario Financeiro\n");
170     printf("*****\n");
171     printf("**\n");
172     printf("**\n");
173     printf("*****HELP\n");
174     printf("**\n");
175     printf("** Utiliza o:\n");
176     printf("** 1- make\n");

```

```

177 printf("***                2- ./tp2 [nome do ficheiro a processar]
    **\n");
178 printf("***
    **\n");
179 printf("***      Notas:
    **\n");
180 printf("***      O ficheiro resultante vai para a mesma pasta com o mesmo
    **\n");
181 printf("***      nome do original, apenas com a modifica o de ter
    **\n");
182 printf("***      \"SAIDA.txt\" no final.
    **\n");
183 printf("***
    **\n");
184 printf("*****HELP
    *****\n");
185
186 }
187
188
189 void yyerror(){
190     printf("Erro Sint tico ou L xico na linha: %d, com o texto: %s\n", yylineno, yytext);
191 }
192
193 void printTradSimples(char* sig){
194     fprintf(yyoutHTML, "<tr><td>%s</td><td><ul>", originalstr);
195
196     fprintf(yyout, "EN %s\n", originalstr);
197
198     if(findSep(sig, ','))
199         splitSignificado(sig, ",");
200     else
201         splitSignificado(sig, ";");
202
203     fprintf(yyoutHTML, "%s", "</ul></td></tr>");
204
205 }
206
207 void printTradIncompleto(char* orig, char* sig){
208
209     char *result = NULL;
210     char *comespacos = (char *) malloc(2 + strlen(originalstr) + 1);
211
212     strcpy(comespacos, " ");
213     strcat(comespacos, originalstr);
214     strcat(comespacos, " ");
215
216     result = replaceWord(orig, " - ", comespacos);
217
218     fprintf(yyout, "EN %s\n", result);
219     fprintf(yyout, "+base %s\n", originalstr);
220
221     fprintf(yyoutHTML, "<tr><td>%s</td><td><ul>", result);
222
223     free(result);
224     free(comespacos);
225
226     if(findSep(sig, ','))
227         splitSignificado(sig, ",");
228     else
229         splitSignificado(sig, ";");
230
231     fprintf(yyoutHTML, "%s", "</ul></td></tr>");
232 }
233
234 void splitSignificado(char* sig, char* simb){
235
236     char * token = strtok(sig, simb);

```

```

237
238 while( token != NULL ) {
239     fprintf(yyoutHTML, "<li>%s</li>", token);
240     fprintf(yyout, "PT %s\n", token );
241     token = strtok(NULL, simb);
242 }
243 fprintf(yyout, "%s", "\n");
244
245 }
246
247 int findSep(char * sig, char sep){
248
249     int result = 0;
250
251     int len = strlen(sig);
252
253     for(int i = 0; i < len && result!=1; i++)
254         if(sig[i] == sep) result = 1;
255
256
257     return result;
258 }
259
260
261
262 char *replaceWord(const char *s, const char *oldW,
263
264                                     const char *newW)
265 {
266     char *result;
267     int i, cnt = 0;
268     int newWlen = strlen(newW);
269     int oldWlen = strlen(oldW);
270
271     for (i = 0; s[i] != '\0'; i++)
272     {
273         if (strstr(&s[i], oldW) == &s[i])
274         {
275             cnt++;
276             i += oldWlen - 1;
277         }
278     }
279
280     result = (char *)malloc(i + cnt * (newWlen - oldWlen) + 1);
281
282     i = 0;
283     while (*s)
284     {
285         if (strstr(s, oldW) == s)
286         {
287             strcpy(&result[i], newW);
288             i += newWlen;
289             s += oldWlen;
290         }
291         else
292             result[i++] = *s++;
293     }
294
295     result[i] = '\0';
296     return result;
297 }
298
299
300
301 void initHTML(char* nomefic, int len){
302
303     char* nome = (char *) malloc(10 + len + 1);
304     strcpy(nome, nomefic);

```

```

305     nome[len-4] = '\0';
306     strcat(nome, "SAIDA.html");
307
308     yyoutHTML = fopen(nome, "w");
309
310     fprintf(yyoutHTML,"%s", "<!DOCTYPE html><html><head><style>#erros {font-family: 'Trebuchet MS',
    Arial, Helvetica, sans-serif;border-collapse: collapse;width: 100%;}#erros td, #customers th {
    border: 1px solid#ddd;padding: 8px;}#erros tr:nth-child(even){background-color:#f2f2f2;}      #
    erros tr:hover {background-color: #ddd;}#erros th {padding-top: 12px;padding-bottom: 12px;text-
    align: left;background-color: Gray;color: white;}</style></head><body><center><h1>Reverse
    Engineering dum Dicion rio Financeiro</h1></center><table id='erros'><tr><th>Original</th><th>
    Tradu es </th></tr>");
311 }
312
313 void closeHTML(){
314     fprintf(yyoutHTML, "%s", "</table></body></html>");
315     fclose(yyoutHTML);
316 }
317 }

```

Bibliografia

- [1] Dicionário de finanças,
https://natura.di.uminho.pt/~jj/pl-20/TP2/dicionario_financas
- [2] C File Handling,
<https://www.programiz.com/c-programming/c-file-input-output>
- [3] Quora - Lex functions and variables,
<https://www.quora.com/What-is-the-function-of-yylex-yyin-yyout-and-fclose-yyout-in-LEX>
- [4] Flex Options,
http://dinosaur.compilertools.net/flex/flex_17.html
- [5] YACC Options,
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.bpxa500/yacc.htm
- [6] Stackoverflow,
<https://stackoverflow.com/>
- [7] Flex and YACC Pratices,
<https://www.epaperpress.com/lexandyacc/pr1.html>