LexMan: um Segmentador e Analisador Morfológico com Transdutores

Alexandre Manuel Fajardo Vicente

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Professor Doutor João Emílio Segurado Pavão Martins
Orientador: Professor Doutor Nuno João Neves Mamede
Co-Orientador: Professor Doutor Jorge Manuel Evangelista Baptista
Vogal: Professor Doutor João Carlos Serrenho Dias Pereira

Junho 2013

Agradecimentos

Gostaria de agradecer o apoio, a constante disponibilidade e a dedicação do meu orientador, Professor Nuno Mamede, e do meu co-orientador, Professor Jorge Baptista.

Gostaria também de agradecer ao Cláudio Diniz pela sua contribuição no desenvolvimento deste trabalho, pela sua disponibilidade e pelas reuniões, sempre produtivas, onde discutimos ideias e soluções.

Um agradecimento à minha família, em especial aos meus pais, avós e à minha irmã, por me terem sempre apoiado ao longo do meu percurso.

Aos meus colegas Filipe Carapinha, João Marques, Rúben Raposo, Tiago Freitas e Tiago Travanca, muito obrigado pelos momentos de ajuda e de trabalho que nós partilhámos.

Lisboa, 14 de Junho de 2013
Alexandre Vicente

Resumo

Na arquitetura inicial da cadeia de Processamento de Língua Natural (PLN) do L^2F, STRING, a segmentação e a análise morfológica eram realizadas por 2 módulos distintos: a segmentação era feita através de expressões regulares enquanto a análise morfológica usava um transdutor para atribuir as etiquetas morfossintáticas aos segmentos.

Este trabalho permitiu, usando transdutores, juntar a segmentação e a análise morfológica num único módulo, o LexMan. Esta mudança possibilitou a transferência das regras de junção de segmentos independentes de contexto, que estavam implementadas no módulo de desambiguação morfossintático da cadeia, o RuDriCo, para o módulo LexMan. A informação usada na geração do transdutor do dicionário foi ainda complementada com informação derivacional, tendo passado a ser possível também o reconhecimento de palavras derivadas por prefixação.

Foram concebidas, construídas e avaliadas duas arquiteturas distintas, comparando-as com a arquitetura inicial, tendo-se concluído que as novas soluções eram mais eficientes no processamento de textos de grandes dimensões. Considerando os textos de maiores dimensões avaliados, a arquitetura baseada na operação *Prune* foi 8.63% mais rápida do que a baseada na operação *ShortestPath*, e 69.6% mais rápida do que a arquitetura inicial.

A melhor solução complementa o transdutor do dicionário com informação derivacional sobre prefixos, permitindo aumentar a cobertura das palavras identificadas e etiquetadas pelo LexMan. A integração deste módulo originou, no processamento dos mesmos textos, uma perda da velocidade de desempenho de 15.56%. Essa perda foi atenuada após terem sido removidas as palavras prefixadas, entretanto tornadas redundantes, do dicionário de lemas.

Abstract

In the L^2F's Natural Language Processing (NLP) chain, STRING, the tokenization and morphological analysis were performed by two independent modules. The tokenization was performed by regular expressions and the morphological analysis was made using a transducer to assign the morpho-syntactical (part-of-speech) tags to the tokens.

This work allowed the union of the tokenization module and the morphological analysis in a single module, LexMan, using transducers. With this change, it was possible to transfer morpho-syntactic, context-independent, joining rules (for compound identification), previously implemented in the chain's morphosyntactic disambiguator, RuDriCo to the LexMan module. The information used in the generation of the dictionary transducer can now be complemented also by derivational information, making possible to recognize prefixed-derived words, particularly neologisms.

Two architectures were created and evaluated, comparing them with the initial architecture. The two new architectures proved to be more efficient in the processing of large-sized texts. Considering the largest texts submitted to evaluation, the Prune-based architecture was 8.63% faster than the ShortestPath-based one, and 69.6% faster that the initial architecture.

It was in this faster, Prune-based architecture that the prefixes' module was integrated. This made possible to extend the coverage of the system lexical resources. The integration of this new module resulted in 15.56% increase in the system's performance time, considering the same evaluation texts. That loss in performance was attenuated after the removal of the now redundant prefixed words from the dictionary of lemmas.

Palavras-Chave
Keywords

Palavras Chave

Processamento de Língua Natural
Transdutores
Segmentação
Análise Morfológica

Keywords

Natural Language Processing
Transducers
Tokenization
Morphological Analysis

Índice

1 Introdução **1**

1.1 Objetivos do Trabalho . . . 3

1.2 Estrutura do Documento . . . 4

2 Trabalho Relacionado **5**

2.1 Segmentação e Análise Morfológica . . . 5

2.2 SMORPH . . . 6

2.2.1 Maiúsculas, minúsculas e diacríticos . . . 8

2.2.2 Prefixos e sufixos . . . 9

2.2.3 Expressões polilexicais . . . 9

2.2.4 Ambiguidade . . . 10

2.2.5 Hifenização . . . 10

2.2.6 Geração . . . 10

2.2.7 Arquitetura do Sistema . . . 10

2.2.8 Aplicações a outras línguas . . . 11

2.3 PALMORF . . . 11

2.4 INTEX . . . 12

2.5 Jspell . . . 13

2.6 Prefixos e Sufixos . . . 14

3 Transdutores **17**

3.1 Especificação e Representação de um Transdutor . . . 17

3.1.1 Especificação de um Transdutor . . . 17

3.1.2 Tabela de Símbolos . . . 18

3.1.3 Representação Gráfica de um Transdutor . . . 18

3.2 Operações sobre Transdutores . . . 19

3.2.1 Operação *ShortestPath* . . . 19

3.2.2 Operação *Prune* . . . 21

3.3 Bibliotecas de Transdutores . . . 22

4 Arquitetura do LexMan **25**

4.1 Arquitetura Original do LexMan . . . 25

4.1.1 1ª Fase do Processamento - Geração de um Transdutor . . . 25

4.1.2 2ª Fase do Processamento - Atribuição de Etiquetas . . . 30

4.2 Regras de Recomposição . . . 33

4.2.1 Expansão da Geração de Palavras do LexMan . . . 34

4.2.2 Inclusão dos Compostos no Dicionário . . . 35

4.3 Segmentação e Análise Morfológica . . . 36

4.3.1 Normalização . . . 36
4.3.2 Transformação das Expressões Regulares . . . 38
4.3.3 Segmentador . . . 43
4.3.4 1ª Solução - *ShortestPath* . . . 47
4.3.5 2ª Solução - *Prune* . . . 52
4.4 Prefixos . . . 55
4.4.1 Geração de Prefixos . . . 55
4.4.2 Alterações à Geração de Palavras . . . 57
4.4.3 Construção do Transdutor do Dicionário . . . 58
4.4.4 Restrições às Palavras Prefixadas . . . 62
4.5 Pesquisa por Lema . . . 63

5 Avaliação **65**
5.1 Avaliação com Corpus . . . 65
5.2 Avaliação do Desempenho . . . 66
5.2.1 Metodologia da Avaliação do Desempenho . . . 66
5.2.2 1ª Solução - *ShortestPath* . . . 68
5.2.3 2ª Solução - *Prune* . . . 69
5.2.4 3ª Solução - *Prune* com Prefixos . . . 72
5.2.5 Evolução dos Resultados . . . 74
5.2.6 Avaliação da Memória . . . 75
5.3 Avaliação dos Prefixos . . . 76
5.3.1 Análise do Dicionário . . . 76
5.3.2 Lista de Palavras Desconhecidas . . . 76

6 Conclusões e Trabalho Futuro **79**
6.1 Contribuições . . . 79
6.2 Trabalho Futuro . . . 80

A Expressões Regulares **85**
A.1 Endereços HTTP . . . 85
A.2 Endereços IP . . . 85
A.3 Endereços de Correio Eletrónico . . . 85
A.4 Números Romanos . . . 85
A.5 Referências Bíblicas . . . 86
A.6 Números Cardinais . . . 87
A.7 Números Enas . . . 88
A.8 Números Ordinais . . . 88
A.9 Números Fracionários . . . 89
A.10 Números Ordinais/Fracionários . . . 89

Lista de Figuras

1.1 Cadeia de processamento de texto 1
1.2 Cadeia de processamento STRING 2
1.3 Segmentação da frase *A República Checa é na Europa.* na entrada do RuDriCo 2
1.4 Segmentação da frase *A República Checa é na Europa.* na saída do RuDriCo 2

2.1 Abordagem proposta por [Aït-Mokhtar, 1998] 8
2.2 Arquitetura do sistema SMORPH 10

3.1 Transdutor que identifica a sequência de carateres *ab* 18
3.2 Exemplo de um transdutor com dois caminhos 20
3.3 Resultado da operação *ShortestPath* sobre o transdutor da figura 3.2 20
3.4 Transdutor com vários caminhos com custos iguais 20
3.5 Resultado da operação *ShortestPath* sobre o transdutor da figura 3.4 21
3.6 Exemplo de um transdutor com quatro caminhos 21
3.7 Resultado da operação *Prune* sobre o transdutor da figura 3.6, com limiar = 0, 1, 2 e 3 . 21
3.8 Resultado da operação *Prune* sobre o transdutor da figura 3.6, com limiar = 4 22

4.1 Arquitetura da primeira fase de processamento do LexMan 26
4.2 Construção de parte do transdutor dos não-verbos a partir das palavras geradas 28
4.3 Construção de parte de um dos transdutores de clíticos 29
4.4 União dos transdutores das formas verbais após concatenação dos clíticos 30
4.5 União final que permite obter o transdutor do dicionário 30
4.6 Arquitetura da segunda fase de processamento do LexMan 31
4.7 Processamento do Etiquetador 32
4.8 Resultado da composição do segmento *amo* com o transdutor do dicionário 33
4.9 Caminho do composto *amigo pessoal* 36
4.10 Arquiteturas das duas soluções implementadas 37
4.11 Concatenação da variável de identificação ao transdutor do dicionário 45
4.12 Identificação de um espaço em branco 45
4.13 Novo segmentador 46
4.14 Processamento da 1ª solução 47
4.15 Construção de parte do transdutor dos não-verbos, na 1ª solução 48
4.16 Esquema do transdutor do dicionário usado na formação do segmentador da 1ª solução . . 49
4.17 Transdutor usado durante a análise morfológica, na 1ª solução 49
4.18 Resultado da composição da sequência *vi tudo.* com o segmentador 50
4.19 Resultado da aplicação da operação *ShortestPath* sobre o transdutor da figura 4.18 50
4.20 Transdutores dos segmentos encontrados no transdutor da figura 4.19 50
4.21 Resultado da composição do segmento *vi* com o analisador 51

4.22 Processamento da 2ª solução 52
4.23 Construção de parte do transdutor dos não-verbos, na 2ª solução 53
4.24 Resultado da composição da palavra *amo* com o segmentador com etiquetas 53
4.25 Resultado da aplicação da operação *Prune* sobre o transdutor da figura 4.24 54
4.26 Resultado da aplicação da operação *Prune* para o texto "*vi tudo.*" 54
4.27 Exemplo de caminhos construídos a partir de lista de prefixos gerados 58
4.28 Parte do transdutor que identifica adjetivos que começam pela letra *r* 59
4.29 Transdutor da figura 4.28 após determinização 59
4.30 União dos transdutores de cada categoria, com prefixos 60
4.31 União dos transdutores que formam o transdutor dos não-verbos 60
4.32 União dos transdutores que formam o transdutor dos verbos, com prefixos 61
4.33 Resultado da operação *Prune* para a cadeia de texto *Auto-estima* 61
4.34 Resultado da pesquisa para o lema *amo* 63

5.1 Estratégia da avaliação do desempenho 67
5.2 Soma dos tempos do Segmentador (na solução original), do LexMan e RuDriCo 74
5.3 Soma dos tempos médios de CPU gastos por palavra, pelo LexMan e RuDriCo 75

Lista de Tabelas

1.1 Tempo médio de CPU gasto por palavra na STRING 3

4.1 Caraterísticas dos transdutores antes e depois da remoção de sequências 43
4.2 Variáveis de identificação comuns às duas soluções e os seus custos 45
4.3 Variável de identificação específica da 1ª solução, com o seu custo 48
4.4 Variável de identificação específica da 2ª solução 52
4.5 Prefixos gerados a partir do lema *aero* e do paradigma Pref14 57
4.6 Prefixos gerados a partir do lema *de* e do paradigma Pref3 57
4.7 Variável de identificação específica para os transdutores dos prefixos 58

5.1 Caraterísticas dos ficheiros utilizados na avaliação de desempenho 66
5.2 Tempo médio de construção e tamanho do transdutor da solução original 67
5.3 Resultados da avaliação da solução original 67
5.4 Tempo médio de CPU gasto por palavra na solução original 68
5.5 Tempo médio de geração e tamanho dos transdutores da 1ª solução 68
5.6 Resultados da avaliação da 1ª solução 68
5.7 Tempo médio de CPU gasto por palavra na 1ª solução 69
5.8 Tempo médio de construção e tamanho do transdutor da 2ª solução para o dicionário D1 70
5.9 Resultados da avaliação da 2ª solução com o dicionário D1 70
5.10 Tempo médio de CPU gasto por palavra na 2ª solução com o dicionário D1 70
5.11 Tempo médio de construção e tamanho do transdutor da 2ª solução para o dicionário D2 71
5.12 Resultados da avaliação da 2ª solução com o dicionário D2 71
5.13 Tempo médio de CPU gasto por palavra na 2ª solução com o dicionário D2 72
5.14 Tempo médio de construção e tamanho do transdutor da 3ª solução para o dicionário D2 72
5.15 Resultados da avaliação da 3ª solução com o dicionário D2 73
5.16 Tempo médio de CPU gasto por palavra na 3ª solução com o dicionário D2 73
5.17 Tempo médio de construção e tamanho do transdutor da 3ª solução para o dicionário D3 73
5.18 Resultados da avaliação da 3ª solução com o dicionário D3 74
5.19 Tempo médio de CPU gasto por palavra na 3ª solução com o dicionário D3 74
5.20 Memória necessária para processar os ficheiros com a solução original e a mais recente . . 76

Capítulo 1

Introdução

O processamento de língua natural (PLN) é uma sub-área da Inteligência Artificial, que tem como um dos seus objetivos, melhorar as interações entre humanos e computadores, ao permitir que os humanos interajam com as máquinas da forma mais natural possível, utilizando a língua natural. A tradução automática e a correção ortográfica de texto são exemplos de aplicações práticas, resultantes do processamento de língua natural.

O processamento de texto pode dividir-se em várias etapas, como se ilustra na figura 1.1. Esta divisão é consensual, sendo usada por muitos sistemas de PLN.

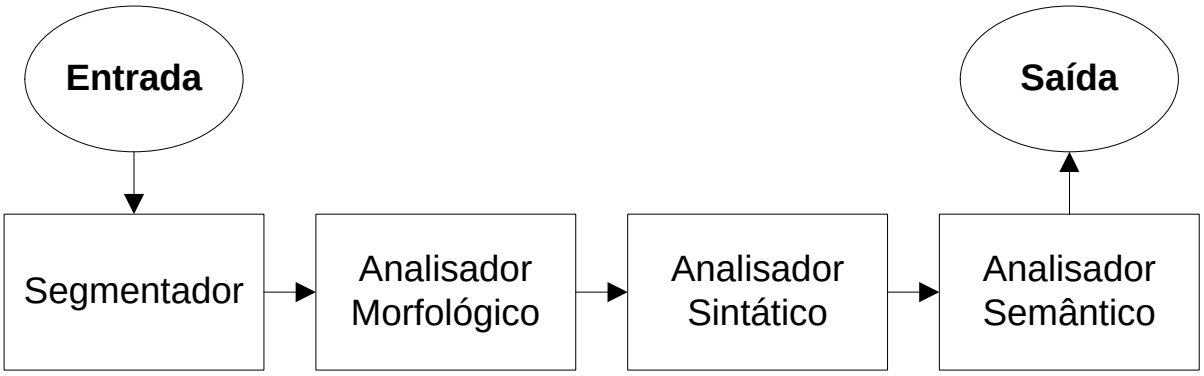

Figura 1.1: Cadeia de processamento de texto

A *segmentação de texto* é a primeira etapa do seu processamento, consistindo na divisão do texto em segmentos. Um segmento pode ser constituído por uma ou várias palavras, como por exemplo *anel* ou *chapéu de coco*, ou por um caráter de pontuação. Outros exemplos de segmentos são números, abreviaturas e endereços de correio eletrónico. Um segmentador tem de conseguir identificar corretamente todos estes elementos, o que faz com que a tarefa de segmentação seja um processo complexo. No entanto, apesar da sua importância e da sua complexidade, a segmentação é uma etapa muitas vezes ignorada nas descrições de cadeias de processamento de texto como, por exemplo, em [Jurafsky e Martin, 2009].

O *analisador morfológico* recebe os dados produzidos pelo segmentador e é responsável pela etiquetação morfossintática de cada segmento. A língua portuguesa contém diversas categorias de palavras: nomes, verbos, adjetivos, pronomes, artigos, etc. Certas palavras podem pertencer a várias categorias. Com a ajuda de um dicionário, o analisador morfológico atribui todas as etiquetas possíveis a cada segmento. Por exemplo, a palavra *canto* é uma palavra ambígua:

(1.1) *Eu canto todas as noites.*

(1.2) *A Maria colocou o vaso no canto da sala.*

Na frase (1.1), *canto* é um verbo, enquanto que na frase (1.2) é um nome. Nessas situações, atribui-se mais do que uma etiqueta ao segmento. Perante casos destes, o analisador morfológico inicia um processo de desambiguação, que lhe permite escolher uma etiqueta de entre as várias etiquetas possíveis. A variedade de categorias existentes na língua portuguesa e a necessidade de etiquetar segmentos ambíguos são fatores que dificultam a tarefa de um analisador morfológico.

A partir dos resultados do analisador morfológico, o analisador sintático procede à construção da estrutura do texto de entrada, através do auxílio de gramáticas.

O analisador semântico analisa essas estruturas e constrói novas estruturas que representam os possíveis significados que o texto pode ter.

A cadeia de processamento de língua natural, denominada STRING (*STatistical and Rule-based Natural lanGuage processing chain*), do Laboratório de Sistemas de Língua Falada (L^2F), tem uma arquitetura semelhante à que foi acima apresentada. A cadeia completa pode ser observada na figura 1.2.

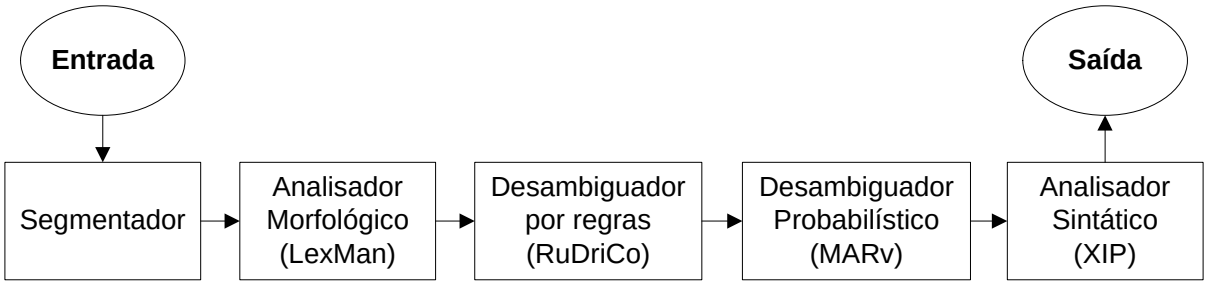

Figura 1.2: Cadeia de processamento STRING

O segmentador, para além de segmentar texto, também atribui etiquetas morfossintáticas a certos tipos de segmentos. Todos os segmentos são enviados para o LexMan, o módulo responsável por efetuar a análise morfológica [Diniz e Mamede, 2011]. O LexMan atribui etiquetas aos segmentos que o segmentador não conseguiu classificar. Na STRING, a análise morfológica e a desambiguação morfossintática são realizadas por dois módulos distintos. Após a análise morfológica, o desambiguador morfossintático RuDriCo [Diniz, 2010] tem como função resolver as ambiguidades de etiquetação produzidas pelo módulo anterior e alterar a segmentação. Para alterar a segmentação do texto, o RuDriCo utiliza um conjunto de regras de recomposição, que lhe permite fazer e desfazer a contração de segmentos. A figura 1.3 representa uma frase, dividida em segmentos, antes de ser transformada pelo RuDriCo. A figura 1.4 representa o resultado da alteração da segmentação, efetuada pelo RuDriCo.

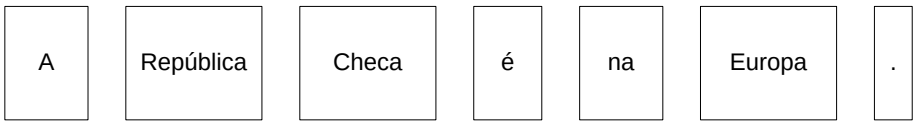

Figura 1.3: Segmentação da frase *A República Checa é na Europa.* na entrada do RuDriCo

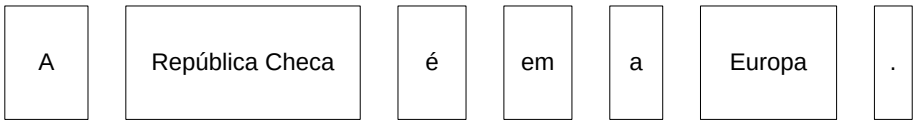

Figura 1.4: Segmentação da frase *A República Checa é na Europa.* na saída do RuDriCo

Na figura 1.4, pode ver-se que os segmentos *República Checa* foram transformados num único segmento. A contração ilustrada pela palavra *na*, por seu turno, foi dividida nos segmentos *em* e *a*.

O MARv [Ribeiro, 2003] trata dos problemas de ambiguidade morfossintática que o módulo anterior não resolveu. Este módulo baseia-se em modelos de Markov e usa o algoritmo de Viterbi [Viterbi, 1967]. O modelo de língua utilizado assenta em modelos de segunda ordem (trigramas), que codificam informação contextual relativa às entidades, e unigramas que codificam informação lexical.

O último módulo da STRING é o XIP (*Xerox Incremental Parser*) [Xerox, 2003], que constrói os fragmentos sintáticos (*chunks*) e estabelece as relações entre eles.

Esta dissertação centra-se no processo de segmentação e na ferramenta de análise morfológica, o LexMan.

Existem vários métodos conhecidos para segmentar texto. Um dos mais comuns é o uso de expressões regulares, que é o método atualmente utilizado pelo segmentador da STRING. Este segmentador é composto por 436 expressões regulares, codificadas na linguagem Perl.

A segmentação de texto também pode ser feita através da utilização de transdutores, uma vez que as expressões regulares podem ser transformadas em transdutores ([Beesley, 1998], [Karttunen, 2001] e [Karttunen et al., 1996]).

Outra possibilidade é a utilização da ferramenta Lex [Levine et al., 1992]. Com o auxílio de expressões regulares, o Lex permite gerar um autómato. Esse autómato é usado para ler o texto recebido na sua entrada e dividi-lo em segmentos. No entanto, esta ferramenta tem uma limitação: só suporta 32.000 estados, o que é insuficiente para satisfazer os requisitos atuais da STRING.

1.1 Objetivos do Trabalho

O tempo de processamento do segmentador da STRING depende do número de expressões regulares que têm de ser testadas. Neste momento, o segmentador contém 436 expressões regulares. Essas expressões regulares estão definidas numa sequência de *if ... else if*. As primeiras 435 expressões regulares têm a função de detetar sequências de texto especiais, como por exemplo, endereços de *email*, números ou abreviaturas. As restantes sequências de texto são detetadas pela última expressão regular. Para cada segmento que se quer encontrar, pode ser necessário testar várias expressões regulares. Um texto é composto maioritariamente por palavras que não representam sequências de texto especiais. Ou seja, na maioria dos casos, é necessário testar todas as expressões regulares para conseguir encontrar um segmento. Como o número de expressões regulares no segmentador tem vindo a aumentar à medida que a cadeia vem sendo desenvolvida, o tempo de processamento tem também aumentado. A evolução dos tempos de processamento da STRING pode ser observada na tabela 1.1. Os tempos são apresentados em milissegundos por palavra (ms/p).

Ano	Segmentador + LexMan (ms/p)	RuDriCo (ms/p)	MARv (ms/p)	XIP (ms/p)	Conv. (ms/p)	Total (ms/p)
2007	0,13	1,42	0,42	2,80	0,79	5,55
2009	0,11	4,24	0,2	1,67	0,50	6,73
2010	0,20	1,97	0,17	1,90	0,42	4,68
Início de 2011	0,19	1,86	0,12	1,64	0,20	4,00
Outubro de 2011	0,88	6,31	0,11	1,56	0,05	8,91

Tabela 1.1: Tempo médio de CPU gasto por palavra na STRING

Os módulos da cadeia STRING têm sido alvo de desenvolvimentos ao longo dos últimos anos. Para além disso, a evolução do *hardware* utilizado também influencia o tempo de processamento. O tempo de processamento do MARv baixou para quase metade em 2011, comparando com o anterior registo do ano

de 2009. Esta melhoria deve-se exclusivamente à evolução do *hardware*.

O tempo de processamento da segmentação e da análise morfológica, por palavra, é hoje 6,7 vezes maior do que em 2007, tal como se pode ver na tabela 1.1. Isto deve-se ao facto de o número de expressões regulares ter aumentado. Durante o ano de 2011, foram criadas expressões regulares para números e abreviaturas. Como consequência, nesse ano, observou-se o maior aumento no tempo médio de processamento de uma palavra gasto pelo segmentador. Outro ponto relevante é o facto de o tempo médio de processamento do segmentador corresponder a mais de 50% do tempo que o analisador sintático da STRING, o XIP. A análise sintática é uma tarefa bastante mais complexa que a segmentação. Como tal, seria de esperar que houvesse uma maior diferença entre os tempos médios de processamento do segmentador e do XIP. Pode-se portanto considerar que a segmentação não está a ser realizada de um modo muito eficiente.

O trabalho que aqui se propõe desenvolver tem vários objetivos. O primeiro objetivo consiste em criar um segmentador com transdutores e incorporá-lo no LexMan. O LexMan receberá texto, que será dividido em segmentos através de transdutores. O uso de transdutores na identificação de cada segmento torna o processo mais rápido, porque depende exclusivamente do comprimento do segmento que está a ser analisado naquele momento, e não do número de expressões regulares testadas.

O segundo objetivo é transferir para o LexMan, algumas regras de recomposição de texto que permitem juntar e dividir segmentos e que estão atualmente implementadas no RuDriCo. Como, para certas regras, é necessário ter informações sobre o contexto, propõe-se transferir para o LexMan todas as regras que não dependam do contexto.

O segundo módulo do LexMan gera um transdutor, com base nos ficheiros de palavras geradas e nos ficheiros responsáveis pelo tratamento dos clíticos. Outro objetivo deste trabalho é permitir que a informação usada na geração do transdutor seja complementada com informações sobre palavras derivadas e/ou flexionadas e neologismos com prefixos. Os prefixos são acrescentados no início de uma palavra: *anti–* (em *antifascista*) e *luso–* (em *lusodescendente*) são exemplos de prefixos. Pretende-se que o LexMan seja capaz de identificar palavras resultantes do processo de acréscimo de um prefixo a uma base. Nos exemplos acima, *fascista* e *descendente* são as bases das diferentes palavras derivadas por prefixação. Desta forma, estas palavras derivadas poderão ser identificadas e etiquetadas pelo LexMan, mesmo sem estarem alistadas nos ficheiros de entrada.

1.2 Estrutura do Documento

Este documento tem 6 capítulos e está estruturado da seguinte forma:

- O capítulo 2 apresenta os principais problemas que surgem durante os processos de segmentação e da análise morfológica. Descrevem-se vários analisadores morfológicos e os processos que permitem formar novas palavras após a adição de um afixo.
- No capítulo 3, define-se a forma como os transdutores podem ser construídos e como é que as operações disponibilizadas por bibliotecas de transdutores foram usadas para criar uma nova solução para o módulo LexMan.
- A arquitetura original do LexMan e as novas arquiteturas construídas no contexto deste trabalho são descritas no capítulo 4.
- O capítulo 5 apresenta as experiências que foram avaliadas e os resultados obtidos.
- As conclusões do trabalho e as sugestões para trabalho futuro são apresentadas no capítulo 6.

Capítulo 2

Trabalho Relacionado

Este capítulo descreve os vários problemas a resolver durante as fases de segmentação e análise morfológica do processamento de texto e as abordagens implementadas em analisadores morfológicos conhecidos.

Na secção 2.1, são abordadas as dificuldades que são habitualmente encontradas e que são comuns a várias línguas, durante a segmentação e a análise morfológica de texto. A secção 2.2 descreve o sistema SMORPH [Aït-Mokhtar, 1998], que é um analisador morfológico que junta num só processo a segmentação e a análise morfológica. Na secção 2.3, descreve-se o PALMORF, que é o analisador morfológico do sistema PALAVRAS [Bick, 2000]. A secção 2.4 apresenta o INTEX, um analisador morfológico que funciona com base em transdutores. Na secção 2.5, descreve-se o analisador morfológico Jspell, construído com base num corretor ortográfico. A secção 2.6 aborda algumas caraterísticas dos prefixos e sufixos existentes na língua portuguesa.

2.1 Segmentação e Análise Morfológica

A ambiguidade é a principal causa de dificuldades do processamento de língua natural, pois afeta todas as fases de processamento, em particular a análise morfológica [Silva, 2007].

As dificuldades que surgem durante a segmentação de texto em frases ou em palavras podem variar consoante a língua na qual o texto está escrito. Em inglês, tal como em português, as palavras são separadas por espaços, mas tal não acontece em línguas como o chinês e o japonês. No entanto, também há problemas comuns. [Jurafsky e Martin, 2009] referem que a existência de certos sinais de pontuação, e em particular o ponto, é a origem de muitos casos ambíguos, o que dificulta a identificação de expressões numéricas, de abreviaturas ou de expressões polilexicais.

A existência de clíticos ou de palavras derivadas por meio de prefixação ou sufixação em textos também dificulta a tarefa dos analisadores morfológicos. São dois problemas que exigem estratégias de análise específicas e que são comuns a várias línguas. As duas dificuldades são identificadas, por exemplo, em [Jurafsky e Martin, 2009] para a língua inglesa, em [Attia, 2006] para a língua árabe, em [Bick, 2000], [Silva, 2007] e [Alencar, 2009] para a língua portuguesa.

A segmentação de texto pode ser efetuada usando expressões regulares. Segundo [Karttunen et al., 1996], [Beesley, 1998] e [Karttunen, 2001], é possível transformar expressões regulares em transdutores. Portanto, os transdutores podem ser utilizados para realizar a segmentação de texto.

A mudança de uma abordagem baseada em expressões regulares para uma abordagem baseada em transdutores apresenta diversas vantagens. Os benefícios dessa abordagem e da utilização de transdutores

são referidos por [Roche e Schabes, 1995], [Beesley, 1998] e [Beesley, 2001]. Apresenta-se uma lista das vantagens enunciadas:

1 - Os transdutores são bidirecionais;

2 - Existe um conjunto de operações como a união, concatenação, composição ou intersecção, que podem ser aplicadas aos transdutores. Estas operações permitem dividir o problema em subproblemas mais simples de resolver;

3 - A atribuição de uma etiqueta morfossintática corresponde ao tempo necessário para percorrer um caminho num transdutor.

De acordo com [Beesley, 2001], a abordagem baseada em transdutores, aplicada durante a análise morfológica, é popular em várias partes do mundo e tem sido utilizada na construção de analisadores morfológicos para muitas línguas europeias e outras, como o japonês e o coreano. A aplicação de métodos baseados em transdutores, em processos da análise morfológica também é defendida em [Mohri, 1996] e em [Clark, 2002].

A segmentação e a análise morfológica podem ser realizadas separadamente, tal como acontece atualmente no LexMan. [Aït-Mokhtar, 1998] e [Garrido-Alenda et al., 2002] referem que, numa abordagem baseada em transdutores, é possível juntar o processo da segmentação ao analisador morfológico.

2.2 SMORPH

O sistema SMORPH (*Segmentation et MORPHologie*) é um analisador morfológico que agrupa numa etapa os passos (normalização, segmentação e análise morfológica) necessários que precedem a análise sintática [Aït-Mokhtar, 1998].

Durante a normalização, o texto recebido é alterado para estar de acordo com as exigências tipográficas do analisador morfológico. Podem ser feitas vários tipos de modificações, como por exemplo, reconstituir as palavras que foram hifenizadas devido à existência de um final de linha ou adicionar diacríticos em falta. Também podem ser feitas modificações que têm o objetivo de facilitar o processo de segmentação, através de inserções de espaços à volta dos travessões e dos pontos.

A segmentação consiste na divisão do texto em segmentos. Os critérios para dividir o texto também variam consoante os sistemas, a língua na qual o texto está escrito e os objetivos do processamento.

Na fase da análise morfológica, o objetivo é encontrar o lema e as informações linguísticas associadas a uma palavra. Desde os anos 80, foram criados vários sistemas responsáveis pela análise morfológica de texto, para várias línguas. Refira-se, como exemplo para o Português, o sistema Palavroso [Medeiros, 1995] e para a língua árabe, o *Buckwalter Arabic Morphological Analyzer* [Attia, 2006]. A maioria destes sistemas funciona com dicionários, representados com autómatos ou transdutores. Dessa forma, é possível representar centenas de milhar de palavras flexionadas, ocupando pouco espaço de memória e analisar milhares de palavras por segundo.

[Aït-Mokhtar, 1998] apresenta um conjunto de observações e conclusões sobre os problemas existentes na fase pré-sintática:

1 - A presença de carateres separadores não indica, de forma segura, o início ou o fim de um segmento. A sequência de palavras "África do Sul" pode ser considerada como um único segmento, apesar de haver espaços entre as várias palavras da sequência. Ou seja, a representação do dicionário e o algoritmo de análise têm de ter em conta que um segmento pode conter carateres separadores;

2 - Uma unidade sintaxicamente mínima (palavra, palavra flexionada, número ou caráter de pontuação), tal como aparece num dicionário, não tem de ser delimitada por separadores. A algumas dessas

unidades, pode juntar-se um prefixo ou um sufixo. Considera-se que os prefixos e os sufixos não podem ser seguidos e precedidos, respetivamente, de um caráter separador;

3 - Os carateres separadores distinguem-se dos outros carateres durante as fases de segmentação e da análise morfológica. Em particular, os espaçamentos, que são uma subclasse dos separadores, são eliminados da saída da segmentação/análise morfológica quando não fazem parte da constituição de um segmento;

4 - É necessário definir de uma forma rigorosa, no alfabeto de base do dicionário, a classe dos carateres separadores e dos espaçamentos;

5 - A divisão em segmentos pode ser ambígua. Algumas unidades sintaxicamente mínimas podem ser segmentadas de várias maneiras. Essa ambiguidade tem de ser resolvida posteriormente, durante a análise sintática ou semântica. Por isso, o algoritmo tem de devolver todas as possibilidades de segmentação;

6 - Excluindo os casos dos segmentos ambíguos, a segmentação é válida se os segmentos obtidos, percorrendo o texto da esquerda para a direita, correspondem aos maiores segmentos possíveis;

7 - As entradas do dicionário devem permitir indicar quais são os casos ambíguos e quais é que o não são. O processo de segmentação tem de ter acesso a essa informação, de modo a obter os melhores resultados possíveis. Para a segmentação poder aceder às informações do dicionário, a segmentação e a análise morfológica são realizadas num só processo;

8 - Enquanto o algoritmo de análise lê os carateres do texto, deve ser possível saber se existem segmentos conhecidos, maiores do que o atual, sem que seja necessário recomeçar a análise desde a posição inicial do segmento. Os transdutores ou autómatos são as estruturas ideais para atingir esse objetivo;

9 - As expressões polilexicais que contêm um caráter de espaçamento podem aparecer no texto com mais de um caráter de espaçamento. Uma expressão polilexical pode estar dividida em duas linhas se, a seguir ao espaço, houver um caráter de mudança de linha. Isso pode ser resolvido durante a análise considerando que as expressões polilexicais podem conter quaisquer sequências de carateres de espaçamento;

10 - As palavras do texto podem estar escritas, total ou parcialmente, com letras maiúsculas ou minúsculas. Também podem aparecer sem diacríticos;

11 - Alguns textos podem conter palavras que estão cortadas por um hífen, no final de uma linha. O resto da palavra começa na linha seguinte;

12 - Um caráter do texto pode representar um conjunto de outros carateres. Os carateres alternativos têm de ser associados ao dicionário, podendo ser modificados, consoante os textos a analisar, sem precisar de modificar o dicionário. O conjunto de carateres alternativos para um caráter do alfabeto denomina-se potencial.

Com base nessas observações e conclusões, [Aït-Mokhtar, 1998] sugere a abordagem proposta na figura 2.1.

A construção do dicionário consiste na declaração dos lemas, dos paradigmas de flexão e na geração de todas as formas possíveis. A análise morfológica consiste na utilização do dicionário para reconhecer as palavras que estão a ser analisadas para obter o(s) seu(s) lema(s) e a informação morfossintática.

Como um dicionário nunca está completo, a abordagem proposta sugere um módulo para as formas desconhecidas, ou seja, para aquelas que não estão no dicionário. Esse módulo permite reconhecer essas

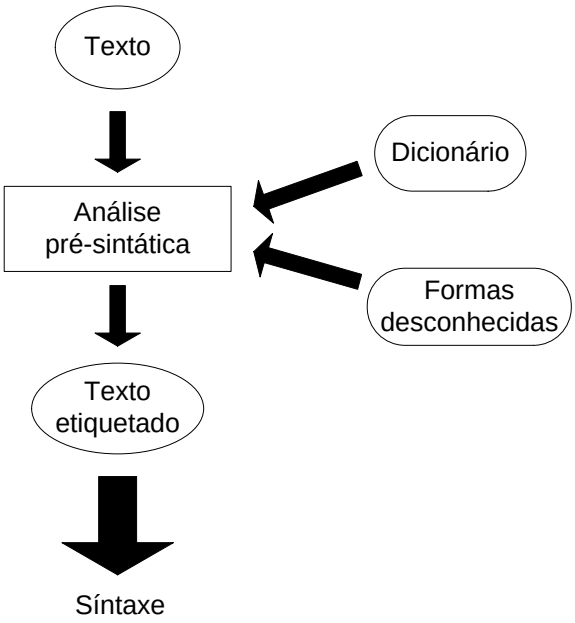

Figura 2.1: Abordagem proposta por [Aït-Mokhtar, 1998]

formas, através de expressões regulares associadas a informações morfossintáticas. O módulo das formas desconhecidas e o dicionário são utilizados durante a análise pré-sintática para realizar, simultaneamente, a segmentação e a análise morfológica.

O sistema SMORPH foi construído para validar a abordagem proposta. É uma ferramenta que permite definir um dicionário, contendo informações tipográficas e morfológicas, e compilá-lo. O dicionário é constituído por definições tipográficas sobre carateres, terminações, paradigmas de flexão e entradas lexicais. A compilação do dicionário gera um autómato lexical determinista e mínimo. A análise agrupa a segmentação em palavras e a análise morfológica, utilizando o dicionário compilado como fonte de informações.

Apresentam-se de seguida, algumas caraterísticas deste sistema, que demonstram de que forma são resolvidos os problemas tipográficos e morfológicos mencionados anteriormente, inerentes aos processos de segmentação e análise morfológica de texto.

2.2.1 Maiúsculas, minúsculas e diacríticos

As palavras de um texto podem estar escritas, total ou parcialmente, com letras maiúsculas ou minúsculas. Em certas línguas, como o francês, as maiúsculas podem estar escritas com ou sem diacríticos. O sistema SMORPH trata de todas essas variantes tipográficas, utilizando o conceito de potencial de caráter.

O papel de um potencial é representar, para cada caráter do alfabeto do dicionário, o conjunto dos outros carateres que ele pode representar. A informação dos potenciais está definida no dicionário. Para um texto escrito com maiúsculas sem acentos e com minúsculas com acentos, os potenciais podem ser definidos da seguinte forma:

POTENS:

A	a à â À Â .
B	b .
C	c ç Ç .
D	d .
E	e é è ê É È Ê .

...
POTENS/

Se for preciso tratar outros tipos de texto, basta redefinir os potenciais.

2.2.2 Prefixos e sufixos

No sistema SMORPH, os prefixos e os sufixos são identificados e segmentados com o mesmo dicionário. Eles são declarados usando o símbolo "ˆ". Para definir um prefixo, o símbolo "ˆ" é colocado no final da forma e significa que a seguir àquele ponto, não pode vir um caráter separador. Quando colocado no início da forma, no caso dos sufixos, significa que não pode ser precedido de um caráter separador. Apresenta-se um exemplo da definição de prefixos e sufixos, no sistema SMORPH:

```
ex–        exˆ        /pref/nom .
–ar        ˆar        /suf/verb .
–inho      ˆinho      /suf/nom .
```

As classes dos separadores e dos espaçamentos são também definidas no dicionário. Podem ser modificadas sem que haja necessidade de modificar ou de recompilar o dicionário. São definidas da seguinte forma:

```
ESPACOS:
        \9      # tabulação
        \10     # nova linha
        \32     # espaço
ESPACOS/

SEPARADORES:
            \9   \10  \32     # os espaços
            "    -    (   )   # aspas, hífen, paren.
            ,    ;    :       # virg., ponto-virg., 2-pont.
            .    !    ?       # pontos
SEPARADORES/
```

2.2.3 Expressões polilexicais

Algumas palavras podem conter carateres separadores. Nestes casos, os carateres separadores não correspondem ao local da divisão de segmentos e a palavra tem de ser analisada como se fosse um único segmento. Essas palavras podem ser definidas no dicionário:

```
chapéu-de-chuva  chapéu-de-chuva   /n/s/m .
recém-nascido    recém-nascido     /n/s/m .
batata_doce      batata_doce       /n/s/f .
```

O caráter "_ " representa uma sequência qualquer de carateres de espaçamento.

2.2.4 Ambiguidade

As formas ambíguas em segmentação são distinguidas das outras no dicionário, de modo a que a regra da forma mais longa não seja aplicada. Para as declarar, utiliza-se o símbolo "~":

```
FC  F.C        /n/f/s .  # Ponto é um ponto de fim de frase.
FC  F.C~.      /n/f/s .  # Segmento ambíguo.
```

2.2.5 Hifenização

O sistema SMORPH também trata dos casos de hifenização. O algoritmo que analisa o texto verifica se uma palavra composta por um hífen, é seguido de espaçamentos e se pelo menos um deles é uma mudança de linha. Se isso acontecer, a sequência de espaçamentos é ignorada.

2.2.6 Geração

O SMORPH contém um módulo de geração que utiliza o mesmo dicionário que a análise. Este módulo tem a função de gerar todas as formas flexionadas dos lemas, com o auxílio dos paradigmas de flexão.

2.2.7 Arquitetura do Sistema

O SMORPH é um sistema composto por três módulos: um compilador, um gerador e um analisador. A sua arquitetura está representada na figura 2.2.

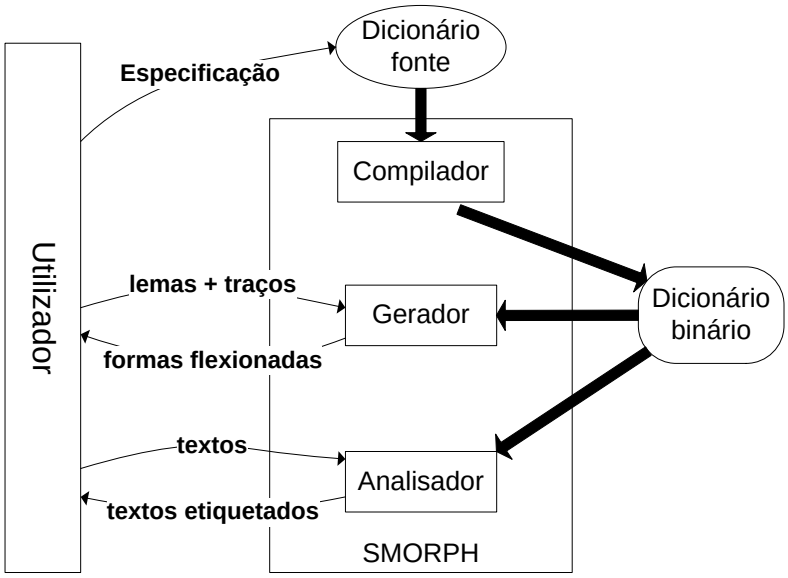

Figura 2.2: Arquitetura do sistema SMORPH

O utilizador tem a responsabilidade de especificar o dicionário-fonte de que o sistema precisa. A sua especificação consiste em definir:

- As definições ASCII das informações tipográficas sobre os carateres (classe dos separadores, dos espaçamentos e dos potenciais tipográficos);

- Os traços usados nas definições morfossintáticas associadas às formas. Para cada traço, é preciso indicar todos os seus possíveis valores. Estas informações são utilizadas pelo compilador, de modo

a manter a coerência do dicionário. São criadas e manipuladas listas de traços-valores onde é possível efetuar operações de disjunção e negação de traços;

- As terminações que serão utilizadas nos paradigmas de flexão;

- Os paradigmas de flexão. Cada paradigma descreve as flexões possíveis de uma classe de formas e os traços morfológicos correspondentes;

- As entradas lexicais onde se associam os paradigmas de flexão aos lemas.

O compilador de dicionários lê as entradas lexicais do dicionário-fonte, calcula as formas flexionadas e organiza-as com as suas informações tipográficas e morfológicas, para criar um autómato finito determinista. A identificação de estados equivalentes permite minimizar o autómato. O autómato é posteriormente compactado, usando um algoritmo de compressão, que regista os estados e as transições numa tabela. O dicionário, minimizado e compactado, é utilizado na geração ou na análise e pode ser guardado no mesmo formato, em disco. Por isso, o seu carregamento em memória é rápido. Tal como proposto, a segmentação e a análise morfológica são realizadas num só processo.

2.2.8 Aplicações a outras línguas

A abordagem proposta e a ferramenta implementada foram inicialmente pensadas para a língua francesa. Apesar disso, o sistema é também utilizado para a construção de dicionários eletrónicos (logo, para analisadores morfológicos) para a língua espanhola, portuguesa e para o alemão.

Cada língua pode apresentar problemas específicos de segmentação e análise morfológica. Uma das caraterísticas particulares do Português consiste no funcionamento dos pronomes clíticos. Estes têm a particularidade de poderem ocorrer em ênclise, isto é, entre o lema verbal e a terminação do tempo-modo do condicional e do futuro. O clítico *me* em *dar-me-á o livro* é um exemplo de um clítico em ênclise. Este fenómeno causa problemas ao SMORPH e a maneira encontrada para resolvê-los foi implementada com uma solução *ad hoc*. Esta solução baseia-se na utilização de dois traços, **raiz** e **termv**, que ficam associados respetivamente ao segmento raiz e de terminação à direita do clítico. Com base nessa solução, da análise à sequência *dar-me-á*, obtém-se [dar/raiz, –me–/clitico, á/termv]. Depois, agrupam-se o primeiro e o último segmento para reconstituir a forma verbal.

2.3 PALMORF

Esta secção descreve o PALMORF, que é o analisador morfológico do sistema PALAVRAS. O PALAVRAS é um analisador criado para analisar texto escrito na língua portuguesa [Bick, 2000]. O método aplicado neste sistema é o de *Constraint Grammar*[1], usado no contexto de *Progressive Level Parsing*, que consiste em várias etapas de processamento de texto: análise léxico-morfológica, desambiguação morfológica, análise sintática e análise semântica.

O PALMORF é um programa que recebe na sua entrada "texto corrido" e produz um ficheiro com a segmentação em palavras e frases resultante, assim como informações sobre as etiquetas atribuídas a cada palavra ou unidade polilexical. Essas etiquetas têm informações sobre a classe gramatical, flexões e derivação/composição. Para as palavras que são morfologicamente ambíguas, são encontradas várias hipóteses de etiquetação. A saída do PALMORF serve de entrada ao módulo responsável pela desambiguação morfológica e em última análise, aos módulos responsáveis pela análise sintática e semântica do PALAVRAS.

[1] `http://beta.visl.sdu.dk/constraint_grammar.html` (data de acesso: 13-05-2013)

O PALMORF pode receber na sua entrada uma única palavra ou um texto, o que influencia o modo como a análise é feita. Quando a entrada é apenas uma palavra, a análise é mais simples. No caso em que a entrada é constituída por um texto, a análise é feita por um módulo de pré-processamento e módulos de heurísticas. O módulo de pré-processamento trata dos casos das expressões polilexicais, abreviaturas, variações ortográficas e delimitação de frases. Os módulos de heurísticas aplicam heurísticas, dependentes do contexto. Os módulos de flexões e derivações fazem parte da análise, independentemente do PALMORF receber uma palavra ou um texto.

O pré-processamento é aplicado a todo o texto recebido na entrada. O primeiro passo é definir tudo o que não é uma palavra. No segundo passo, o pré-processador separa, por linhas, tudo o que ele considera ser palavras. Finalmente, um conjunto de expressões polilexicais, definidas por métodos *ad hoc*, é enviado para o léxico do PALMORF. É útil identificar o mais cedo possível, durante o processo de análise, as expressões polilexicais, de modo a reduzir a ambiguidade do texto.

O léxico do PALMORF contém informações sobre prefixos e sufixos da língua portuguesa. Para cada prefixo e sufixo, existe um conjunto de regras sobre o modo como eles podem ser combinados com palavras. Com base nessa informação, associado a cada prefixo e sufixo, está a informação morfossintática do prefixo ou sufixo que tem de ser atribuída à palavra.

O PALMORF é testado com corpora. Após a análise dos resultados obtidos, são implementadas novas estratégias, numa tentativa de melhorar o desempenho do sistema e obter melhores resultados. Após a implementação das estratégias, o PALMORF volta a ser testado com os mesmos textos e os resultados obtidos a partir dos novos testes são comparados com os antigos, para verificar se, de facto, houve uma melhoria nos resultados. Às vezes, verifica-se que as novas estratégias não produziram melhorias significativas. Nestes casos, é feita uma avaliação da complexidade do sistema. Se a implementação das novas estratégias aumentou a complexidade do sistema, dificultando a sua gestão, pode-se considerar que as melhorias obtidas nos resultados não compensam o custo do aumento da complexidade do sistema. A solução implementada é descartada e inicia-se a procura de novas estratégias.

Certos problemas de ambiguidade são causados pela forma como o texto é analisado. Um analisador morfológico olha individualmente para cada segmento. Alguns problemas de ambiguidade só podem ser resolvidos olhando para o contexto em que os segmentos são inseridos. A *Constraint Grammar* é uma abordagem gramatical que tem o objetivo de resolver esses casos de ambiguidade através da criação de regras de desambiguação. A utilização de regras permite ter uma visão do contexto em que cada segmento está inserido. O analisador morfológico devolve, para cada segmento, todas as etiquetas morfossintáticas possíveis. Um programa que implementa as regras de desambiguação deve receber na sua entrada, a saída gerada pelo analisador morfológico e escolher, de entre todas as etiquetas, aquela que melhor se adequa ao contexto em que o segmento está inserido e descartar as restantes.

2.4 INTEX

O INTEX é um sistema que foi construído com o objetivo de ser usado na análise automática de texto contendo milhões de palavras, a partir de descrições de línguas naturais fornecidas por linguistas [Mota, 1999]. Este sistema usa tecnologia baseado em estados finitos e é usado por investigadores europeus, que construiram dicionários e gramáticas para várias línguas: Inglês, Francês, Alemão, Italiano, Polaco, Português e Espanhol.

O sistema permite a identificação de palavras simples e compostas, associando-as a uma ou mais etiquetas através do analisador morfológico do sistema.

O analisador morfológico funciona com base em transdutores, o que permite a reutilização de transdutores já construídos para formar transdutores mais complexos, tais como números romanos e números por extenso. Sequências de texto representadas com expressões regulares podem ser convertidas para

transdutores equivalentes.

O INTEX aplica sobre o texto a analisar um dicionário de palavras flexionadas. Para a análise de textos escritos em português, existe um dicionário denominado DELAF. O DELAF é construído a partir de um dicionário de lemas, chamado DELAS, e um conjunto de regras de flexão. Cada regra de flexão é representada por um transdutor. As regras de recomposição que permitem juntar ou desfazer a contração de segmentos também existem sob a forma de transdutores. Os transdutores das regras de flexão têm operadores que permitem operar as mudanças necessárias para a geração de todas as palavras, nomeadamente, a eliminação e substituição de letras a aplicar ao lema da palavra.

As palavras derivadas são tratadas de um modo diferente. Como os prefixos e sufixos são comuns a várias palavras, eles são colocados nos seus próprios transdutores. Esses transdutores são depois combinados com as palavras para formar as palavras derivadas, quer seja por prefixação e/ou sufixação.

O processo de derivação pode exigir modificações na palavra à qual os prefixos e sufixos são adicionados. Pode ser necessário remover um acento por exemplo (*admirável* + *mente* → *admiravelmente*) ou mesmo letras (*des* + *humano* → *deshumano*). Estas modificações são feitas com a ajuda de operadores incluídos no transdutor.

As palavras ambíguas também são tratadas de um modo especial no INTEX. São aplicadas diferentes estratégias para prevenir estes casos e desambiguar as palavras da melhor maneira possível.

O dicionário DELAF, utilizado no INTEX, foi avaliado para medir o impacto das palavras ambíguas na língua portuguesa [Baptista e Faísca, 2007]. O impacto foi medido usando o INTEX e o corpus do CETEMPúblico[2].

2.5 Jspell

O Jspell é um analisador morfológico desenvolvido na Universidade do Minho, com base no corretor ortográfico Ispell [Simões e Almeida, 2002]. O código fonte do Ispell está disponível e com permissão para alterações. O desenvolvimento do Jspell consistiu na adição de funcionalidades ao Ispell, o que permitiu diminuir o tempo de desenvolvimento do Jspell, que pode ser utilizado através de uma interface *online*[3].

O Jspell baseia-se num dicionário e num conjunto de regras de flexão e derivação. Desta forma, não é necessário listar todas as palavras, o que permite ter um dicionário mais pequeno. Cada entrada do dicionário corresponde a um lema, ao qual é associado a sua descrição morfológica e um conjunto de regras de flexão e derivação. As regras definem como é que uma palavra é formada, a partir do lema. Contêm informações sobre as alterações a aplicar ao lema e nas suas propriedades morfológicas. Apresenta-se a sintaxe de uma regra que descreve a formação do plural para palavras que acabam em *ão*:

```
Ã O          >      -ÃO,ÕES ; "N=p"
```

No processo de formação de palavras, estas alterações podem ser a remoção e adição de letras ao lema. Relativamente a alterações das propriedades morfológicas, pode haver mudanças ao nível do número (p.e. de singular para plural), de género (p.e. masculino para feminino) e nos casos da adição de afixos, a categoria gramatical também pode mudar. Por exemplo, a adição do sufixo *–mente* ao adjetivo *anterior* implica uma mudança de categoria gramatical de adjetivo para advérbio, formando a palavra *anteriormente*.

[2]Corpus de Extratos de Textos Eletrónicos MCT/Público
[3]`http://natura.di.uminho.pt/webjspell/jsol.pl` (data de acesso: 13-05-2013)

2.6 Prefixos e Sufixos

A análise morfológica de uma língua como o Português implica o correto processamento de dois conjuntos de fenómenos: a **flexão** e os processos de **formação de palavras**.

A **flexão** consiste na variação formal do lema de uma unidade lexical nas categorias gramaticais caraterísticas da respetiva parte-do-discurso. Temos assim os verbos a flexionar (conjugar) em tempo, modo, pessoa e número; os nomes a flexionar em género e número; os adjetivos em género e número (e por vezes em grau); e os advérbios ocasionalmente a apresentar variação em grau.

Esta variação formal é um processo produtivo e em larga medida regular (na STRING, o LexMan é o módulo responsável por este processamento). A **formação de palavras** é normalmente estruturada em processos de **derivação** e de **composição**.

Do ponto de vista morfológico, a **derivação** é o processo onde se formam novas unidades lexicais a partir das unidades já existentes através da adição de afixos às formas de base. Em Português, há apenas dois tipos de afixos: os prefixos, que se juntam ao início da base e os sufixos, que se juntam ao final da base. Os prefixos não alteram a categoria gramatical da base. Os sufixos podem alterar a categoria gramatical da base.

Os sufixos podem ser organizados, entre outros critérios, pela categoria resultante do processo derivacional. Existem **sufixos nominais**, que dão origem a nomes e adjetivos e **sufixos verbais**, que permitem formar verbos. O sufixo *–mente* tem um comportamento especial, dando origem a advérbios (sobretudo advérbios de modo) a partir de formas de base adjetivais.

Tanto os prefixos como os sufixos são morfemas que acrescentam um significado particular ao significado da base. Assim por exemplo, os prefixos *re–* e *des–* juntam-se a bases para criar novas palavras que significam "repetição" ou "acção contrária" do significado da base, como em *refazer* ou *desfazer*. Os sufixos *–idade* ou *–agem* também se podem juntar a bases adjetivais e verbais, respetivamente, associando a noção de "qualidade"/"atributo" ou "resultado de processo"/"processo", tal como em *gramaticalidade* (de *gramatical*) ou *lavagem* (de *lavar*).

Certos processos derivacionais podem ser aplicados de forma recursiva, obtendo-se novas palavras formadas sobre palavras derivadas mais simples. Por exemplo, palavras como *impagável* podem ser analisadas pela adjunção do prefixo *im–* ao adjetivo deverbal *pagável*, que, por sua vez, deriva de *pagar* e do sufixo *–vel*, sendo possível representar a palavra por uma estrutura hierarquizada como: (*im–* (*pagar* + *vel*)).

Em alguns casos, a palavra derivada forma-se a partir da adição simultânea de prefixo e sufixo (este fenómeno é conhecido por **parassíntese**). É o caso da palavra *repatriar* (de *re* + *pátria* + *ar*) (exemplo de [Cunha e Cintra, 1986, p. 103]) que não pode ser derivado de formas de base obtidas apenas por pre- ou sufixação, ou seja, as formas *repátria* e *patriar* não existem.

Muitos processos derivacionais, porém, já não são sincronicamente produtivos, quando a base e o derivado adquiriram novos significados, não sendo possível relacioná-los através de regras produtivas e gerais. É o caso, por exemplo, de *lavagem* (comida que se dá aos animais, em particular aos porcos) ou de *desfazer* (*O Pedro desfez a Ana* com o significado "dizer mal de", "insultar"). Independentemente da sua origem histórica, estes casos são entradas lexicais autónomas, não estando mais relacionadas, através de processos derivacionais produtivos, às formas de base que lhe deram origem.

Nesta dissertação, não se atribuirá importância a estes aspetos históricos, da diacronia. A preocupação será identificar adequadamente as palavras derivadas – em especial as palavras regular e produtivamente formadas – a partir de formas de base lexicalizadas e que, por não estarem no dicionário ou por se tratarem de neologismos, não são ainda reconhecidas pelo sistema.

No quadro da derivação, existe também uma distinção entre os processos como os que ilustrámos acima e a chamada **derivação erudita**. Trata-se igualmente de formação de novas palavras mas recorrendo a

prefixos e sufixos de origem clássica (gregos e latinos) e usuais em certos domínios científicos e técnicos.

A estas expressões, estão associados problemas de análise específicos. Apesar de se poder reconhecer um dos formantes (prefixo ou sufixo) nem sempre a forma de base existe na língua contemporânea. É o caso de *cleptomaníaco*, em que o sufixo *maníaco*, que designa o estado psicológico patológico, se junta diretamente ao radical do verbo grego "roubar".

Os processos de derivação envolvem a utilização de afixos, isto é, de **morfemas presos** que não podem existir sem estarem ligados a formas de base (ou radicais). A **composição** é um processo que combina **morfemas livres**, isto é, duas ou mais palavras autónomas para formar uma nova unidade lexical. Frequentemente, o significado do composto não é composicional, porque não pode ser calculado a partir do significado que cada uma das palavras componentes apresenta quando usada separadamente. Os problemas de identificação de palavras compostas no quadro da cadeia STRING são abordados em [Portela, 2011].

Para a escrita desta secção, foram consultados os livros de [Cunha e Cintra, 1986] e de [Mira Mateus et al., 2003], que abordam os dois fenómenos mencionados: a flexão e a formação de palavras.

Capítulo 3

Transdutores

Este capítulo descreve algumas propriedades dos transdutores e das bibliotecas de transdutores, assim como exemplos que demonstram como é que essas propriedades foram aproveitadas no contexto deste trabalho.

Na secção 3.1, apresentam-se alguns exemplos do modo como os transdutores utilizados neste trabalho são especificados e representados. Na secção 3.2, descrevem-se as operações que foram utilizadas para construir os transdutores e elaborar duas soluções para atingir os objetivos propostos neste trabalho. Na secção 3.3, indicam-se as bibliotecas de transdutores usadas e as razões que levaram à sua utilização.

3.1 Especificação e Representação de um Transdutor

Um transdutor pode ser especificado através da criação de um ficheiro de texto e de uma tabela de símbolos. O ficheiro de texto é uma representação textual do transdutor. Os transdutores também podem ter uma representação gráfica. Nesta secção, estes componentes são apresentados através da especificação do transdutor que identifica a sequência de carateres *ab*.

Na secção 3.1.1, descrevem-se as caraterísticas de um transdutor e de que forma pode ser especificado. A construção da tabela de símbolos é apresentada na secção 3.1.2. A representação gráfica do transdutor criado é apresentada na secção 3.1.3.

3.1.1 Especificação de um Transdutor

Um transdutor é composto por estados e transições. Cada transição de um transdutor tem um símbolo de entrada, um símbolo de saída e um custo. Uma transição tem um estado de partida e um estado de chegada (podem coincidir). Cada estado tem um identificador (ID). Um transdutor tem um único estado inicial e esse estado tem sempre o ID igual a zero. Um transdutor tem um ou mais estados finais, que também têm um custo. O custo pode ser um número inteiro ou decimal.

Um transdutor pode ser especificado através da criação de um ficheiro de texto que respeita o formato AT&T[1]. Nesse ficheiro de especificação, indicam-se as transições entre os estados do transdutor juntamente com as suas propriedades. São também indicados os estados finais. O transdutor que identifica a sequência de carateres *ab* é especificado da seguinte forma:

```
0    1    a    a    0.5
1    2    b    b
2    7
```

[1] `http://www2.research.att.com/~fsmtools/fsm/man4/fsm.5.html` (data de acesso: 13-05-2013)

Cada linha do ficheiro de texto tem a especificação de uma transição ou de um estado final do transdutor. No exemplo apresentado, as duas primeiras linhas representam duas transições. A primeira coluna corresponde ao ID do estado de partida da transição. A segunda coluna corresponde ao ID do estado de chegada da transição. A terceira e quarta coluna correspondem, respetivamente, ao símbolo de entrada e ao símbolo de saída da transição. Na quinta coluna, indica-se o custo da transição. Esta última coluna é opcional quando se considera que a transição tem um custo igual a zero. Assim, é possível verificar que a transição que corresponde ao caráter *a* tem um custo igual a 0.5 e que a transição do caráter *b* tem um custo igual a zero. Na terceira e última linha, indica-se que o estado com o ID igual a 2 é o estado final do transdutor. Na segunda coluna, indica-se o custo do estado final que neste exemplo, é igual a 7. Esta segunda coluna é opcional se o custo for igual a zero. Os símbolos usados na especificação de um transdutor têm de estar mapeados numa tabela de símbolos.

3.1.2 Tabela de Símbolos

A representação interna de um símbolo de entrada ou de um símbolo de saída é um inteiro. A criação de uma tabela de símbolos permite mapear cada símbolo usado na especificação de um transdutor com um número inteiro. Apresenta-se de seguida um exemplo de uma tabela de símbolos que respeita o formato AT&T e que pode ser usada para o transdutor especificado no exemplo da secção 3.1.1:

```
eps     0
a       97
b       98
c       99
d       100
```

O número 0 está reservado para o *epsilon*, que representa uma sequência vazia de carateres. O *epsilon* é representado pelo símbolo *eps* nas transições. As linhas seguintes mapeiam os códigos UTF-8 dos carateres *a*, *b*, *c* e *d*. Uma tabela de símbolos pode ser usada para construir um transdutor em que os símbolos de entrada e de saída utilizados no transdutor pertencem à tabela de símbolos. A tabela de símbolos apresentada pode ser usada para construir o transdutor especificado na secção 3.1.1, visto que os símbolos *a* e *b* estão mapeados nessa tabela.

No contexto deste trabalho, as tabelas de símbolos são construídas de uma forma equivalente. Usa-se um intervalo de 1 a 255 para poder representar em transdutores os códigos UTF-8 dos carateres pertencentes aos blocos *Unicode Basic Latin* e *Latin-1 Supplement* da tabela de codificação UTF-8[2].

3.1.3 Representação Gráfica de um Transdutor

A figura 3.1 é uma representação gráfica do transdutor que identifica a sequência de carateres *ab*, especificado na secção 3.1.1, usando a tabela de símbolos apresentada na secção 3.1.2. No exemplo apresentado, mostra-se a notação usada para representar o símbolo de entrada, o símbolo de saída e os pesos.

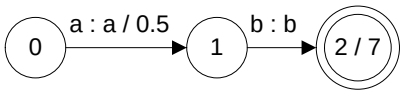

Figura 3.1: Transdutor que identifica a sequência de carateres *ab*

[2] `http://www.utf8-chartable.de` (data de acesso: 13-05-2013)

Neste documento, os estados são representados por círculos numerados. Os estados finais diferenciam-se dos outros através de um duplo círculo. As transições são representadas por arcos dirigidos. Os símbolos de entrada, símbolos de saída e o custo de cada transição são colocados junto da própria transição. Os símbolos de entrada são separados dos símbolos de saída pelo símbolo de dois pontos (:). O peso da transição é separado do símbolo de entrada e do símbolo de saída por uma barra (/). Nos estados finais, a barra separa o identificador do estado do custo desse estado. O custo de uma transição ou de um estado final pode não ser representado caso seja igual a zero.

3.2 Operações sobre Transdutores

Existe um conjunto de operações que podem ser usadas para construir e combinar transdutores, de modo a obter transdutores mais complexos. No contexto deste trabalho, foram utilizadas as seguintes operações: fecho (*Closure*), compilação (*Compile*), composição (*Compose*), concatenação (*Concat*), determinização (*Determinize*), diferença (*Difference*), codificação (*Encode*), poda (*Prune*), remoção de *epsilons* (*RmEpsilon*), caminho mais curto (*ShortestPath*), ordenação topológica (*TopSort*) e união (*Union*).

A operação *Compile* transforma os ficheiros de texto que definem a especificação de um transdutor para o formato próprio da biblioteca, o que permite que o transdutor possa ser manipulado por outras operações. A aplicação da operação *Closure* sobre um transdutor permite que as sequências de texto identificadas por esse transdutores sejam identificadas zero ou mais vezes seguidas. A composição de transdutores é feita entre 2 transdutores, por exemplo, A e B. Os símbolos de saída do transdutor A emparelham com os símbolos de entrada do transdutor B. O resultado é um transdutor com os caminhos que emparelharam desde o estado inicial até um estado final do transdutor B. A concatenação de um transdutor A com um transdutor B permite a identificação de uma sequência que pertence ao transdutor A, seguida de uma sequência que pertence ao transdutor B. A determinização de um transdutor junta os estados equivalentes do transdutor. A operação *Encode* permite que os pares de símbolos entrada-saída das transições sejam representados por um único símbolo. A união de 2 transdutores cria um único transdutor, que permite a identificação de todas as sequências que são identificadas por cada transdutor usado na operação da união. A operação *TopSort* ordena topologicamente os estados de um transdutor, de acordo com o identificador de cada estado. Aos transdutores que têm transições com o símbolo *epsilon* como símbolo de entrada e símbolo de saída, pode ser aplicada a operação *RmEpsilon*, que cria um transdutor equivalente sem aquelas transições.

A possibilidade de usar as operações *ShortestPath* e *Prune* permitiu a implementação de duas arquiteturas para o novo módulo LexMan. Estas duas operações são descritas nas secções 3.2.1 e 3.2.2.

3.2.1 Operação *ShortestPath*

Um transdutor pode ser visto como um conjunto de caminhos que partem de um estado inicial e que acabam num estado final. Os caminhos são constituídos pelas transições que são percorridas para chegar a um estado final, partindo do estado inicial do transdutor. Tal como mencionado na secção 3.1, cada transição tem um peso associado. Para a operação *ShortestPath*, esses pesos funcionam como um custo. A segmentação de uma palavra pode ser ambígua. Com transdutores, é possível definir prioridades entre as segmentações de uma palavra. Se cada segmentação corresponder a um caminho de um transdutor, é possível atribuir um custo a cada um dos caminhos usando pesos. O caminho com o custo mais baixo corresponde à melhor segmentação.

Esta operação permite encontrar os n-caminhos com os custos mais baixos do transdutor. O custo de um caminho corresponde à soma dos pesos associados às transições e ao estado final que compõem esse caminho. Os exemplos de aplicação da operação *ShortestPath* apresentados correspondem sempre

à obtenção do caminho que tem o custo mais baixo de todos, denominado 1-caminho. A figura 3.2 representa um transdutor composto por dois caminhos, com custos diferentes.

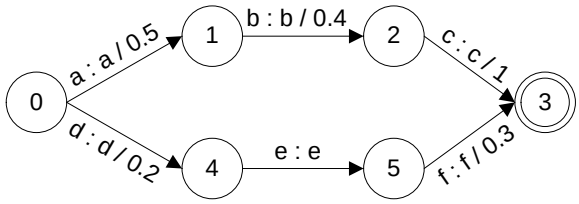

Figura 3.2: Exemplo de um transdutor com dois caminhos

Aplicando a operação *ShortestPath* sobre o transdutor da figura 3.2, obtém-se um transdutor correspondente ao caminho de menor custo daquele transdutor. Neste caso, é o caminho da sequência *def* com custo igual a 0.5. A figura 3.3 ilustra o resultado da aplicação da operação *ShortestPath* sobre o transdutor representado na figura 3.2.

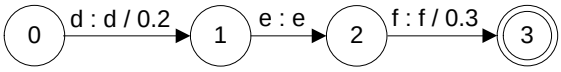

Figura 3.3: Resultado da operação *ShortestPath* sobre o transdutor da figura 3.2

O transdutor apresentado na figura 3.2 contém dois caminhos, com custos diferentes. Um transdutor pode conter vários caminhos com custos iguais.

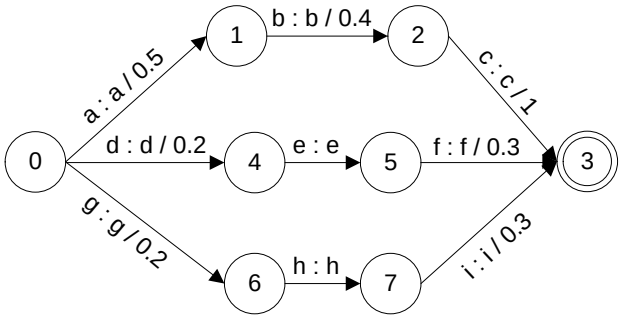

Figura 3.4: Transdutor com vários caminhos com custos iguais

A figura 3.4 representa um transdutor com essas caraterísticas. Tal como se pode observar na figura 3.4, existem dois caminhos com o menor custo, correspondentes às sequências de carateres *def* e *ghi*. Usando a operação *ShortestPath*, só um dos dois caminhos é que faz parte do transdutor produzido pela operação *ShortestPath*. A figura 3.5 representa o resultado da aplicação da operação *ShortestPath* sobre o transdutor da figura 3.4. Do conjunto de caminhos com menor custo, a operação escolhe o último caminho que foi especificado na construção do transdutor.

Usando pesos e a operação *ShortestPath*, é possível obter o caminho com o custo mais baixo, correspondente à melhor segmentação de uma sequência de carateres. É esse o raciocínio que levou à elaboração da solução descrita na secção 4.3.

No entanto, tal como demonstrado no exemplo acima, se houver mais de um caminho cujo custo é igual ao custo do caminho com o menor custo, a operação *ShortestPath* só escolhe um desses caminhos. Se houver informação morfológica associada a cada um dos segmentos representados nesses caminhos, a aplicação da operação *ShortestPath* não permite obter toda a informação necessária. Por causa dessa limitação, criou-se uma segunda estratégia, também descrita na secção 4.3, que se baseia na utilização da operação *Prune*, descrita na secção 3.2.2.

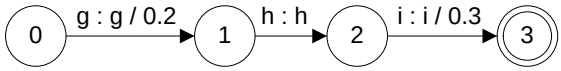

Figura 3.5: Resultado da operação *ShortestPath* sobre o transdutor da figura 3.4

3.2.2 Operação *Prune*

A operação *Prune* recebe como argumento um valor que é utilizado como limiar e apaga do transdutor os caminhos cujos custos têm uma diferença maior do que o limiar definido, relativamente ao caminho de menor custo presente no transdutor. Por exemplo, se o limiar for igual a zero, os caminhos que não correspondem aos caminhos de menor custo do transdutor são removidos. Se o limiar for igual a dois, a operação mantém os caminhos de menor custo e aqueles cuja diferença de custo em relação aos caminhos de menor custo do transdutor é menor ou igual a dois.

A figura 3.6 ilustra um transdutor com quatro caminhos, antes de ser aplicado a operação *Prune*: 2 dos caminhos têm um custo igual a seis, outro tem um custo igual a dez. O 4º caminho tem um custo igual a onze.

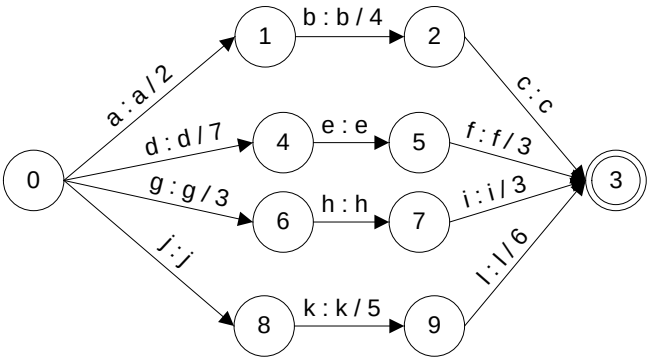

Figura 3.6: Exemplo de um transdutor com quatro caminhos

Aplicando a operação *Prune* sobre o transdutor representado na figura 3.6 e escolhendo um limiar igual a zero, o transdutor que se obtém contém unicamente os caminhos de menor custo do transdutor, ou seja, os caminhos com custo igual a seis, que representam as sequências de carateres *abc* e *ghi*. O transdutor obtido está representado na figura 3.7.

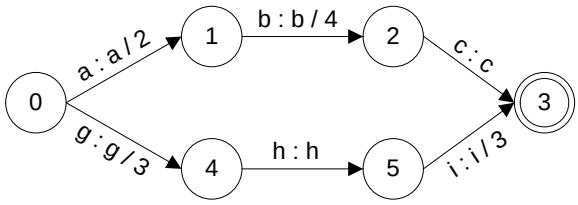

Figura 3.7: Resultado da operação *Prune* sobre o transdutor da figura 3.6, com limiar = 0, 1, 2 e 3

Variando o valor do limiar para 1, 2 e 3, o resultado da aplicação da operação *Prune* sobre o transdutor da figura 3.6 continua a ser o transdutor representado na figura 3.7. Tal deve-se ao facto de não existirem outros caminhos no transdutor com essas diferenças de custo em relação aos caminhos de menor custo do transdutor.

O resultado varia escolhendo o valor 4 para o limiar. Com esse novo limiar, o transdutor resultante da aplicação da operação *Prune* sobre o transdutor da figura 3.6 contém os caminhos de menor custo do transdutor assim como todos os caminhos cuja diferença de custo em relação a esses caminhos é menor ou igual a 4. O caminho que representa a sequência de carateres *def* enquadra-se nesse critério, portanto,

com um limiar igual a 4, o transdutor produzido contém esse terceiro caminho. O resultado é apresentado na figura 3.8.

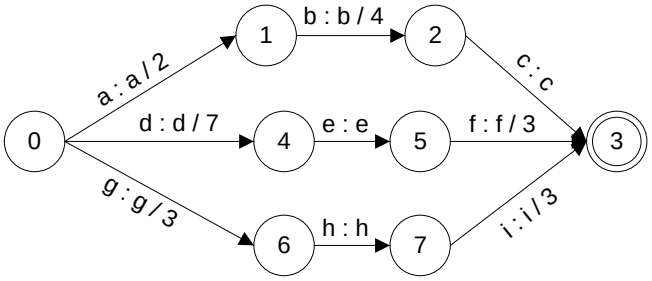

Figura 3.8: Resultado da operação *Prune* sobre o transdutor da figura 3.6, com limiar = 4

Como o quarto caminho do transdutor da figura 3.6 tem um custo igual a 11, a diferença de custo em relação aos caminhos de menor custo do transdutor é 5. Portanto, seria necessário aplicar a operação *Prune* com um limiar igual a 5 para que esse quarto caminho fosse incluído no transdutor resultante da operação. Nesse caso, o transdutor obtido seria igual ao transdutor original (figura 3.6).

Tal como exemplificado na figura 3.7, a operação *Prune* permite obter todos os caminhos que correspondem a um caminho de menor custo do transdutor sobre o qual a operação é aplicada. No contexto deste trabalho, esta funcionalidade é útil para encontrar todas as melhores segmentações para uma cadeia de carateres. Essa possibilidade permitiu a elaboração da solução descrita na secção 4.3.

3.3 Bibliotecas de Transdutores

Para poder utilizar as operações apresentadas na secção 3.2, é necessário ter acesso a uma biblioteca de transdutores. Durante o desenvolvimento da solução original do LexMan, foram utilizadas duas bibliotecas de transdutores: a biblioteca *FSM* da AT&T[3] e a biblioteca *OpenFst*, desenvolvida pela *Google Research* e pelo Instituto Courant de Ciências Matemáticas da Universidade de Nova Iorque[4].

Ambas as bibliotecas permitem a criação de transdutores em que cada transição tem um peso associado. Estas bibliotecas colocam ao dispor dos seus utilizadores um conjunto de operações que permitem a criação e combinação de transdutores, o que permite criar transdutores mais complexos a partir de transdutores mais simples. Para além disso, a biblioteca *OpenFst* pode ser utilizada por um programa escrito na linguagem C++, dando acesso a um conjunto de funções que permitem criar, combinar e efetuar pesquisas sobre transdutores a partir do programa.

O facto da biblioteca *OpenFst* disponibilizar as operações necessárias para construir e manipular transdutores, dando a possibilidade de o fazer também num programa escrito em C++, foi tido em conta na escolha de uma biblioteca de transdutores para implementar a solução original do LexMan. No entanto, esta biblioteca tem uma limitação. Antes do início deste trabalho, a primeira fase do processamento do LexMan consistia na geração de um transdutor que correspondia a um dicionário de palavras. As caraterísticas do transdutor do dicionário impedem a sua determinização usando somente a biblioteca *OpenFst*. Só é possível aplicar a operação de determinização da biblioteca *OpenFst* em transdutores que sejam funcionais, ou seja, quando cada símbolo de entrada existente no transdutor estiver associado a um único símbolo de saída. Se no transdutor houver pares *entrada-saída* que partilham o mesmo símbolo de entrada mas símbolos de saída diferentes, o transdutor não é funcional e não pode ser determinizado usando a biblioteca *OpenFst*. O transdutor do dicionário da arquitetura original do LexMan, assim como o transdutor do dicionário construído neste trabalho não são funcionais, tal como demonstrado no

[3] `http://www2.research.att.com/~fsmtools/fsm/` (data de acesso: 13-05-2013)
[4] `http://www.openfst.org` (data de acesso: 13-05-2013)

capítulo 4. Por isso, no desenvolvimento da solução original do LexMan, usou-se a biblioteca *FSM* para determinizar o transdutor do dicionário. O transdutor resultante é depois convertido para um transdutor da biblioteca *OpenFst*.

Por estas razões, decidiu-se continuar a utilizar as bibliotecas *FSM* e *OpenFst* no desenvolvimento de uma nova solução para o LexMan.

Capítulo 4

Arquitetura do LexMan

Na secção 4.1, apresenta-se a arquitetura do módulo LexMan antes do início deste trabalho. Para atingir os objetivos propostos neste trabalho, foram feitas alterações no segmentador e no analisador morfológico da cadeia STRING, o LexMan. Na secção 4.2, descreve-se como as regras de recomposição sem contexto, que estavam no RuDriCo, foram transferidas para o LexMan. A secção 4.3 descreve a solução encontrada para incorporar um segmentador construído com transdutores no LexMan, substituindo o segmentador da cadeia STRING.

Para finalizar, apresentam-se na secção 4.4, as alterações realizadas, para permitir que o LexMan seja capaz de identificar palavras resultantes do processo de acréscimo de um prefixo a uma base.

4.1 Arquitetura Original do LexMan

O LexMan é responsável pela análise morfológica da STRING. O seu processamento divide-se em duas fases: geração de um transdutor e atribuição de pelo menos uma etiqueta morfossintática a uma palavra. Na secção 4.1.1, apresenta-se uma descrição das várias etapas que permitem construir o transdutor que é usado como dicionário de palavras da análise morfológica. Na secção 4.1.2, mostra-se como se obtêm as etiquetas morfossintáticas para as palavras recebidas na entrada do LexMan.

4.1.1 1ª Fase do Processamento - Geração de um Transdutor

O transdutor construído durante a primeira fase de processamento do LexMan é construído antes de se iniciar a análise morfológica. A arquitetura da primeira fase do LexMan está representada na figura 4.1.

O primeiro módulo é um gerador de palavras que tem como base os dicionários de lemas e os paradigmas de flexão. A cada palavra gerada, é associada uma etiqueta morfossintática. O segundo módulo tem como objetivo gerar um transdutor para as palavras geradas pelo módulo anterior e combinar todos esses transdutores num único transdutor. Este módulo também usa a informação sobre os clíticos para complementar as formas verbais.

Na secção 4.1.1.1, apresenta-se a solução utilizada para gerar as palavras do dicionário, usando os lemas e paradigmas de flexão. Nas secções 4.1.1.2, 4.1.1.3 e 4.1.1.4, mostra-se como as palavras geradas, assim como os clíticos das formas verbais, são utilizados para criar um transdutor correspondente ao dicionário de palavras que é utilizado durante a análise morfológica.

4.1.1.1 Geração de Palavras

A geração de palavras realiza-se usando o dicionário de lemas e os paradigmas de flexão. É um processo documentado pelos seus autores [Diniz e Mamede, 2011]. Os paradigmas de flexão indicam que trans-

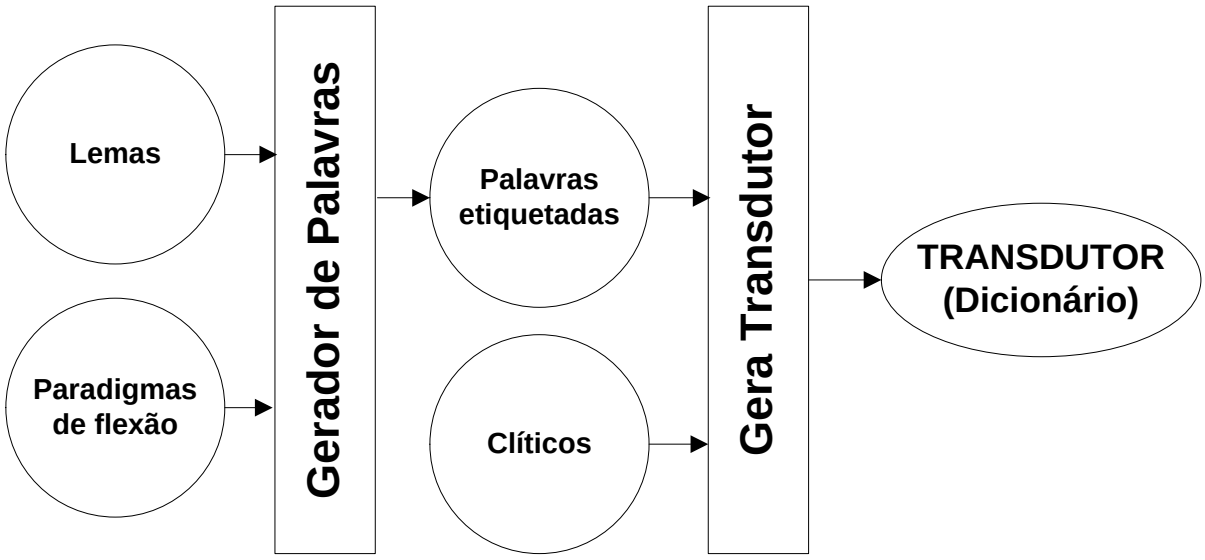

Figura 4.1: Arquitetura da primeira fase de processamento do LexMan

formações devem ser feitas nos lemas para obter uma palavra e que etiqueta deve ser associada. Existem 2 tipos de sintaxe para associar um lema a um paradigma. Este é a primeira sintaxe:

<lema> ((custo)? ([cat+subcat (,cat+subcat)*])* (paradigma)+ (<radical>)+)+

Na primeira sintaxe apresentada, indica-se na primeira coluna o lema da palavra. O custo é opcional. Assume-se que é igual a zero quando não está definido. As categorias e sub-categorias da palavra são opcionais. A seguir, coloca-se uma ou mais referências de paradigmas. A seguir aos paradigmas, indica-se um ou mais radicais da palavra. Na segunda sintaxe, o paradigma é substituído pela etiqueta morfológica e o radical pela forma superficial:

<lema> ((custo)? ([cat+subcat (,cat+subcat)*])* etiquetaMorfológica <formaSuperficial>)+

O processo de geração é exemplificado através da geração de todas as palavras possíveis que têm o lema *professor*. Esta é a entrada no dicionário de lemas, que respeita a primeira sintaxe:

<professor> [Nc] FlN62 <professor>

Neste exemplo, o lema *professor* é um nome comum, associado ao paradigma FlN62. Apresenta-se uma entrada do dicionário para a conjunção coordenativa *seja*, que respeita a segunda sintaxe.

<seja> [Cc] .....= <seja>

No dicionário dos lemas, é possível usar o ponto de interrogação para indicar que há carateres opcionais. É o caso da entrada do nome *ator*, cujo lema é representado na atual ortografia, enquanto a forma superficial admite também a variante com a consoante surda *actor*. Deste modo, é possível utilizar o dicionário com textos escritos nas duas ortografias.

<ator> [Nc] FlN14 <ac?tor>

Cada entrada no ficheiro de paradigmas deve respeitar a seguinte sintaxe:

```
paradigma <lemaExemplo>
    (etiquetaMorfológica número (<carateresRem><carateresAdd>)+ (CliticFlag)?)+
$
```

O campo paradigma representa o nome do paradigma. A seguir, indica-se o exemplo de um lema ao qual o paradigma possa ser aplicado. A etiqueta morfológica é a restante etiqueta, depois da categoria e sub-categoria atribuída à palavra gerada. O campo 'número' permite saber que radical irá sofrer as alterações. <carateresRem> representa os carateres que têm de ser removidos no final do radical. <carateresAdd> representa os carateres a adicionar ao final do radical. <CliticFlag> é um campo opcional, só usado no caso dos verbos, que indica que categoria de clíticos é que tem de ou pode ser associada à forma verbal. A entrada no ficheiro dos paradigmas do paradigma FlN62, associado ao lema *professor* é a seguinte:

```
FlN62 <idealizador>
    ...smn.== 0 <><>
    ...sfn.== 0 <><a>
    ...pmn.== 0 <><es>
    ...pfn.== 0 <><as>
$
```

A partir do lema *professor* e usando o paradigma FlN62 apresentado, são geradas 4 palavras diferentes: *professor*, *professora*, *professores* e *professoras*. A primeira transformação não modifica o lema, resultando na palavra *professor*. Na segunda transformação, adiciona-se a letra *a* ao final do radical, o que produz a palavra *professora*. A terceira e quarta transformação produzem as palavras equivalentes, no plural. Cada palavra gerada tem a sua própria etiqueta. Neste exemplo, as variações são ao nível do género e do número.

A geração de palavras concretiza-se com a criação de ficheiros em que cada linha contém informação sobre uma palavra gerada. Apresenta-se de seguida a sintaxe respeitada em cada linha desses ficheiros:

surface lema etiqueta peso

Cada linha é composta por 4 colunas, separadas por tabulações. Na primeira coluna, encontra-se a palavra gerada, denominada por *surface* (forma superficial). Corresponde à palavra que se quer analisar e que pode ser encontrada num texto. Na segunda coluna, é indicado o lema correspondente. A etiqueta morfossintática da palavra gerada situa-se na terceira coluna. Na quarta e última coluna, encontra-se um valor numérico correspondente ao peso que a palavra gerada tem no dicionário. Existem palavras que podem ser geradas várias vezes nos ficheiros, associadas a etiquetas diferentes. O peso é utilizado para definir prioridades em palavras ambíguas. Apresenta-se a seguir as linhas criadas para cada palavra gerada a partir dos lemas *professor* e *ator*:

```
ac?tor    ator    Nc...smn.==   0
ac?triz   ator    Nc...sfn.==   0
ac?tores  ator    Nc...pmn.==   0
ac?trizes ator    Nc...pfn.==   0
professor professor Nc...smn.== 0
```

professora professor Nc...sfn.== 0
professores professor Nc...pmn.== 0
professoras professor Nc...pfn.== 0

As etiquetas do LexMan são sempre compostas por 11 campos. Cada um dos campos indica uma característica da palavra. Essa informação é usada durante o processamento da cadeia STRING, após a realização da análise morfológica.

Devido à necessidade de complementar as formas verbais com os clíticos, as palavras geradas são divididas em duas categorias: verbos e não-verbos. As formas verbais geradas são também divididas em 17 sub-categorias. Esta sub-divisão é feita de acordo com o tipo de clíticos aos quais as formas verbais podem ser associadas.

4.1.1.2 Construção do Transdutor do Dicionário

Depois de se ter gerado todas as formas verbais e não-verbais, inicia-se a construção de um transdutor. Isso é feito gerando um transdutor para cada sub-categoria das formas verbais e um transdutor para a categoria dos não-verbos. Usando as operações disponibilizadas pelas bibliotecas *FSM* e *OpenFst*, esses transdutores são combinados para formar o transdutor final, correspondente ao dicionário.

A construção do dicionário é composta por três fases. A primeira fase é a análise das palavras geradas anteriormente para criar ficheiros onde é guardada toda a informação morfológica de cada palavra e criar as tabelas de símbolos dos transdutores.

Esta análise permite registar e guardar em ficheiros todos os lemas das palavras analisadas. São criados 27 ficheiros de lemas para cada uma das categorias (verbos e não-verbos). Os lemas a guardar nesses ficheiros são indexados consoante a primeira letra do lema. 26 desses ficheiros são usados para guardar lemas cuja primeira letra faz parte do alfabeto. O 27º ficheiro é utilizado para guardar os lemas que começam por qualquer outra letra (p.e. letras com acentos). As etiquetas ligadas a palavras que pertencem à categoria dos verbos e não-verbos são guardadas em 2 ficheiros (um para cada categoria). Na tabela de símbolos, são criadas entradas que permitem indexar os lemas e as etiquetas, consoante o ficheiro e a linha desse ficheiro onde estão guardados.

A segunda fase consiste na construção dos transdutores que formam o dicionário. As palavras geradas são percorridas e cada forma superficial passa a ter um caminho entre o estado inicial e o estado final do transdutor. Também se incluem informações que permitem obter o lema e a etiqueta da palavra analisada durante a segunda fase do processamento do LexMan. A figura 4.2 representa alguns exemplos de palavras pertencentes à categoria dos não-verbos, no respetivo transdutor.

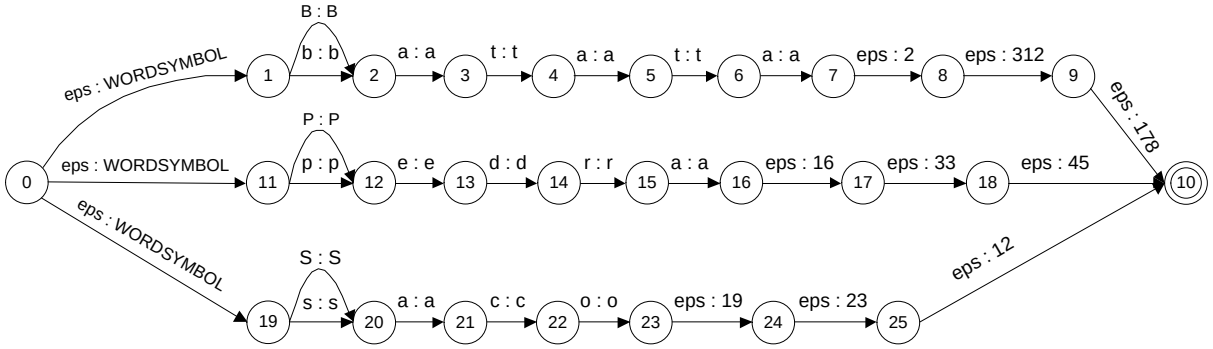

Figura 4.2: Construção de parte do transdutor dos não-verbos a partir das palavras geradas

As palavras exemplificadas são *batata*, *pedra* e *saco*. Estas três palavras pertencem à categoria das não-verbos. O transdutor representado na figura 4.2 é uma representação dessas três palavras no transdutor

construído para a categoria dos não-verbos. Como os verbos podem ser complementados com clíticos, o processamento que permite obter as etiquetas de um verbo é diferente do processamento que permite obter as etiquetas de palavras que não são verbos. Por isso, no transdutor, cada palavra é antecedida de uma transição com um identificador para saber a que transdutor pertence.

Cada letra de uma palavra corresponde a uma transição entre dois estados. Quando se constroem as transições que formam uma palavra, considera-se sempre que a primeira letra da palavra pode começar por uma maiúscula ou por uma minúscula. Os estados, as transições e as suas propriedades vão sendo criados à medida que é necessário. Cada transdutor é definido num ficheiro de texto, respeitando o formato descrito na secção 3.1.

Cada letra de uma palavra é colocada no símbolo de entrada e saída da respetiva transição. Depois de criada a transição da última letra da palavra, criam-se transições especiais, que vão até ao estado final. Estas transições especiais permitem obter a informação morfológica sobre a palavra, durante a análise morfológica. Na primeira transição especial, encontra-se um número inteiro que varia entre 1 e 27. Esse número é um índice que indica em que ficheiro foi guardado o lema da palavra. A segunda transição especial permite saber em que linha desse ficheiro foi guardado o lema. A terceira transição especial indica em que linha do ficheiro das etiquetas foi guardada a etiqueta dessa palavra. Esses três valores são usados durante a análise morfológica. A figura 4.2 exemplifica a construção do transdutor dos não-verbos. O processo de construção para cada uma das sub-categorias dos verbos é equivalente.

4.1.1.3 Clíticos

Durante a construção do transdutor do dicionário, usam-se os clíticos para complementar as formas verbais. Uma forma verbal pode ter entre zero e dois clíticos. Existem várias categorias de clíticos. Para cada uma delas, constrói-se um transdutor. A construção desses transdutores é diferente da dos transdutores dos verbos e não-verbos. A figura 4.3 representa um exemplo de construção desses transdutores, em que existem dois clíticos, seguidos da terminação do verbo.

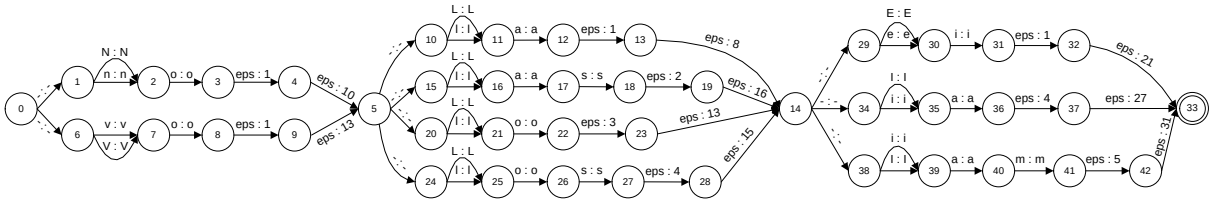

Figura 4.3: Construção de parte de um dos transdutores de clíticos

Existem estados intermédios onde acabam os caminhos de um clítico. Esses mesmos estados são usados como estado inicial do clítico seguinte ou da terminação do verbo. A grande diferença relativamente aos transdutores dos verbos e não-verbos é que depois das transições das letras que compõem um clítico, só existem duas transições especiais. Esta diferença deve-se ao facto dos lemas dos clíticos serem guardados num único ficheiro, por isso, ser desnecessário ter uma transição que permita indexar o ficheiro dos lemas dos clíticos, visto que se trata sempre do mesmo.

Após a construção do transdutor para os não-verbos, para cada uma das sub-categorias dos verbos e para cada categoria dos clíticos, é necessário criar o transdutor final.

4.1.1.4 Junção dos Transdutores

Na terceira fase de construção do dicionário, os transdutores construídos são combinados para formar o transdutor final do dicionário. A sequência de comandos a aplicar está definida num *shell script*. Os ficheiros de texto criados durante a construção dos transdutores são compilados usando a operação

Compile da biblioteca *FSM*. O transdutor dos não-verbos e todos os transdutores das formas verbais são determinizados. A operação de determinização serve para juntar os estados equivalentes do transdutor num único estado. Após a determinização, todos os transdutores são convertidos para transdutores da biblioteca *OpenFst*. Os transdutores são primeiro convertidos para ficheiros de texto usando a operação *print* e compilados usando a respetiva operação da biblioteca *OpenFst*. A partir desse momento, começa-se a combinar os transdutores entre si. Os transdutores dos clíticos são concatenados aos transdutores das formas verbais às quais devem ficar ligados. Após a concatenação das formas verbais com os clíticos, usa-se a operação da união para unir os transdutores resultantes das concatenações, criando-se assim um transdutor que representa todos os verbos do dicionário. O resultado da execução destas duas operações está representado na figura 4.4.

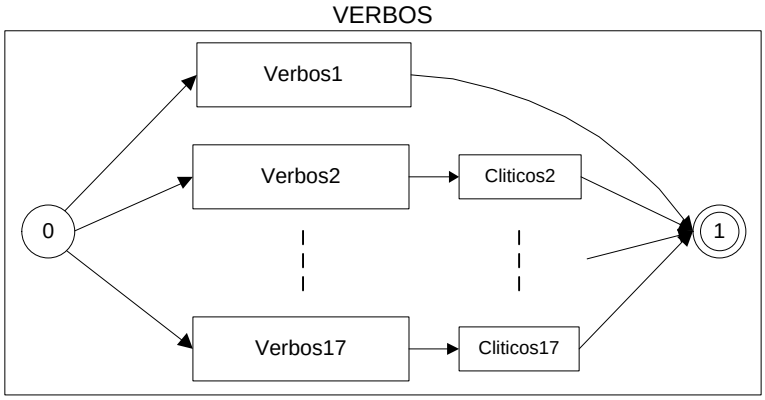

Figura 4.4: União dos transdutores das formas verbais após concatenação dos clíticos

Obtendo o transdutor que representa os verbos do dicionário, usa-se a operação da união para unir os verbos aos não-verbos e a um transdutor suplementar. Esse transdutor suplementar contém clíticos como *ei-los* e *ei-las* que não são associados a nenhuma forma verbal. Como se trata de exceções, foram tratados de uma maneira diferente da dos outros clíticos. São adicionados ao dicionário depois dos transdutores dos verbos e não-verbos estarem construídos. O transdutor obtido após essas uniões representa o dicionário de palavras a usar durante a análise morfológica. A figura 4.5 representa o transdutor da construção do dicionário.

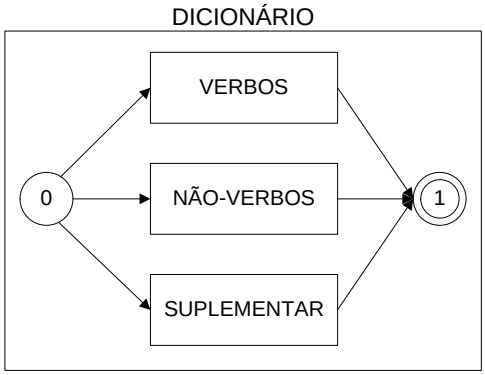

Figura 4.5: União final que permite obter o transdutor do dicionário

4.1.2 2ª Fase do Processamento - Atribuição de Etiquetas

A figura 4.6 contém uma representação da arquitetura da segunda fase de processamento efetuada pelo LexMan.

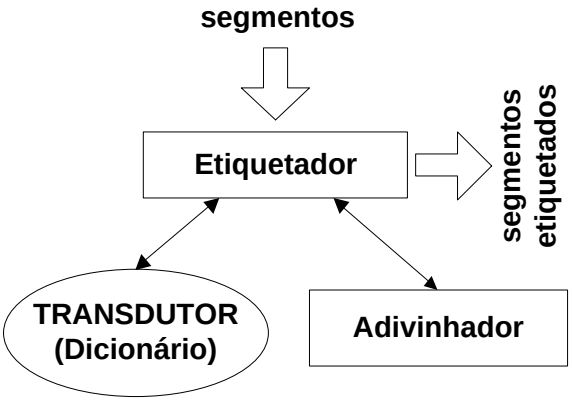

Figura 4.6: Arquitetura da segunda fase de processamento do LexMan

A segunda fase de processamento permite obter a classificação morfológica das palavras a analisar. O segmentador cria uma cadeia com todos os segmentos encontrados, colocando etiquetas nos segmentos identificados pelas expressões regulares que tratam de sequências de texto especiais. Essa cadeia de segmentos é enviada para a entrada do LexMan e recebida pelo módulo Etiquetador. O seu processamento está ilustrado na figura 4.7.

O etiquetador analisa a cadeia e separa os segmentos. Cada segmento que não foi classificado pelo segmentador é transformado num transdutor e é composto com o transdutor produzido na primeira fase de processamento. Se o resultado da composição for não vazio, tal significa que se encontrou uma ou mais etiquetas morfossintáticas. A composição retorna um transdutor constituído pelos caminhos do dicionário que emparelharam com o segmento. A figura 4.8 representa um exemplo do resultado da composição da palavra *amo* com o dicionário.

O transdutor resultante da composição é percorrido. Todos os caminhos são analisados. As transições especiais colocadas a seguir às formas superficiais são utilizadas para obter os lemas e as etiquetas necessárias dos respetivos ficheiros, criados durante a construção do dicionário. Esta informação é associada ao segmento.

Quando o resultado da composição for vazio, é porque o segmento analisado não é reconhecido pelo transdutor. Para a composição entre o segmento e o dicionário não retornar um resultado vazio, só a primeira letra da palavra é que pode estar escrita em maiúscula, tal como demonstrado na figura 4.2. O segmento que foi composto com o dicionário é analisado para verificar se a palavra é somente constituída por letras maiúsculas. Caso seja, cria-se um novo transdutor para o segmento em que só a primeira letra é maiúscula. Este novo transdutor é composto com o dicionário. Se o resultado desta segunda composição não for vazio, obtém-se as etiquetas, tal como descrito anteriormente. Se for vazio, considera-se que não é possível encontrar uma etiqueta morfossintática através da composição do segmento com o dicionário. Quando isto acontece, usa-se o módulo Adivinhador. Com base numa análise às terminações da palavra e nos resultados daí obtidos, este módulo propõe etiquetas morfossintáticas para o segmento desconhecido.

Depois de se obter as etiquetas para os segmentos, é feita uma análise com o objetivo de dividir o texto em frases. Na saída do LexMan, são colocados todos os segmentos recebidos e as etiquetas morfossintáticas de cada segmento. Os segmentos que tiveram de ser analisados pelo módulo Adivinhador podem ser guardados num ficheiro para facilitar a depuração das palavras que foram consideradas desconhecidas. Para o exemplo da palavra *amo*, esta seria a saída do LexMan:

```
Sent[0] - word[0]: |amo|                    [0,2]        POS->[amar]V.ip1s=...= [amo]Nc...smn.==
```

Os segmentos encontrados são colocados entre '|'. Na cadeia STRING, é importante saber em que

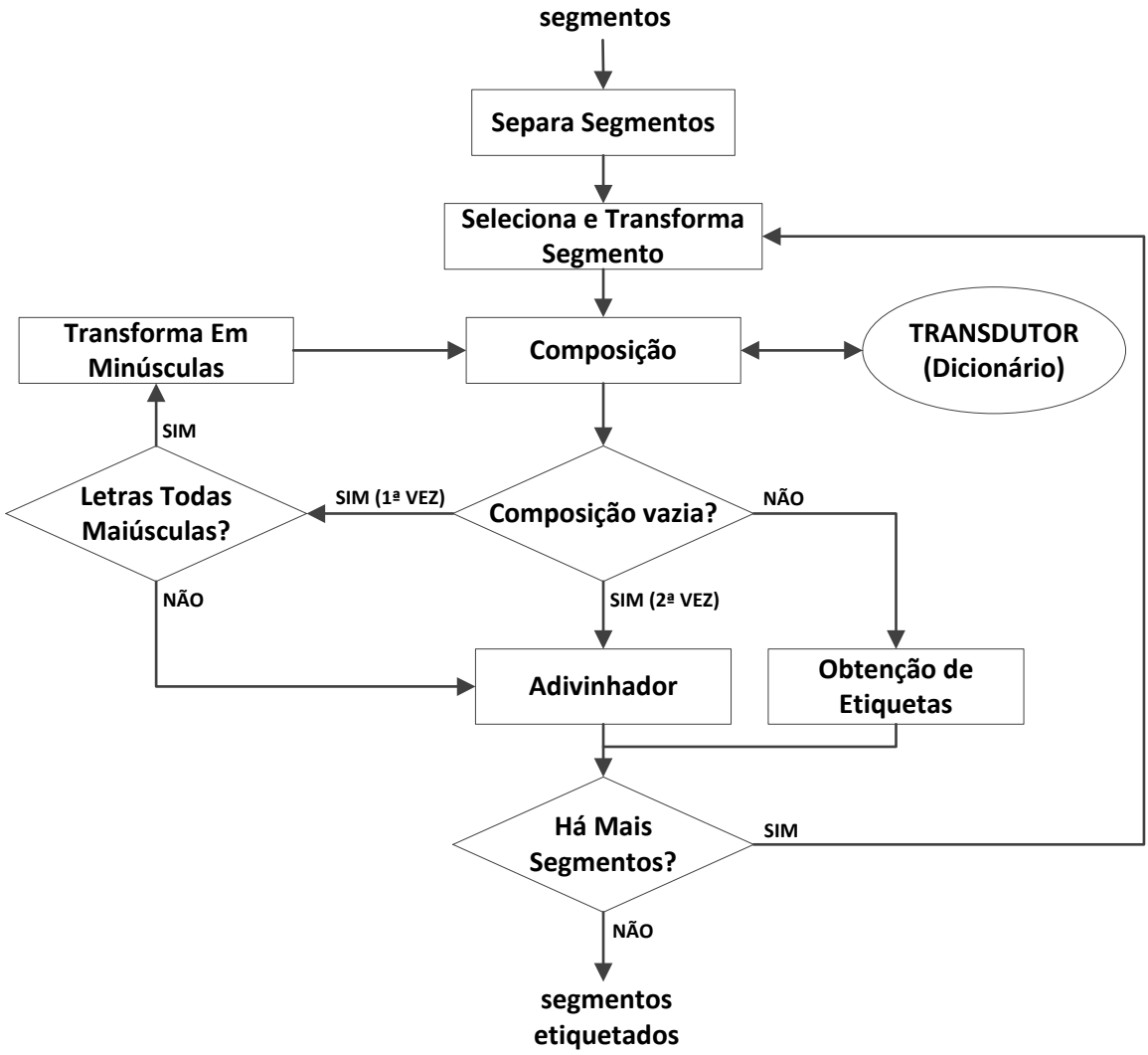

Figura 4.7: Processamento do Etiquetador

posição do texto se encontrou o segmento. Por isso, entre parênteses retos e para cada segmento, indica-se a posição da primeira e última letra do segmento relativamente ao texto analisado. Depois, coloca-se a lista dos lemas e das etiquetas morfossintáticas encontradas. Quando existem vários segmentos, eles são colocados em diferentes linhas da saída do LexMan. As frases são identificadas com o texto *Sent*, seguido de um índice entre parênteses retos que indica a posição da frase no texto original. Cada segmento da frase é precedido do texto *word*, seguido do índice entre parênteses retos da posição do segmento na frase a que pertence. Isso é demonstrado no seguinte exemplo, resultado da análise do LexMan para a cadeia de texto "*O Manuel comprou um carro. O carro é azul.*":

```
Sent[0] - word[0]: |O|          [0,0]     POS->[eu]Pp..3sm.a== [o]Td...sm...=
Sent[0] - word[1]: |Manuel|     [2,7]     POS->[Manuel]Np...smn.==
Sent[0] - word[2]: |comprou|    [9,15]    POS->[comprar]V.is3s=...=
Sent[0] - word[3]: |um|         [17,18]   POS->[um]Ti...sm...= [um]Pi..=sm.===
Sent[0] - word[4]: |carro|      [20,24]   POS->[carro]Nc...smn.==
Sent[0] - word[5]: |.|          [25,25]   POS->[.]O.........

Sent[1] - word[0]: |O|          [27,27]   POS->[eu]Pp..3sm.a== [o]Td...sm...=
```

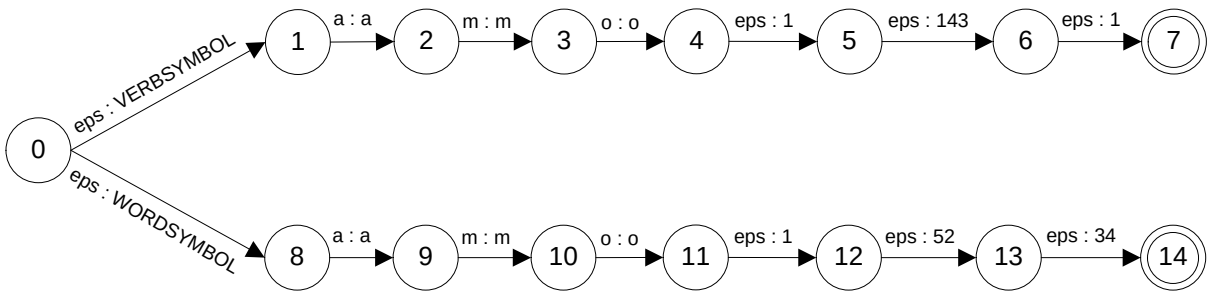

Figura 4.8: Resultado da composição do segmento *amo* com o transdutor do dicionário

Sent[1] - word[1]: |carro|	[29,33]	POS->[carro]Nc...smn.==
Sent[1] - word[2]: |é|	[35,35]	POS->[ser]V.ip3s=...=
Sent[1] - word[3]: |azul|	[37,40]	POS->[azul]A....s=n.== [azul]Nc...smn.==
Sent[1] - word[4]: |.|	[41,41]	POS->[.]O.....

Este é o formato da saída apresentada ao utilizador. A passagem de resultados entre os módulos da cadeia STRING é feita através do formato XML. Após se ter encontrado os segmentos, os respetivos lemas e etiquetas morfossintáticas, constrói-se uma estrutura XML que contém os resultados da análise do LexMan. Essa estrutura é enviada para o módulo seguinte da cadeia STRING, o RuDriCo, que analisa a estrutura para efetuar o seu processamento.

4.2 Regras de Recomposição

Um dos objetivos deste trabalho era transferir regras de descontração[1] implementadas no RuDriCo para o segmentador. Chegou-se à conclusão que não seria uma solução muito adequada. Só se poderiam transferir as regras de descontração que não necessitam de contexto. Das 270 regras de descontração implementadas no RuDriCo, só 67 regras poderiam ser transferidas para o LexMan, o que representa aproximadamente 24,81% do total das regras de descontração.

A análise das regras de descontração existentes no RuDriCo revelou que a presença de contexto não é o único fator importante na aplicação das regras. Mesmo considerando somente as regras que não dependem do contexto, concluiu-se que a ordem pela qual essas regras são aplicadas também era importante. No RuDriCo, é possível definir a ordem pela qual as regras são aplicadas. As regras são organizadas em camadas e cada camada pode conter uma ou mais regras. A ordem de aplicação das regras depende da camada a que pertencem. Dentro da mesma camada, a ordem pela qual as regras são aplicadas depende da ordem pela qual são definidas. As camadas também definem a prioridade entre as regras. Cada camada tem um número.

Durante a aplicação das regras de descontração no RuDriCo, os elementos que são descontraídos são logo identificados e etiquetados. Não se pretende que o novo segmentador construído seja capaz de etiquetar os elementos da linguagem que identifica, ao contrário do antigo segmentador. Essa responsabilidade é atribuída ao analisador morfológico.

Em suma, chegou-se à conclusão de que o pequeno número de regras (67) que seria possível transferir para o segmentador não justifica o aumento da complexidade que teria de ser adicionado ao segmentador, de modo a poder aplicar algumas regras de descontração durante o processo de segmentação.

As regras de contração que estavam implementadas no RuDriCo e que não dependiam do contexto

[1]nota: Usamos o termo "'descontração" no sentido preciso de "processo contrário" por uma questão de comodidade e economia, a fim de evitar outras formas de indicar esta operação. Estamos, naturalmente, consciente do significado que a palavra tem na língua geral.

(36.299) foram transferidas para o dicionário do LexMan. As regras de contração que dependem do contexto respeitam a seguinte sintaxe:

camada> |contexto_esquerdo| antecedente |contexto_direito| :> consequente .

Uma regra de contração que depende do contexto é composta por itens. Cada item emparelha com um segmento. Os padrões das regras dos contextos são opcionais. Quando os padrões especificados pelo antecedente e pelos contextos emparelham com sucesso com uma sequência de segmentos, o antecedente é substituído por aquilo que se encontra no consequente. Esta é uma regra de contração do RuDriCo em que é necessário contexto:

```
0> |[lemma='um']|
   [surface='sem'],
   [surface='número']
   |[lemma='de']|
  :>
   [surface=@@+,lemma='sem-número',CAT='nou',SCT='com',GEN='m',NUM='s'].
```

Esta regra pertence à camada zero do RuDriCo. Usando notação própria do RuDriCo, esta regra aplica os padrões para verificar se existe um segmento cujo lema é *um*, seguido de 2 segmentos com as formas superficiais *sem* e *número*. E caso a sequência acabe com um segmento que tem o lema *de*, considera-se que todos os padrões emparelharam com um segmento e o antecedente é substituído pelo consequente. Neste caso, os segmentos com as formas *sem* e *número* são unidos num só segmento. A regra define quais são as alterações a fazer nas propriedades do novo segmento a nível do lema e das componentes que constituem a sua informação morfológica.

As regras de contração que dependem do contexto utilizam um conjunto de informações que não estão disponíveis no texto recebido na entrada do LexMan. É por isso que só se transferiram para o LexMan as regras que não dependem do contexto. A passagem dessas regras para o LexMan resultou na expansão do processo de geração de palavras do LexMan. Estas alterações são descritas na secção 4.2.1.

4.2.1 Expansão da Geração de Palavras do LexMan

As regras de contração permitem juntar num só segmento, diferentes palavras. Essas palavras são separadas por espaços em branco ou por hífenes. A esse conjunto de palavras que formam um único segmento, atribui-se a designação de **composto**.

Na Língua Portuguesa, os compostos podem sofrer transformações que dão origem a outras palavras. O composto *sociedade científica* por exemplo, pode ser escrito no plural, o que resulta no composto *sociedades científicas*. Por isso, os compostos tiveram de ser incluídos no processo de Geração de Palavras do LexMan. Foram criados ficheiros para o dicionário de lemas e paradigmas de flexão, específicos para os compostos, tendo sido definido também uma sintaxe própria para esses ficheiros. Esta é a sintaxe escolhida para os dicionários dos lemas:

<lema> (([cat+subcat (,cat+subcat)*])* etiquetaMorfológica <formaSuperficial1 ><formaSuperficialN>)+

Esta sintaxe é semelhante àquela apresentada na secção 4.1.1.1. Neste caso, são indicadas N formas superficiais que correspondem à sequência de formas superficiais em que se divide o composto. Através

de uma letra maiúscula, indica-se a obrigatoriedade, opcionalidade ou a proibição de ligar os constituintes de um composto com hífen. Estas são as letras maiúsculas definidas na sintaxe:

E(spaço) - só aceita como separador dos constituintes o espaço
H(ífen) - só aceita como separador dos constituintes o hífen
O(pcional) - aceita como separador dos constituintes o hífen ou o espaço

De seguida, apresenta-se uma entrada do dicionário de lemas que foi definida para o composto *anjo da guarda* e que respeita a sintaxe apresentada:

<anjo da guarda> [Nc] ...smn.== E<anjo><da><guarda>

Os paradigmas das palavras individuais que existiam no dicionário servem de base para as transformações a aplicar nos compostos. Estas transformações dependem das categorias das palavras que fazem parte do composto. Foram definidos 9 paradigmas que permitem tratar das transformações necessárias, para cada caso. Na sintaxe definida, o nome do paradigma substitui a etiqueta morfológica. Esta é a entrada do dicionário para o composto *amigo pessoal*, que fica associado ao paradigma *Comp1*:

<amigo pessoal> Comp1 E<amigo><pessoal>

O paradigma *Comp1* foi definido para tratar das transformações a aplicar aos compostos constituídos por um par NOME-ADJETIVO, como é o caso do composto *amigo pessoal*. Os compostos gerados com a associação do lema *amigo pessoal* ao paradigma *Comp1* juntam-se ao conjunto de palavras geradas a partir dos dicionários de lemas e paradigmas do LexMan. Os compostos são colocados em ficheiros, respeitando a sintaxe que foi apresentada na secção 4.1.1.1:

amigo pessoal amigo pessoal Nc...smn.== 0
amiga pessoal amigo pessoal Nc...sfn.== 0
amigos pessoais amigo pessoal Nc...pmn.== 0
amigas pessoais amigo pessoal Nc...pfn.== 0

As transformações aplicadas ao lema *amigo pessoal* dão origem a 4 compostos. As diferenças entre os compostos ocorrem ao nível do género e do número.

4.2.2 Inclusão dos Compostos no Dicionário

Tal como mencionado anteriormente, as palavras que fazem parte de um composto podem ser separadas por espaços em branco e por hífenes. Na Língua Portuguesa, o espaço em branco é considerado um separador de palavra, ou seja, é um caráter que provoca a rutura de segmentos. Mas quando um espaço em branco faz parte de um composto, não deve haver rutura. O composto deve formar um único segmento. Por isso, durante a construção do caminho de um composto no transdutor do dicionário, o espaço em branco é tratado como um caráter normal. A figura 4.9 representa o caminho a adicionar ao transdutor do dicionário para a forma superficial *amigo pessoal*, com as transições que permitem obter a informação morfológica do composto.

Como se pode ver na figura 4.9, existem duas transições para a primeira letra de palavra pertencente ao composto: uma para a letra em minúscula, outra para a letra em maiúscula. Desta forma, o LexMan consegue identificar as sequências *amigo pessoal*, *Amigo Pessoal*, *Amigo pessoal* e *amigo Pes-*

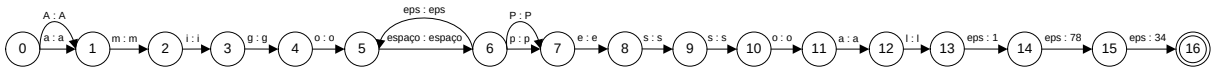

Figura 4.9: Caminho do composto *amigo pessoal*

soal. O símbolo do espaço em branco, tal como os outros carateres, é colocado na entrada e na saída da transição. O número de espaços em branco a separar as palavras que fazem parte de um composto não tem influência na identificação. Por isso, os caminhos dos compostos que contêm espaços em branco têm a particularidade de ter uma transição com o símbolo *epsilon* colocado na entrada e na saída da transição. Esta transição permite que haja um ou mais espaços em branco a separar duas palavras de um composto. A forma como o espaço em branco é tratado quando funciona como separador de palavra é explicada na secção 4.3.3.

4.3 Segmentação e Análise Morfológica

Antes do início deste trabalho, a segmentação e a análise morfológica eram realizadas em dois módulos. Neste trabalho, criou-se um novo segmentador com transdutores e duas novas arquiteturas para o LexMan que permitiram juntar a segmentação e análise morfológica num único módulo. Para determinar qual é a melhor, essas duas soluções foram implementadas e avaliadas. As duas soluções são apresentadas na figura 4.10.

Do lado esquerdo da figura 4.10, encontra-se a 1ª solução implementada. Esta solução foi elaborada com base na utilização da operação *ShortestPath*. Nesta solução, descrita na secção 4.3.4, a segmentação e a análise morfológica são realizadas usando dois transdutores.

Do lado direito da figura 4.10, apresenta-se a 2ª solução implementada. Esta solução foi implementada com o objetivo de juntar a segmentação e a análise morfológica numa só etapa, usando um único transdutor. Isso foi conseguido através da utilização da operação *Prune*. Esta solução é descrita na secção 4.3.5.

O novo LexMan recebe o texto que se pretende segmentar e etiquetar. Esse texto é primeiro transformado num transdutor durante a normalização. Esta etapa é comum às duas soluções implementadas e é descrita na secção 4.3.1.

A seguir à normalização, realiza-se a segmentação. Construiu-se um novo segmentador usando transdutores, com o intuito de substituir o segmentador da arquitetura original do LexMan, que se baseava em expressões regulares para segmentar texto. A estrutura do segmentador é apresentada na secção 4.3.3.

O processamento do LexMan acaba com a obtenção das etiquetas morfossintáticas de cada segmento. Esse processo foi modificado, relativamente à solução original do LexMan, para se adaptar à especificação de cada solução. As alterações efetuadas são descritas nas secções 4.3.4 e 4.3.5.

4.3.1 Normalização

Antes de tentar obter as etiquetas de cada segmento do texto, é necessário transformar o texto, para que esteja de acordo com a especificação dos módulos seguintes. Esta etapa denomina-se **normalização**.

Constrói-se um transdutor equivalente ao texto recebido na entrada, efetuando ao mesmo tempo, substituições de carateres. Apresentam-se alguns exemplos de substituições definidas no LexMan:

<\n> : < >
< > : < >
<≪> : <"">

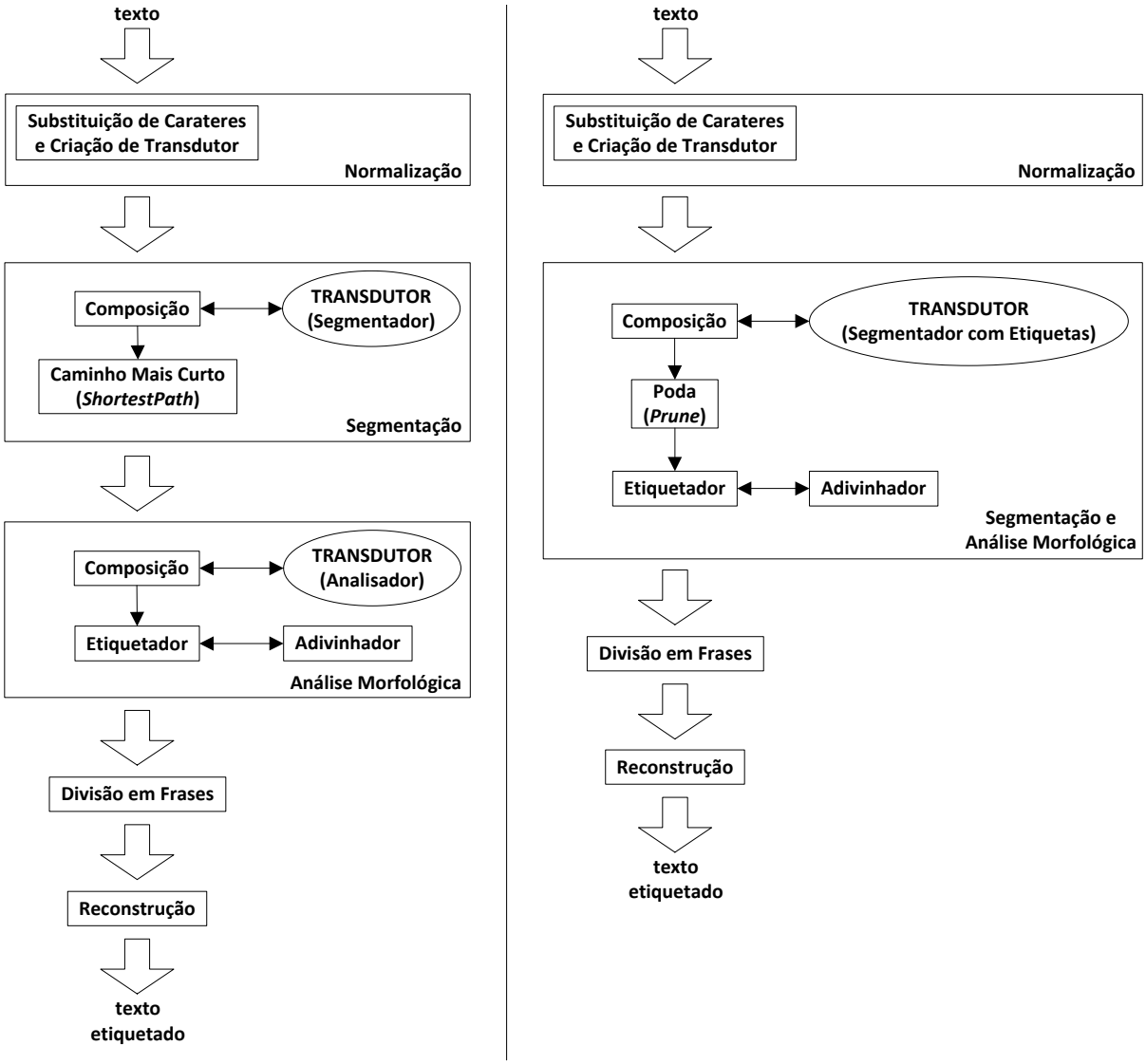

Figura 4.10: Arquiteturas das duas soluções implementadas

<≫> : <"> <—> : <-> Os carateres são rodeados por '<' e '>' e separados por ':'. O caráter especificado à esquerda de ':' indica o caráter que se pretende substituir. O caráter especificado à direita de ':' indica o caráter pelo qual ele tem de ser substituído. O texto na entrada do LexMan é analisado, caráter a caráter. Para cada caráter, verifica-se se é necessário substituí-lo.

Um caráter pode ser substituído por diferentes razões. A uniformização é uma delas. Existem vários tipos de carateres de espaço: espaço em branco, tabulações e mudanças de linha. Como não é necessário trabalhar com diferentes tipos de carateres de espaço, decidiu-se uniformizá-los e, por isso, cada tabulação e mudança de linha é substituída por um espaço em branco. Seria possível representar uma sequência de um ou mais carateres de espaço por um único espaço em branco, no entanto, é importante que o texto normalizado mantenha o mesmo número de carateres que o texto recebido na entrada do LexMan, para que não se perca a posição de cada palavra na cadeia de carateres de entrada. Esta posição é importante, por exemplo, quando se pretende adicionar marcas no texto original. Por exemplo no HAREM, as entidades mencionadas eram balizadas com etiquetas XML. É por isto que um espaço em

branco substitui um único caráter de tabulação e mudança de linha, mesmo que haja vários seguidos. As aspas e os travessões também são carateres que mereceram especial atenção. Os textos testados continham diversas formas de representar aspas e travessões. Decidiu-se que estes casos também deviam ser uniformizados. As aspas ficam todas com a mesma representação. Os travessões são transformados em hífenes.

Outra razão para substituir um caráter é o facto de ele não corresponder a uma "letra". Os carateres de controlo são um exemplo deste tipo de carateres. Eles podem aparecer em textos, mas como não têm nenhum representação, é necessário substituí-los por um outro caráter, por forma a que a análise do texto se realize sem problemas. Decidiu-se que esses carateres seriam substituídos por espaços.

O texto analisado pelo LexMan pode conter carateres que são específicos de línguas estrangeiras (por exemplo, carateres chineses). Como o LexMan trabalha com um dicionário da língua portuguesa, os carateres específicos de línguas estrangeiras são considerados desconhecidos e, por isso, têm de ser tratados de uma forma diferente da dos outros carateres durante a normalização. Todos esses carateres são substituídos por um caráter especial, previamente definido, para que seja possível saber que, naquela posição do texto, existe um caráter desconhecido da língua portuguesa. Como existem muitos carateres desconhecidos, não seria adequado especificar as substituições uma a uma, por cada caráter desconhecido. Decidiu-se que o conjunto de carateres desconhecidos a substituir iria ser representado na lista de substituições através da string UNKNOWNCHAR, ao qual é associado um caráter especial pelo qual cada caráter desconhecido no texto é substituído. À medida que se fazem as substituições, constrói-se um transdutor equivalente ao texto analisado, com as substituições de carateres indicadas anteriormente.

Um caráter desconhecido é substituído para facilitar o processo da segmentação e da análise morfológica, mas sempre que um caráter desconhecido é substituído, guarda-se essa informação porque eles têm de voltar a ser introduzidos no texto nas posições corretas, no fim do processo, ou seja, na saída do LexMan. Isso é feito na última etapa do processamento, denominada **reconstrução**.

4.3.2 Transformação das Expressões Regulares

A segunda etapa do processamento é a **segmentação**. O transdutor construído na normalização é composto com um segmentador, implementado com transdutores. Este novo segmentador tem a função de substituir o segmentador da arquitetura original do LexMan, por isso, "transferiram-se" as expressões regulares que existiam no antigo segmentador para o LexMan.

As expressões regulares mais simples (aquelas que não contêm a operação da união, por exemplo) foram transferidas para o dicionário de lemas, tendo definido *macros* que representam grupos de carateres. Estas são algumas das *macros* definidas:

```
%d  [0123456789]
%c  [abcdefghijklmnopqrstuvwxyzáàãâéèêíìóòõôúùç]
%C  [ABCDEFGHIJKLMNOPQRSTUVWXYZÁÀÃÂÉÈÊÍÌÓÒÕÔÚÙÛÇ]
```

Além disso, é possível usar os símbolos habitualmente usados em expressões regulares como o '+', '*', parênteses retos e chavetas. A utilização destes carateres é ilustrada nesta entrada do dicionário de lemas que permite a identificação de códigos ISBN, tais como *ISBN 9971-5-0210-0*:

```
<>    [Nc] ...===.=g   <ISBN(-10)? ?:? +%d{1,5}[ -]*%d{1,7}[ -]*%d{1,6}[ -]*[%dX]>
```

Durante a construção do transdutor do dicionário, as palavras geradas que contêm expressões regulares são analisadas de modo a que sejam inseridas todas as transições necessárias para construir um

caminho equivalente no transdutor.

O transdutor do dicionário construído após a geração de palavras identifica muitos dos elementos da língua portuguesa que eram identificados pelas expressões regulares do antigo segmentador. No entanto, algumas das expressões regulares do antigo segmentador eram demasiado complexas para serem representadas no dicionário de lemas. Estes casos foram igualmente tratados através da construção manual de transdutores, no âmbito deste trabalho/projeto.

Verificou-se que as expressões regulares que identificavam endereços **HTTP**, **IP** e de **correio eletrónico**, **referências bíblicas**, **números romanos** e **números por extenso** eram demasiado complexas para serem transferidas para o dicionário de lemas. Por isso, as expressões regulares foram convertidas manualmente para transdutores.

Foi feita uma análise às expressões regulares que precisavam de ser convertidas manualmente para transdutores. A análise revelou que as expressões regulares que identificavam endereços HTTP, de correio eletrónico e números romanos continham alguns erros. Os erros foram corrigidos, melhorando-se assim as expressões regulares que foram convertidas para transdutores. As expressões regulares a partir das quais os transdutores foram construídos estão apresentadas no anexo A.

Para cada um desses transdutores gerados manualmente foram criados exemplos de teste para avaliar o desempenho de cada um dos transdutores. Cada teste consiste numa lista de exemplos que deveriam ser corretamente identificados pelo respetivo transdutor. Nas secções seguintes, descreve-se com mais detalhes os transdutores que foram construídos manualmente.

4.3.2.1 Endereços HTTP

Os endereços HTTP eram identificados no antigo segmentador através de uma única expressão regular. O transdutor deveria ser capaz de identificar diferentes tipos de endereço: começados por *http*, *https* ou *www*, de diferentes tamanhos e com todo o tipo de carateres suscetíveis de aparecer num endereço HTTP. Construiu-se o transdutor equivalente, com 73 estados, 51 dos quais são estados finais, e 3.602 arcos.

Criou-se uma lista com 848 exemplos de endereços HTTP. A lista de exemplos foi criada a partir dos marcadores (*bookmarks*) de um utilizador. Apresenta-se, de seguida, uma pequena lista dos exemplos testados:

http://146.193.52.103/reap.pt-v0.2/
http://academic.research.microsoft.com/Author/862201
https://id.ist.utl.pt/cas/login?service=https%3A%2F%2Ffenix.ist.utl.pt%2FloginCAS.do
https://www.l2f.inesc-id.pt/wiki/index.php/Seminars
https://projectos.ist.utl.pt/
www.inesc-id.pt/
www.ist.utl.pt

Detetaram-se alguns erros, provocados pela presença de carateres menos comuns e que não eram considerados na expressão regular do antigo segmentador. A expressão regular foi corrigida e construiu-se o transdutor equivalente. Após essa correção, voltou-se a testar a lista de exemplos e desta vez, não foram detetados erros.

4.3.2.2 Endereços IP

Os endereços IP eram identificados no antigo segmentador através de uma única expressão regular. A expressão regular tinha em conta o facto de a seguir à sequência de dígitos e pontos que compõem o endereço, poder vir o caráter '/', seguido de um número que indica o número de bits utilizados. A partir

dessa expressão regular, construiu-se o transdutor equivalente, com 46 estados, 14 dos quais são estados finais, e 322 arcos.

Criou-se uma lista com 27 exemplos de endereços IP, recolhidos de páginas da *internet*. Apresenta-se de seguida, uma pequena lista dos exemplos testados:

0.0.0.0
8.20.15.1
18.101.25.153
102.168.212.226
204.17.5.0
129.5.255.2/16
192.168.183.15/24
131.108.2.1/30
7.7.7.7/8

4.3.2.3 Endereços de Correio Eletrónico

Os endereços de correio eletrónico eram identificados no antigo segmentador através de uma única expressão regular. Constatou-se que a expressão regular não identificava todos os possíveis endereços de correio eletrónico porque certos carateres, apesar de poderem aparecer num endereço de correio eletrónico, não eram tidos em conta pela expressão regular. A expressão regular foi corrigida e construiu-se o transdutor equivalente a essa nova expressão. Trata-se de um transdutor com 20 estados, 2 dos quais são estados finais, e 1.423 arcos.

Criou-se uma lista com 1.113 exemplos de endereços de correio eletrónico. A lista de exemplos foi criada a partir dos contactos eletrónicos de um utilizador. Apresenta-se de seguida, uma pequena lista dos exemplos testados:

amfv@ist.utl.pt
jorge.manuel.baptista@gmail.com
Nuno.Mamede@inesc-id.pt

4.3.2.4 Números Romanos

Os números romanos[2] eram identificados no antigo segmentador através de uma única expressão regular. Uma análise a essa expressão regular revelou que uma sequência vazia de carateres podia ser detetada através dessa expressão, o que não devia acontecer. A principal preocupação na identificação de números romanos foi garantir que seriam identificadas sequências de carateres que começavam sempre por um caráter suscetível de aparecer numa sequência de carateres pertencentes a um número romano (C, D, I, L, M, V, X), evitando assim a deteção de uma sequência vazia. Tal como acontecia anteriormente, a nova expressão regular considera unicamente a hipótese de um número romano poder ser composto inteiramente por letras. Construiu-se o transdutor equivalente a essa nova expressão, que apresenta 187 estados, 172 dos quais são estados finais, e 452 arcos.

Criou-se uma lista com 104 exemplos de números romanos. Apresenta-se de seguida, uma pequena lista dos exemplos testados:

[2] Apenas se consideraram os números romanos na ortografia atual. As antigas formas *iiij*, *R*, etc. não foram levadas em consideração.

II
ii
IV
iv
XVIII
xviii
MDCC
mdcc
MCMLXXXIV
mcmlxxxiv

4.3.2.5 Referências Bíblicas

Para a identificação das referências bíblicas, estavam definidas 11 expressões regulares auxiliares que eram depois combinadas para formar a expressão regular final. Construiu-se um transdutor equivalente para cada uma das expressões regulares auxiliares. No final, construiu-se um transdutor equivalente à expressão regular final que, através de um conjunto de operações (p.e. união, concatenação), combina os transdutores auxiliares para formar o transdutor final responsável pela identificação das referências bíblicas. O transdutor resultante tem 569 estados, 29 dos quais são estados finais, e 1.432 arcos.

Criou-se uma lista com 384 exemplos de referências bíblicas. Apresenta-se de seguida, uma pequena lista dos exemplos testados:

I Jo 12
II Crón 12-16
Mt 5,20 - 6,3
Mt 5-8,10-12,14-16
Mt 5, 12.14-16.18-20 ; 6, 1.3-5.7-10

4.3.2.6 Números por Extenso

Os transdutores para a identificação dos números por extenso foram os transdutores mais complexos de construir. Tal como nas referências bíblicas, os números por extenso eram identificados através de várias expressões regulares auxiliares. Adotou-se uma estratégia semelhante à utilizada para as referências bíblicas: construiram-se transdutores para cada uma das expressões regulares auxiliares, que foram seguidamente utilizados para construir o transdutor final.

Os números por extenso foram divididos em cinco categorias: **números cardinais**, números **enas**, **números ordinais**, **números fracionários** e números que podem ser, dependendo do contexto, **ordinários** ou **fracionários**. Construiu-se um transdutor para cada uma destas categorias.

Os números cardinais são aqueles que expressam a noção de *quantidade* ou a "cardinalidade" de um conjunto (p.e. *trinta*, *cento e vinte*, *dois milhões e vinte e cinco*). Foram construídos 10 transdutores auxiliares que foram depois utilizados na construção de um transdutor final, com 4.951 estados, 1.236 dos quais são estados finais, e 11.300 arcos.

Os números *enas* são números que tipicamente são seguidos de um sintagma preposicional introduzido pela preposição "de" (p.e. *dez milhões de habitantes*, *centenas de filmes*, *vinte dúzias de batatas*). Foi construído um transdutor auxiliar que foi utilizado na construção de um transdutor final, com 5.993

estados, 112 dos quais são estados finais, e 15.436 arcos.

Os números ordinais são aqueles que expressam a noção de *posição* (p.e. *primeiro*, *segundo*) numa série. Existem números ordinais que também podem ser fracionários porque também expressam a noção de *fração* (p.e. *quinto*, *oitavo*). Para esta categoria, construiu-se um transdutor responsável por identificar os números que só são ordinais: terminam em *primeiro*, *primeira*, *segundo*, *segunda*, *terceiro* e *terceira*. *décimo primeiro*, *octogésimo segundo* e *milionésima milésima centésima décima terceira* são exemplos de números ordinais. As expressões regulares existentes no antigo segmentador não faziam a divisão entre os números que eram somente ordinais e somente fracionários. As expressões regulares tiveram portanto de ser modificadas, para que fosse possível identificar números que eram somente ordinais. Tal como nos números cardinais, o transdutor responsável pela identificação de números ordinais foi construído com base em transdutores auxiliares (9). O transdutor construído tem 5.314 estados, 426 dos quais são estados finais, e 12.030 arcos.

Os números fracionários são aqueles que expressam a noção de *fração*. Considerou-se que os números fracionários que não são ordinais são aqueles que terminam em *terço*, *terços* ou em *avos*. *dois terços* e *vinte e cinco avos* são exemplos de números fracionários. O transdutor responsável pela identificação dos números fracionários tem a particularidade de poder ser construído a partir dos transdutores dos números cardinais, ordinais e ordinais/fracionários, para além de outros transdutores auxiliares específicos dos números fracionários. O transdutor resultante tem 6.086 estados, 6 dos quais são estados finais, e 15.087 arcos.

Os números que são ao mesmo tempo ordinais e fracionários expressam a noção de *posição* e de *fração*. As expressões regulares implementadas, específicas dos números ordinais e fracionários foram modificadas para não considerarem os números que são unicamente ordinais. Existem números ordinais/fracionários que são construídos a partir de números cardinais (p.e. *dois quintos*). Por isso, o transdutor final foi construído com base em dez transdutores auxiliares e no transdutor construído para os números cardinais, o que resultou num transdutor com 11.788 estados, 1.605 dos quais são estados finais, e 28.120 arcos.

4.3.2.7 Remoção de Sequências

O dicionário de lemas do LexMan já inclui palavras simples que representam números por extenso (p.e. *dois*, *vinte*, etc.), assim como os números romanos compostos por uma única letra. Decidiu-se que esses casos permaneceriam no dicionário e que não havia necessidade de os respetivos transdutores, construídos manualmente, identificarem estes casos, visto que causaria ambiguidade com o dicionário. Em vez de alterar as expressões regulares para que aqueles casos não fossem considerados, decidiu-se que seria mais simples usar uma operação suportada pelos transdutores: a diferença.

Para cada uma das categorias, criou-se um transdutor com os casos que tinham de ser removidos do respetivo transdutor final. Através da operação de diferença aplicada aos transdutores, conseguiu-se remover os casos que já estavam incluídos no dicionário. Na tabela 4.1, são apresentados os valores das principais caraterísticas dos transdutores, antes e depois de se ter efetuado a operação da diferença em cada um deles.

Aplicou-se a operação da diferença nos transdutores dos números romanos, cardinais, enas, ordinais e ordinais e fracionários. A análise da tabela 4.1 demonstra que o número de estados finais diminuiu em todos os transdutores após a operação. A operação diminui o número de sequências de texto identificáveis em cada um dos transdutores, o que causa uma diminuição do número de estados finais.

Nota-se também que só nos transdutores dos números ordinais e dos números ordinais e fracionários é que houve uma diferença no número de estados e de arcos. Chega-se à conclusão de que, nos outros transdutores, as sequências removidas fazem parte de outras sequências, o que impediu a remoção dos estados e dos arcos.

		Nº de Estados	Nº de Arcos	Nº de Estados Finais
Números Romanos	antes	187	452	186
	depois	187	452	172
Números Cardinais	antes	4.951	11.300	1.277
	depois	4.951	11.300	1.236
Números Enas	antes	5.993	15.436	134
	depois	5.993	15.436	112
Números Ordinais	antes	5.337	12.082	432
	depois	5.314	12.030	426
Números Ordinais/Fracionários	antes	11.810	28.175	1.664
	depois	11.788	28.120	1.605

Tabela 4.1: Caraterísticas dos transdutores antes e depois da remoção de sequências

4.3.3 Segmentador

O segmentador do LexMan tem a função de segmentar o texto que foi normalizado na etapa anterior do processamento. Foram tidos em conta vários aspetos na sua construção:

- Existem elementos da linguagem, chamados separadores, que causam rutura da segmentação;
- Existem dois tipos de rutura de segmentação: rutura à direita e à esquerda, ou somente à direita;
- Tem de existir pelo menos um separador entre duas palavras;
- Podem aparecer vários separadores seguidos;
- O segmentador deve ser capaz de identificar palavras que não estão no dicionário e sequências de carateres que não correspondem a casos identificáveis pelos transdutores construídos manualmente;
- Considera-se que a melhor segmentação é aquela que contém o menor número de segmentos.

Por isso, para além do transdutor do dicionário e dos transdutores construídos manualmente, a construção do segmentador também utiliza transdutores para identificar os vários separadores existentes. Considerou-se que existem 3 tipos de separadores: separadores de palavras (espaços em branco), os símbolos que causam rutura na segmentação à direita e à esquerda (rutura dupla), e os separadores que só causam rutura à direita.

Considera-se que os símbolos de rutura à direita e à esquerda são aqueles que, mesmo não havendo espaços em branco à volta deles, ficam separados dos carateres que possam estar à sua direita ou esquerda. Um exemplo de símbolos de rutura à direita e à esquerda são símbolos de pontuação. Na frase "*Amanhã,vai chover.*", não existem espaços a separar a vírgula das palavras *Amanhã* e *vai*. Mas a vírgula é um símbolo de pontuação, o que provoca a segmentação da sequência "*Amanhã,vai*" em 3 segmentos: "*Amanhã*", "*,*" e "*vai*". O ponto final da frase também causa rutura relativamente à palavra que o precede. Elementos como *tags* XML, dígitos e texto entre parênteses podem aparecer junto de palavras. Como estes casos devem causar uma rutura na segmentação, são considerados separadores de rutura à direita e à esquerda.

Os símbolos de rutura à direita são aqueles que, não havendo espaços em branco à sua volta, só causam rutura relativamente a palavras que estão à sua direita. Isto acontece quando o símbolo de rutura faz parte do segmento que está à sua esquerda. Por exemplo, o ponto faz parte do segmento de uma abreviatura. No exemplo "*Av.Roma*", não existem espaços em branco à volta do ponto, mas neste caso, o ponto faz parte da abreviatura de *Avenida* (*Av.*). Desta forma, só existe rutura à direita do ponto, formando os segmentos "*Av.*" e "*Roma*". Esta separação só é possível quando existe um separador a separar a abreviatura do contexto que estiver à sua esquerda. A segmentação da sequência "*daAv.Roma*"

cria os segmentos "*daAv*", "." e "*Roma*" porque "*Av*" não está separado da sequência "*da*". Como "*daAv.*" não é uma abreviatura válida, o ponto não faz parte do segmento à sua esquerda e causa rutura à esquerda e à direita. Se pelo contrário, a sequência for "*da Av.Roma*", obtêm-se os segmentos "*da*", "*Av.*" e "*Roma*".

Também se adicionou ao segmentador um transdutor capaz de identificar todas as palavras possíveis. Este transdutor permite aceitar sequências de texto compostas por qualquer caráter pertencente aos blocos *Unicode Basic Latin* e *Latin-1 Supplement* da tabela de codificação UTF-8, à exceção dos carateres considerados separadores de palavras.

O hífen e o *underscore* também tiveram um tratamento especial. As sequências de texto identificadas por este transdutor não podem ter um hífen ou um *underscore* no início e/ou no final da sequência de texto, porque, nesses casos, são considerados separadores. Este transdutor é usado na deteção de palavras desconhecidas e palavras que contêm erros ortográficos, permitindo identificar palavras que não se encontram no dicionário nem são reconhecidos pelos transdutores construídos manualmente.

Após a segmentação, é importante saber qual dos transdutores é que conseguiu identificar os segmentos que compõem o texto. Além disso, há sequências de carateres ambíguas porque podem ser identificadas por diferentes transdutores. O transdutor responsável por identificar todas as palavras possíveis também é capaz de identificar palavras que estão no dicionário. A palavra *batata*, por exemplo, é uma palavra do dicionário mas também pode ser identificada pelo transdutor que identifica todas as palavras possíveis. Por causa deste tipo de ambiguidade, é necessário atribuir prioridades aos transdutores. O dicionário deve ter uma prioridade mais elevada. Neste caso, pretende-se que *batata* seja identificada como uma palavra pertencente ao dicionário. Outros exemplos de casos ambíguos são os números romanos. A sequência de carateres *vi* pode ser um número romano mas também pode ser uma forma verbal do verbo *ver*. Considerou-se que o dicionário tem prioridade sobre todos os outros transdutores na identificação dos segmentos, sempre que existe este género de ambiguidade.

Para resolver estes dois problemas, adicionou-se uma transição no início de cada transdutor. As transições adicionadas a cada um dos transdutores não consomem nenhum caráter do texto, por isso, o símbolo *epsilon* é colocado na entrada da transição. A exceção são os espaços em branco que funcionam como separadores de palavras. Como eles são descartados na análise morfológica, optou-se por colocar o identificador e o custo na própria transição destinada à sua identificação. O identificador é sempre colocado na saída da transição. Os identificadores são utilizados pelo analisador morfológico para saber que etiquetas atribuir. O custo da transição é usado para estabelecer prioridades entre os transdutores. Quanto maior é o valor do custo, menor é a prioridade do transdutor.

Para cada um dos transdutores que compõem o segmentador, criou-se uma variável de identificação e uma variável de custo. Na tabela 4.2, apresentam-se as variáveis definidas para identificação e os custos dos vários transdutores. As variáveis desta tabela são comuns às duas soluções implementadas. As variáveis específicas de cada solução são apresentadas nas respetivas secções.

Os valores de custo são parametrizáveis. Foram realizados testes empíricos que permitiram determinar as prioridades entre os transdutores e atribuídos valores de custo coerentes com as conclusões obtidas. O transdutor das palavras desconhecidas tem o custo mais alto porque se considera que todos os outros transdutores devem ter prioridade sobre ele. Para resolver a ambiguidade conhecida entre o transdutor dos números romanos e do dicionário, atribui-se um custo mais alto ao transdutor dos números romanos para dar mais prioridade ao dicionário. Os custos dos restantes transdutores foram atribuídos com base nesses valores.

Através da operação da concatenação, as transições com as variáveis de identificação são colocadas no início de todos os transdutores. A figura 4.11 ilustra a adição ao dicionário da respetiva transição para a 1ª solução. A transição é colocada antes do transdutor do dicionário. O processo de adição das transições aos outros transdutores que compõem o segmentador é equivalente.

Transdutor	Variável de identificação	Custo
Verbos	VERBSYMBOL	10
Não-Verbos	WORDSYMBOL	10
Separadores de palavras	SEPARADORDEPALAVRA	2
Endereços de correio eletrónico	EMAILSTART	10
Endereços HTTP	HTTPSTART	10
Endereços IP	IPSTART	9
Referências bíblicas	REFBIBSTART	10
Números romanos	NUMROMSTART	12
Números cardinais	NUMCARDSTART	10
Números enas	NUMENASTART	10
Números ordinais	NUMORDSTART	10
Números fracionários	NUMFRAQSTART	10
Números ordinais/fracionários	NUMORDFRAQSTART	13
Palavras desconhecidas	UNKSTART	15
Suplementar	EXTRASYMBOL	0

Tabela 4.2: Variáveis de identificação comuns às duas soluções e os seus custos

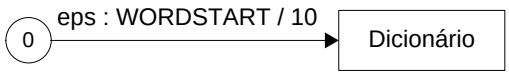

Figura 4.11: Concatenação da variável de identificação ao transdutor do dicionário

Tal como mencionado anteriormente, a adição do identificador e do custo aos separadores de palavras (espaços em branco) é feita de maneira diferente. A figura 4.12 representa a identificação de um espaço em branco durante a segmentação. Os espaços em branco são depois descartados durante a análise morfológica.

Figura 4.12: Identificação de um espaço em branco

Devido à necessidade de separar o que se considera ser separadores, alterou-se a solução original do LexMan. Nessa solução, os separadores faziam parte do dicionário. Neste novo segmentador, os 3 tipos de separadores são gerados em transdutores diferentes.

4.3.3.1 Construção do Segmentador

A geração dos transdutores construídos manualmente começa com a criação de ficheiros de texto que especificam transdutores auxiliares usados na construção do transdutor da respetiva categoria (endereços HTTP, números romanos, etc.). Depois, utilizam-se *scripts* que chamam as operações das bibliotecas *OpenFst* para combinar os transdutores.

Os transdutores do dicionário, dos separadores e o transdutor suplementar são construídos com um programa escrito na linguagem C++. Este programa analisa os ficheiros gerados durante a geração de palavras, constrói as tabelas de símbolos e especifica em ficheiros de texto os transdutores dos não-verbos, dos clíticos, das sub-categorias dos verbos e dos separadores. De seguida, usa-se um *script* para combinar os transdutores dos não-verbos, das sub-categorias dos verbos, dos clíticos e o transdutor suplementar, por forma a obter o transdutor do dicionário. Este processo é igual ao processo de criação do transdutor do dicionário da solução original, que está descrito na secção 4.1.1.2.

Após se ter construído o transdutor do dicionário, um programa escrito em C++ gera o transdutor

das palavras desconhecidas diretamente no formato da biblioteca *OpenFst*, por este não precisar de ser determinizado. O programa também é responsável por concatenar a todos os transdutores previamente construídos, as transições com os respetivos identificadores.

Por fim, o programa utiliza as operações disponibilizadas pela biblioteca *OpenFst* para combinar todos os transdutores construídos e necessários para a construção do segmentador, representado na figura 4.13.

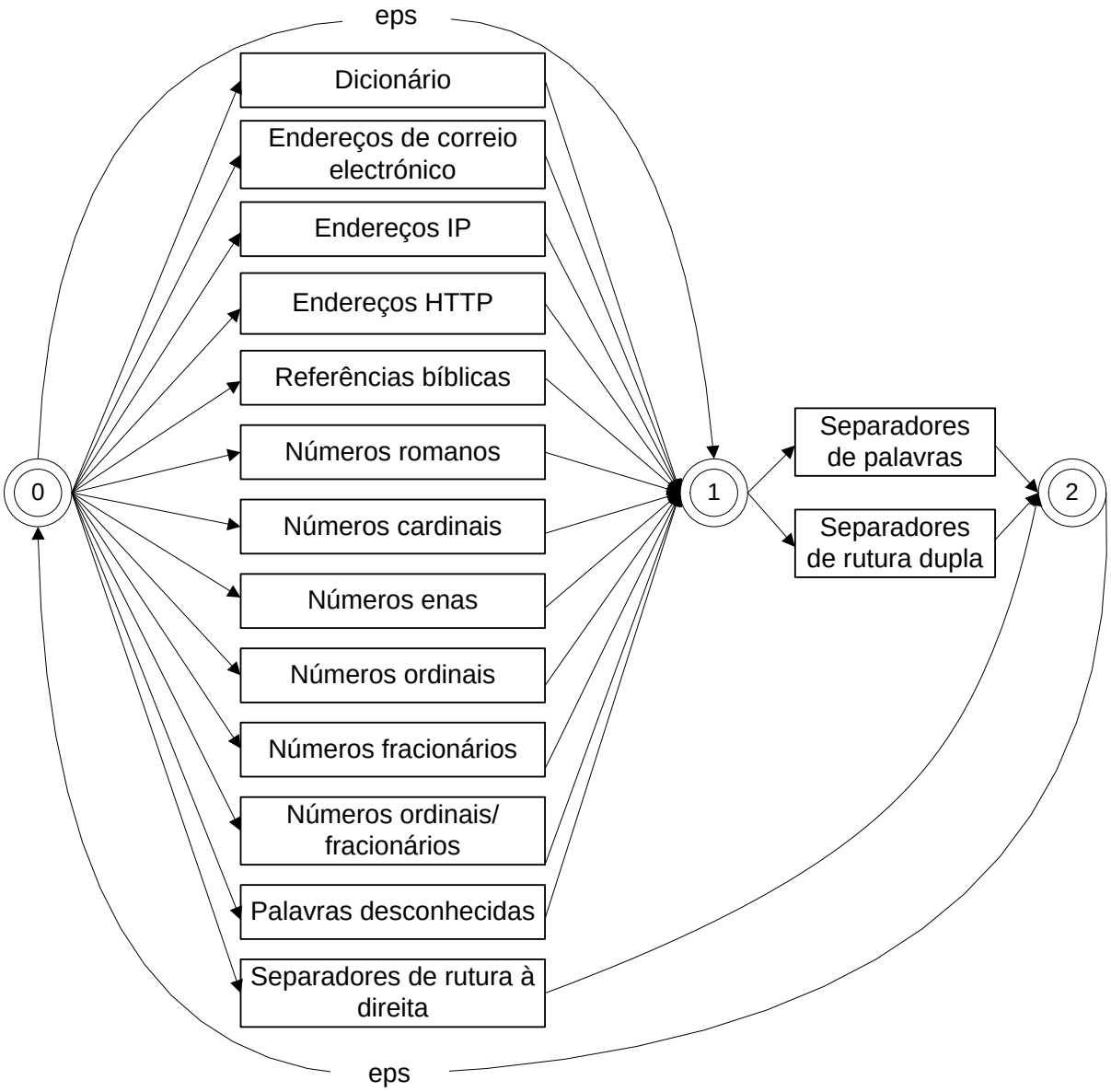

Figura 4.13: Novo segmentador

O transdutor do dicionário, os transdutores construídos manualmente, das palavras desconhecidas e uma transição, que tem o símbolo *epsilon* como símbolo de entrada e símbolo de saída, são unidos e concatenados ao resultado da união dos separadores de palavra e dos separadores de rutura dupla. Ao transdutor resultante, une-se o transdutor dos separadores de rutura à direita. Por fim, efetua-se o fecho do transdutor produzido para implementar um ciclo. O último passo consiste em alterar as propriedades do estado de onde partem os transdutores de separadores de palavras e separadores de rutura dupla para ele passar a ser um estado final.

O segmentador da figura 4.13 consegue segmentar texto que não acaba com um sinal de pontuação de fim de frase. Para além disso, as transições com o símbolo *epsilon* permitem implementar um ciclo.

Assim, por exemplo, é possível aceitar várias palavras ou separadores seguidos, o que permite ao LexMan receber frases na sua entrada.

O dicionário de lemas do LexMan é finito. Ou seja, é possível que o texto contenha segmentos desconhecidos para o LexMan. Esses segmentos desconhecidos são identificados pelo transdutor das palavras desconhecidas e têm o seu identificador, UNKSTART, para que possam ser facilmente identificados e tratados da forma adequada durante a análise morfológica.

O segmentador esquematizado na figura 4.13 é usado nas duas diferentes abordagens que foram implementadas para conseguir segmentar texto e obter as etiquetas morfossintáticas de cada segmento. As diferenças no modo como o segmentador é utilizado nas duas soluções são abordadas nas respetivas secções.

4.3.4 1ª Solução - *ShortestPath*

Foi criada uma solução que permite juntar a segmentação e análise morfológica durante o processamento do LexMan, com base no novo segmentador e na solução original do LexMan para a análise morfológica e usando a funcionalidade da operação *ShortestPath* da biblioteca *OpenFst*. A arquitetura dessa solução está ilustrada na figura 4.14.

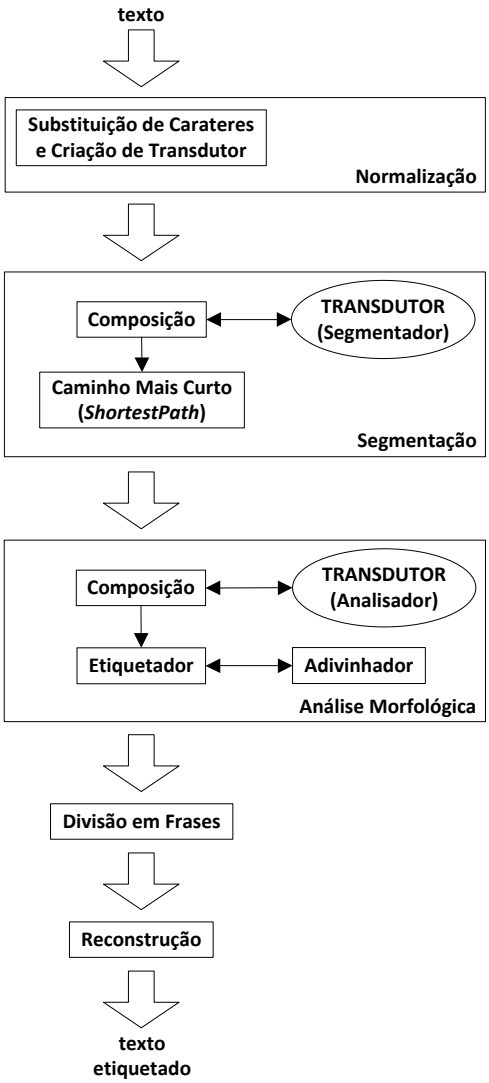

Figura 4.14: Processamento da 1ª solução

Foi criada uma variável de identificação específica para esta solução. Na tabela 4.3, indica-se a que transdutores é que essa variável de identificação fica associada, assim como o seu custo.

Transdutor	Variável de identificação	Custo
Dicionário do segmentador	WORDSTART	10
Separadores de rutura	WORDSTART	10
Analisador	WORDSTART	0

Tabela 4.3: Variável de identificação específica da 1ª solução, com o seu custo

4.3.4.1 Segmentador

A etapa da segmentação desta solução utiliza um transdutor. Esse transdutor é construído com base no esquema do segmentador da figura 4.13. Para formar o transdutor final do segmentador, é necessário criar o transdutor do dicionário. Nesta solução, o segmentador não precisa de incluir a informação morfológica das palavras mas tem de incluir o peso de cada palavra disponível nos ficheiros de dicionário gerados, tal como descrito na secção 4.1.1. Por isso, o transdutor do dicionário da solução original do LexMan foi adaptado para esta solução. A figura 4.15 representa o transdutor equivalente àquele apresentado anteriormente na figura 4.2, adaptado para esta 1ª solução.

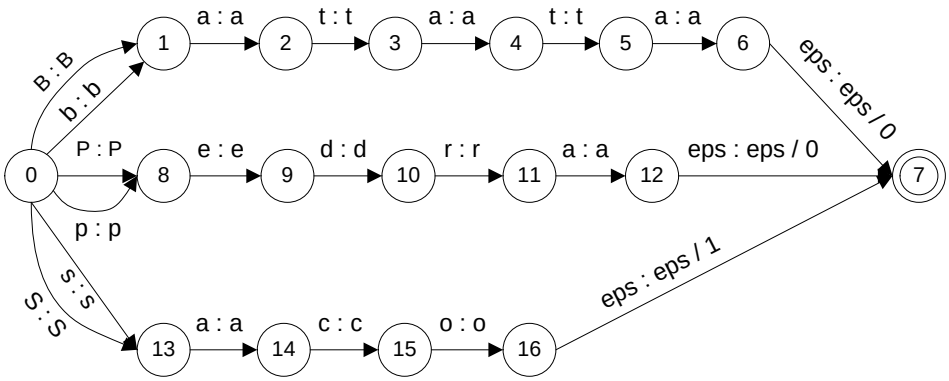

Figura 4.15: Construção de parte do transdutor dos não-verbos, na 1ª solução

Após a transição da última letra da forma de superfície, cria-se uma transição com o símbolo *epsilon* na entrada e na saída. O custo dessa transição é o peso que está associado à respetiva forma de superfície no resultado da geração de palavras. No transdutor representado, as formas *batata* e *pedra* têm um custo igual a zero. A forma *saco* tem um custo igual a um.

Este método de construção é usado em todos os transdutores que permitem formar o transdutor do dicionário. Para esta solução, construiram-se de forma equivalente os transdutores dos separadores de rutura usados na construção do transdutor do segmentador.

O processo de formação do transdutor do dicionário é semelhante àquele descrito na secção 4.1.1. A única diferença é que, antes dos transdutores dos verbos, não-verbos e do transdutor suplementar, adiciona-se uma transição com o identificador WORDSTART. O resultado está representado na figura 4.16. Uma transição com o identificador WORDSTART é também adicionada aos transdutores dos separadores de rutura.

O transdutor do dicionário e os outros transdutores necessários para formar o transdutor do segmentador (ver secção 4.3.3) permitem construir o transdutor que é usado como segmentador desta solução.

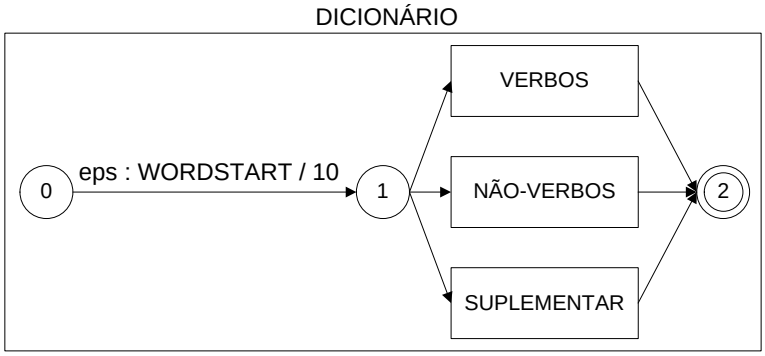

Figura 4.16: Esquema do transdutor do dicionário usado na formação do segmentador da 1ª solução

4.3.4.2 Analisador

Durante a análise morfológica, usa-se um transdutor denominado **analisador**. Este transdutor é semelhante ao transdutor do dicionário que era usado na solução original do LexMan, apresentada na secção 4.1. A construção dos caminhos a partir das palavras geradas, para as categorias dos verbos e não-verbos, respeita o esquema apresentado na figura 4.2. Existe uma transição inicial com um identificador WORDSYMBOL ou VERBSYMBOL, consoante se trata do transdutor das palavras que pertencem à categoria dos não-verbos ou dos verbos. Após as transições construídas para a forma superficial, estão as transições que permitem obter as etiquetas morfossintáticas. Tal como na solução original do LexMan, as sequências de carateres que são consideradas separadores são incluídas no transdutor dos não-verbos. A junção dos transdutores que permitem formar o transdutor do dicionário é feita de acordo com as figuras 4.4 e 4.5. No final, adiciona-se uma transição que tem o identificador WORDSTART como símbolo de entrada e de saída. A figura 4.17 representa o transdutor do analisador que é usado durante a análise morfológica.

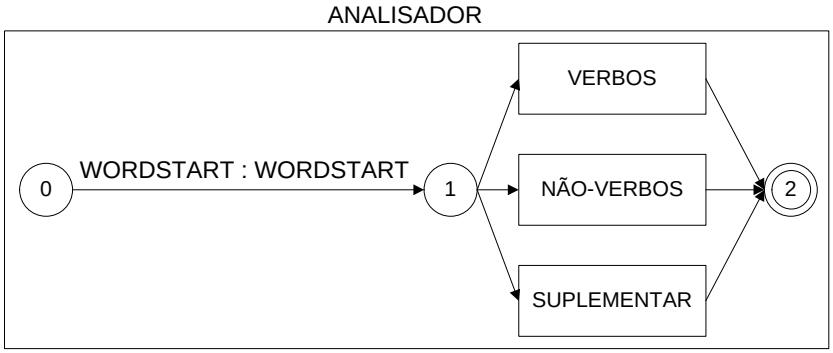

Figura 4.17: Transdutor usado durante a análise morfológica, na 1ª solução

4.3.4.3 Processamento de Texto

O LexMan recebe texto que é transformado num transdutor durante a normalização. Esse transdutor é composto com o segmentador construído para esta solução. O resultado da composição do transdutor do texto com o segmentador é um transdutor cujos caminhos representam as diferentes formas possíveis de segmentar o texto. A figura 4.18 representa o resultado da composição do transdutor criado na normalização da sequência de texto "*vi tudo.*" com o segmentador.

A palavra *vi* é identificada em 3 transdutores. É uma palavra ambígua porque pode ser um número romano ou uma forma verbal do verbo *ver*. Para além disso, essa palavra também é identificada pelo

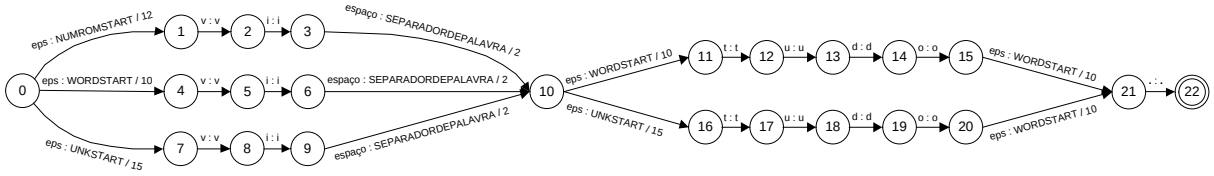

Figura 4.18: Resultado da composição da sequência *vi tudo.* com o segmentador

transdutor de palavras desconhecidas, responsável por identificar quaisquer sequências de carateres que possam formar palavras. Existem três sub-caminhos possíveis no transdutor resultante da composição para a palavra *vi*. Cada sub-caminho do transdutor começa com uma transição que tem o identificador do transdutor onde foi possível encontrar o sub-caminho. A seguir, o espaço funciona como um separador de palavra. Depois, vêm os sub-caminhos da palavra *tudo*. É identificada pelo transdutor das palavras desconhecidas e é uma palavra que faz parte do transdutor do dicionário utilizado na construção do segmentador. O transdutor acaba com uma transição correspondente à identificação do ponto final como separador de rutura dupla.

Ao resultado da composição, aplica-se a operação *ShortestPath* que escolhe o caminho mais curto, ou seja, o caminho com menor custo existente no transdutor resultante da composição. Aplicando a operação *ShortestPath* ao transdutor representado na figura 4.18, obtém-se o transdutor apresentado na figura 4.19

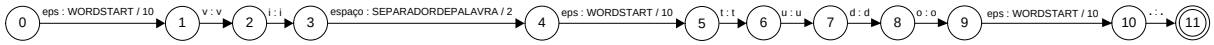

Figura 4.19: Resultado da aplicação da operação *ShortestPath* sobre o transdutor da figura 4.18

O resultado da operação é um transdutor com um único caminho. O caminho escolhido corresponde à melhor segmentação encontrada para o texto. O caminho é composto pelos sub-caminhos com menor custo do transdutor que resulta da composição. A figura 4.19 contém os segmentos encontrados precedidos por transições com os respetivos identificadores. O passo seguinte consiste em analisar esses segmentos e atribuir-lhes etiquetas. O transdutor produzido na segmentação é percorrido para extrair os segmentos. As transições relativas a cada segmento formam um novo transdutor. As transições com o identificador SEPARADORDEPALAVRA são descartadas. Os pesos das transições com os identificadores não são necessários durante a análise morfológica, por isso, não são incluídos nos novos transdutores. São construídos tantos transdutores quanto o número de segmentos encontrados. A figura 4.20 representa os 3 transdutores obtidos após análise do transdutor apresentado na figura 4.19.

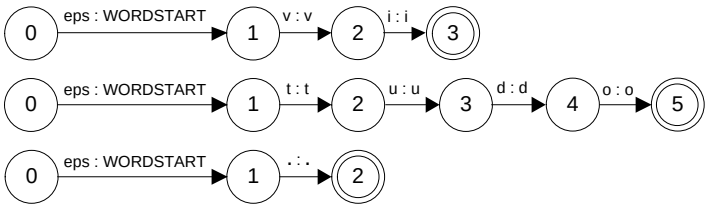

Figura 4.20: Transdutores dos segmentos encontrados no transdutor da figura 4.19

Para cada segmento identificado por um transdutor construído manualmente (números romanos, cardinais, etc.), atribui-se logo a etiqueta correspondente ao segmento. Caso seja um segmento com o identificador WORDSTART, faz-se a composição com o analisador.

Os segmentos com o identificador WORDSTART são compostos com o analisador, representado na figura 4.17. Para cada segmento, obtém-se um transdutor que contém todos os caminhos do dicionário

que emparelharam com o segmento. A figura 4.21 representa o resultado da composição do segmento *vi* com o analisador.

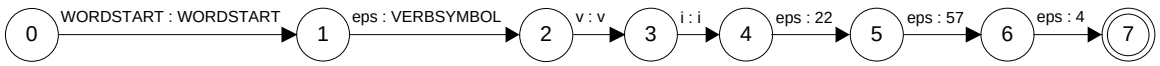

Figura 4.21: Resultado da composição do segmento *vi* com o analisador

A análise do transdutor resultante da composição do segmento com o analisador permite obter as etiquetas morfossintáticas. Todos os caminhos do transdutor são percorridos. Neste caso, só existe um caminho, o da forma verbal do verbo *ver*. Em cada caminho, analisam-se as 3 últimas transições que levam ao estado final. Analisando os símbolos de saída dessas transições, é possível saber em qual dos ficheiros de lemas criados na construção do transdutor do dicionário e em que linha desse ficheiro é que se guardou o lema da forma superficial. A última transição permite indexar a linha do ficheiro onde foram guardadas as etiquetas dos verbos.

Neste exemplo, o resultado da composição só produz um caminho. Quando a palavra é ambígua, o transdutor resultante da composição com o analisador possui vários caminhos. Se a palavra for ao mesmo tempo uma forma verbal e um nome comum, por exemplo, 2 caminhos iriam começar a partir do estado 1. Um dos caminhos teria o identificador VERBSYMBOL e o outro, WORDSYMBOL. É necessário percorrer todos os caminhos do transdutor para procurar o lema e a etiqueta associados a cada um deles. Os lemas e as etiquetas obtidas são todas atribuídas ao segmento analisado.

Caso haja um segmento na saída da segmentação que seja considerado desconhecido (com o identificador UNKSTART), é porque não foi identificado nem pelo transdutor do dicionário, nem pelos transdutores construídos manualmente e nem pelos transdutores relativos aos separadores de rutura. Nestes casos, considera-se que a palavra possa conter maiúsculas no meio da sequência de carateres. Tal como exemplificado na construção do dicionário, considera-se que só a primeira letra da palavra a identificar pode ser maiúscula. Por isso, cria-se um transdutor para este segmento em que todos as letras da palavra, exceto a primeira, são transformadas em minúsculas. Esse transdutor é depois composto com o analisador.

Se o resultado da composição for vazio, considera-se que a palavra é mesmo desconhecida e chama-se o módulo Adivinhador para obter as etiquetas prováveis para o segmento. Caso contrário, a palavra foi identificada pelo analisador após a transformação das letras e é possível percorrer o transdutor resultante para obter as etiquetas morfossintáticas.

A análise morfológica permite obter as etiquetas morfossintáticas de todos os segmentos encontrados. Após a obtenção das etiquetas, é feita uma divisão em frases dos segmentos. Após a divisão em frases, procura-se nos segmentos o caráter especial, usado durante a normalização para substituir carateres desconhecidos. Quando esse caráter especial é encontrado, ele é substituído pelo caráter original do texto, tal como recebido na entrada do LexMan.

Por fim, o texto recebido na entrada do LexMan é colocado na saída do LexMan. O formato da saída apresentada ao utilizador é o mesmo que na solução original do LexMan, descrito na secção 4.1.2. Esta é a saída do LexMan para o exemplo "*vi tudo.*":

```
Sent[0] - word[0]: |vi|          [0,1]     POS->[ver]V.is1s=...=
Sent[0] - word[1]: |tudo|        [3,6]     POS->[tudo]Pi..===.===
Sent[0] - word[2]: |.|           [7,7]     POS->[.]O.........
```

O resultado do processamento do LexMan é colocado numa estrutura XML que é enviada para o módulo seguinte da cadeia STRING, o RuDriCo.

4.3.5 2ª Solução - *Prune*

A elaboração desta solução exigiu a criação de um novo identificador. Este novo identificar é associado aos transdutores dos separadores de rutura e é apresentado na tabela 4.4, juntamente com o seu custo.

Transdutor	Variável de identificação	Custo
Separadores de rutura	RUPTURESYMBOL	10

Tabela 4.4: Variável de identificação específica da 2ª solução

A solução apresentada anteriormente na secção 4.3.4 requer a criação de dois transdutores diferentes. Um para o segmentador, que não contém nenhuma informação morfológica e outro para o analisador morfológico. O texto recebido na entrada do LexMan é composto com o segmentador. Os segmentos resultantes que ficaram marcados com o identificador WORDSTART têm de ser compostos com o transdutor do módulo da análise morfológica.

Com o intuito de melhorar o desempenho do LexMan, foi criada uma segunda solução que requer só um transdutor e uma composição. Esta solução foi criada com base na operação *Prune* da biblioteca *OpenFst*. A arquitetura desta segunda solução está representada na figura 4.22.

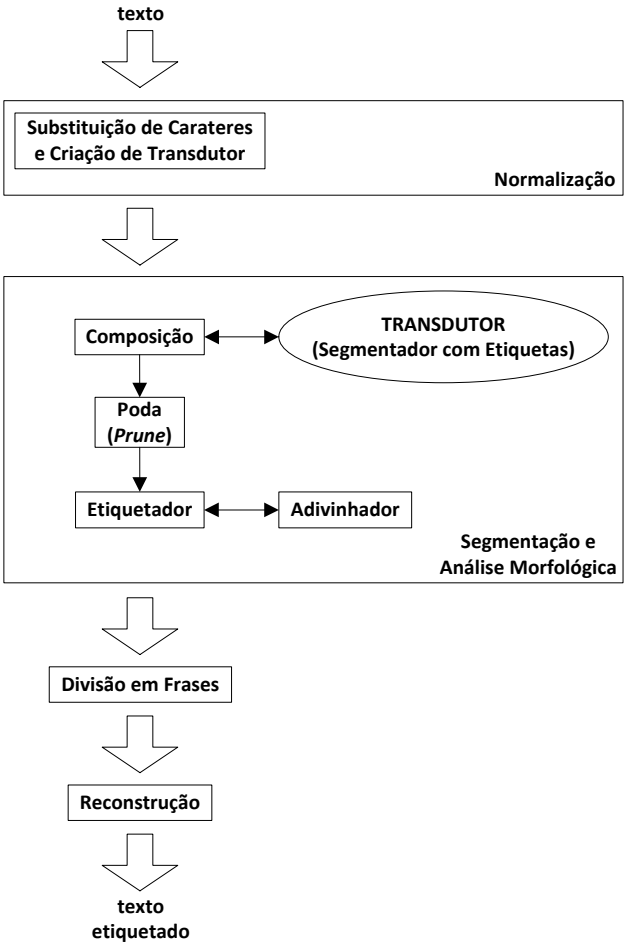

Figura 4.22: Processamento da 2ª solução

4.3.5.1 Segmentador com Etiquetas

Os transdutores do dicionário e dos separadores de rutura que compõem o segmentador são adaptados para esta solução. Os caminhos desses transdutores incluem as três transições especiais a seguir às transições relativas à forma superficial, o que permite efetuar a análise morfológica. Para além disso, os pesos associados às formas superficiais na geração de palavras são também colocados nos caminhos construídos, na transição que leva ao estado final. A inclusão dos pesos é necessária para que possa ser escolhida a melhor segmentação para o texto recebido na entrada do LexMan. A figura 4.23 mostra a construção do transdutor para as palavras que pertencem à categoria dos não-verbos.

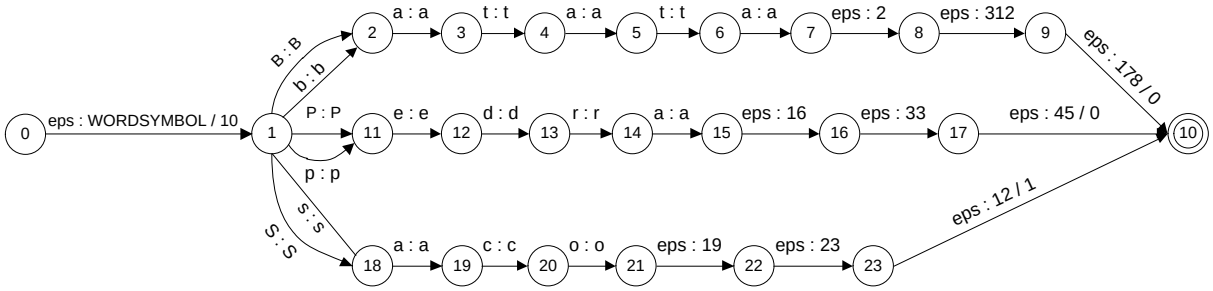

Figura 4.23: Construção de parte do transdutor dos não-verbos, na 2ª solução

Os caminhos construídos para cada palavra começam todos no mesmo estado. Após se terem construído todos os caminhos das palavras que compõem o transdutor dos não-verbos, concatena-se no início do transdutor uma transição onde é colocado o identificador WORDSYMBOL, com o seu respetivo peso. No caso do transdutor dos verbos, a transição com o identificador VERBSYMBOL é adicionada depois de se unirem todos os transdutores correspondentes às formas verbais. Tal como mencionado anteriormente, as 3 transições especiais que permitem obter a informação morfológica são colocadas após as transições que compõem cada forma superficial. Os pesos de cada palavra são colocados na última transição de cada caminho. Os transdutores dos separadores de rutura são construídos de forma equivalente ao transdutor dos não-verbos. O que varia é o identificador que nesta solução, chama-se RUPTURESYMBOL. A junção dos transdutores que compõem o transdutor do dicionário é realizada da mesma forma que na solução original do LexMan, descrita na secção 4.1.1.2.

4.3.5.2 Processamento de Texto

A primeira etapa desta solução é a normalização. O texto recebido na entrada do LexMan é normalizado e transformado num transdutor que é composto com o transdutor do segmentador com etiquetas. O resultado é um transdutor com todos os caminhos do segmentador que emparelharam com o transdutor que foi composto. A figura 4.24 representa o resultado da composição da palavra *amo* com o segmentador com etiquetas.

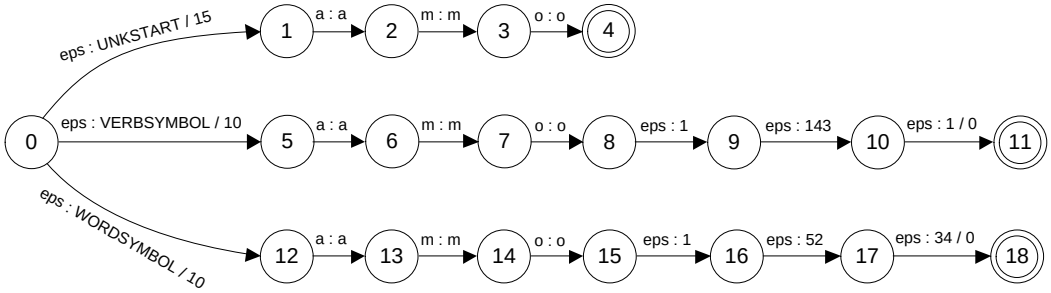

Figura 4.24: Resultado da composição da palavra *amo* com o segmentador com etiquetas

A palavra *amo* é uma palavra ambígua porque pode ser uma forma verbal do verbo *amar* ou um nome comum. Por causa disso, os caminhos dos transdutores dos verbos e não-verbos fazem parte do transdutor resultante da composição. Tal como acontecia na 1ª solução, o transdutor das palavras desconhecidas também emparelha com palavras do transdutor do dicionário que faz parte do segmentador com etiquetas.

Ao resultado da composição, aplica-se a operação *Prune*. Esta operação obtém os caminhos cujo custo é igual ao custo do caminho com menor custo existente no transdutor. Olhando para a figura 4.24, é possível ver que existem 2 caminhos com custo igual a 10 e outro com custo igual a 15. O custo mais baixo é 10 e existem 2 caminhos com esse custo. O transdutor resultante da aplicação da operação *Prune* contém os 2 caminhos que têm custo igual a 10. Este resultado está ilustrado na figura 4.25.

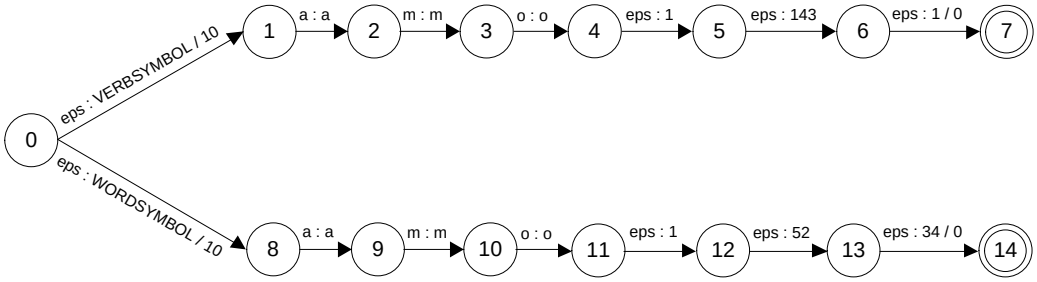

Figura 4.25: Resultado da aplicação da operação *Prune* sobre o transdutor da figura 4.24

Os caminhos selecionados pela operação *Prune* correspondem aos caminhos que definem a melhor segmentação possível para o texto. Para encontrar os segmentos, é necessário percorrer o transdutor resultante. Os segmentos são encontrados através das transições que têm os identificadores. Esses identificadores funcionam como marcadores de início de segmento, exceto o identificador SEPARADORDEPALAVRA que indica a presença de um espaço em branco, que é ignorado na formação de segmentos. A vantagem desta solução em relação à 1ª solução apresentada na secção 4.3.4 é que quando se percorre o transdutor resultante da aplicação da operação *Prune* para encontrar os segmentos, obtém-se ao mesmo tempo a informação morfológica relativa a cada segmento. Caso se trate de um segmento identificado por um dos transdutores construídos manualmente, o lema e a etiqueta são logo atribuídos ao segmento. Caso seja um segmento identificado por WORDSYMBOL, VERBSYMBOL ou RUPTURESYMBOL, analisam-se as 3 transições especiais de modo a obter a informação morfológica.

O processo de obtenção dos segmentos e das suas etiquetas teve de ser modificado para esta solução. Na solução apresentada na secção 4.3.4, os segmentos são divididos em vários transdutores (um para cada segmento). Os segmentos com o identificador WORDSTART são compostos com o analisador, ou seja, o transdutor que é analisado para obter as etiquetas contém os caminhos com etiquetas de uma única palavra. Nesta 2ª solução, o transdutor resultante da normalização é composto com o segmentador com etiquetas. Após a aplicação da operação *Prune*, obtém-se a melhor segmentação para o texto e não se constrói novos transdutores para cada um dos segmentos encontrados. Analisa-se o transdutor resultante da operação *Prune* para obter as etiquetas. A figura 4.26 mostra o transdutor após a aplicação da operação *Prune*, quando o texto recebido na entrada do LexMan é "*vi tudo.*".

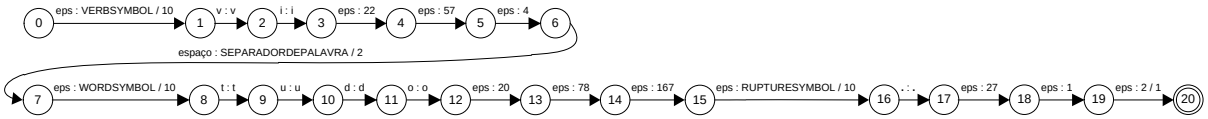

Figura 4.26: Resultado da aplicação da operação *Prune* para o texto "*vi tudo.*"

O transdutor da figura 4.26 é percorrido para encontrar os segmentos através das transições com os identificadores e as respetivas etiquetas. A análise do transdutor teve de ser modificada para permitir

recolher todos os segmentos e todas as etiquetas presentes nos caminhos do transdutor.

Quando uma palavra é considerada desconhecida, aplica-se uma estratégia semelhante àquela apresentada na secção 4.3.4.3. As letras da palavra, exceto a primeira, são transformadas em minúsculas. Constrói-se o transdutor equivalente que é composto com o segmentador com etiquetas.

Após a obtenção dos segmentos e das respetivas etiquetas, realiza-se a divisão em frases dos segmentos encontrados. Após a divisão em frases, é feita uma análise aos segmentos para substituir o caráter especial, usado na normalização para substituir carateres desconhecidos, pelo caráter original do texto recebido na entrada do LexMan.

O resultado do processamento do LexMan é apresentado ao utilizador e enviado ao RuDriCo, nos formatos descritos nas secções 4.1.2 e 4.3.4.3.

4.4 Prefixos

Um objetivo deste trabalho era permitir que a informação usada na geração do transdutor do dicionário fosse complementada com informações sobre palavras derivadas e/ou flexionadas e neologismos com prefixos. Foi tido em conta o facto de que na Língua Portuguesa, os prefixos só são adicionados a adjetivos, advérbios, nomes comuns e verbos. A adição dos prefixos foi integrada na arquitetura construída para a solução descrita na secção 4.3.5.

Um prefixo é adicionado ao início de uma base. Dependendo do prefixo que é adicionado e da primeira letra da base, pode ser necessário efetuar alterações ao lema do prefixo, como por exemplo, acrescentar um hífen no final do prefixo. Por isso, criou-se um módulo responsável pela geração de prefixos. Usando paradigmas de junção, esse módulo aplica as transformações necessárias aos lemas dos prefixos antes de eles serem adicionados a uma base. Este processo está descrito na secção 4.4.1.

4.4.1 Geração de Prefixos

Os prefixos são gerados através de um ficheiro de lema e de paradigma de junção. Cada paradigma de junção tem uma ou mais regras que definem as possíveis transformações a aplicar ao lema do prefixo, consoante a primeira letra da base à qual é adicionado. Nesta secção, define-se a sintaxe usada em cada um desses ficheiros e descreve-se o resultado produzido.

Cada linha do ficheiro dos lemas dos prefixos refere-se a um prefixo. Esta é a sintaxe usada em cada linha:

<prefixo> [<categoria(s)>] paradigma-de-junção

Cada linha é composta por três colunas, separadas por tabulações. Na primeira coluna, entre os símbolos '<' e '>', indica-se qual é o prefixo que está a ser definido. Na segunda coluna, indicam-se entre parênteses retos as categorias de palavras às quais o prefixo pode ser adicionado. Existem quatro valores possíveis: A (adjetivo), ADV (advérbio), N (nome comum) e V (verbo). Para finalizar, na terceira e última coluna, escreve-se o identificador do paradigma que deve ser usado com o respetivo prefixo. De seguida, apresenta-se um exemplo concreto de duas linhas do ficheiro:

<aero> [A,N,V,ADV] Pref14
<de> [A,N,V] Pref3

Os paradigmas de junção são definidos noutro ficheiro. Cada paradigma é definido da seguinte forma:

```
paradigma-de-junção <exemplo>
        regra-1
        regra-2
        ...
        regra-n
$
```

A sintaxe de cada regra é a seguinte:

```
<restrição-base>        CUSTO        <#car1><adiciona-prefixo>Hífen
```

em que:

```
<restrição-base> - caráter inicial (um único) da palavra base; um * significa restantes casos.
CUSTO - custo desta geração (se não for indicado assume-se 0)
<#car1> - número de carateres a retirar ao fim do prefixo
<adiciona-prefixo> - carateres a adicionar ao prefixo depois de aplicar a operação associada a #car1
Hífen - H = hífen obrigatório
        S = sem hífen
        O = hífen opcional
```

Apresenta-se de seguida a definição dos paradigmas para o exemplo apresentado anteriormente:

```
Pref3    <novilatino>
<*>        <0><>S
$

Pref14   <neuro-osteopata>
<o>        <0><>H
<h>        <0><>H
<s>        <0><s>S
<s>        <0><>H
<r>        <0><r>S
<r>        <0><>H
<*>        <0><>O
$
```

O resultado da geração é dividido consoante a categoria das palavras às quais os prefixos são adicionados. Além disso, a regra a aplicar depende da primeira letra da base à qual o prefixo é adicionado. Por isso, para cada categoria, os prefixos gerados são divididos consoante a primeira letra da base à qual são adicionados. No resultado da geração, os prefixos ficam associados ao custo definido na regra que foi aplicada.

As tabelas 4.5 e 4.6 mostram quais são os prefixos gerados após aplicação das regras dos paradigmas aos quais os lemas *aero* e *de* estão associados.

Para cada uma das categorias, existe um ficheiro de texto por para cada restrição-base existente no ficheiro dos paradigmas. No exemplo apresentado, existem 5: *o*, *h*, *s*, *r* e *. Os prefixos gerados a partir da aplicação das regras são distribuídos pelos ficheiros a que pertencem. Por exemplo, os prefixos *aeros*

Restrição-Base	Prefixos Gerados
o	aero-
h	aero-
s	aeros, aero-
r	aeror, aero-
*	aero, aero-

Tabela 4.5: Prefixos gerados a partir do lema *aero* e do paradigma Pref14

Restrição-Base	Prefixos Gerados
*	de

Tabela 4.6: Prefixos gerados a partir do lema *de* e do paradigma Pref3

e *aero-* são colocados no ficheiro de texto que guarda os prefixos a aplicar a bases que começam pela letra *s*. Como o prefixo *aero* pode ser aplicado a bases que pertencem à categoria dos adjetivos, nomes comuns, verbos e advérbios, os prefixos gerados são replicados nos 4 ficheiros respetivos. Estas são as 2 entradas criadas em cada um dos ficheiros:

```
aero-	0
aeros	0
```

O custo indicado na regra que gerou o prefixo é colocado na mesma linha que o prefixo, separado por uma tabulação.

Durante a depuração de problemas na geração de prefixos, verificou-se que alguns prefixos eram gerados em duplicado, com o mesmo custo. Chegou-se à conclusão de que, apesar de não haver nenhum prefixo duplicado no ficheiro dos lemas, as regras aplicadas do paradigma associado faziam com que para a mesma categoria e para a mesma restrição da primeira letra da base, fossem gerados prefixos duplicados. Isso acontece com os prefixos *di-* "dois" e *dis-* "cessação, separação, negação; anomalia; dois, duas vezes" (definições do DLPC da Academia das Ciências (2001)). Segundo os paradigmas definidos, o prefixo *dis* pode ser adicionado a adjetivos e nomes comuns, não existindo restrições sobre a primeira letra da base. Esse prefixo mantém-se igual. Por outro lado, o prefixo *di* também pode ser adicionado a adjetivos e nomes comuns. Se a base começar pela letra *s*, é necessário acrescentar um *s* no final do prefixo, o que produz o prefixo *dis*. Ou seja, para adjetivos e nomes comuns cuja primeira letra é um *s*, o mesmo prefixo gerado seria adicionado duas vezes no mesmo ficheiro, o que é desnecessário.

Por isso, após terem sido gerados todos os prefixos, é feita uma verificação das listas produzidas e os duplicados que surgem na mesma categoria, para a mesma restrição-base, são removidos.

4.4.2 Alterações à Geração de Palavras

Tal como explicado na secção 4.1.1.1, na solução original do LexMan, a geração de palavras dividia os seus resultados em duas categorias: verbos e não-verbos. Devido às restrições impostas pela adição de prefixos, foi necessário alterar o processo de geração de palavras. Os prefixos só são adicionados a adjetivos, advérbios, nomes comuns e verbos, por isso, as palavras geradas a partir do dicionário de lemas e dos paradigmas de flexão são divididas de uma maneira diferente. A categoria dos verbos mantém-se. A categoria dos não-verbos passa a englobar 4 categorias de palavras: **adjetivos**, **advérbios**, **nomes comuns** e uma quarta categoria especial denominada **Outros**, onde se inserem todas as palavras que não são nem adjetivos, nem advérbios, nem nomes comuns nem verbos.

Os prefixos são adicionados às palavras que forem geradas nas categorias dos adjetivos, advérbios, nomes comuns e verbos. Às palavras que forem geradas na categoria Outros, não são adicionados prefixos. Os ficheiros produzidos na geração de palavras são criados respeitando a sintaxe descrita na secção 4.1.1.1.

4.4.3 Construção do Transdutor do Dicionário

A adição dos prefixos é feita no transdutor do dicionário. Os ficheiros de texto que contêm os prefixos gerados são analisados para formar transdutores. Cada prefixo é usado na formação de um caminho. Na última transição do caminho, coloca-se uma transição com o peso que ficou associado ao prefixo no resultado da geração. Para identificar os transdutores dos prefixos, criou-se uma nova variável de identificação. O seu nome e o seu custo são apresentados na tabela 4.7.

Transdutor	Variável de identificação	Custo
Prefixos	PREFIXSTART	1

Tabela 4.7: Variável de identificação específica para os transdutores dos prefixos

Estas são linhas do ficheiro de texto criado para os prefixos que podem ser adicionados a adjetivos que começam pela letra r:

```
anti-    0
antir    0
bi-      0
bir      0
ir       0
```

A figura 4.27 representa o transdutor construído a partir dos prefixos apresentados no exemplo.

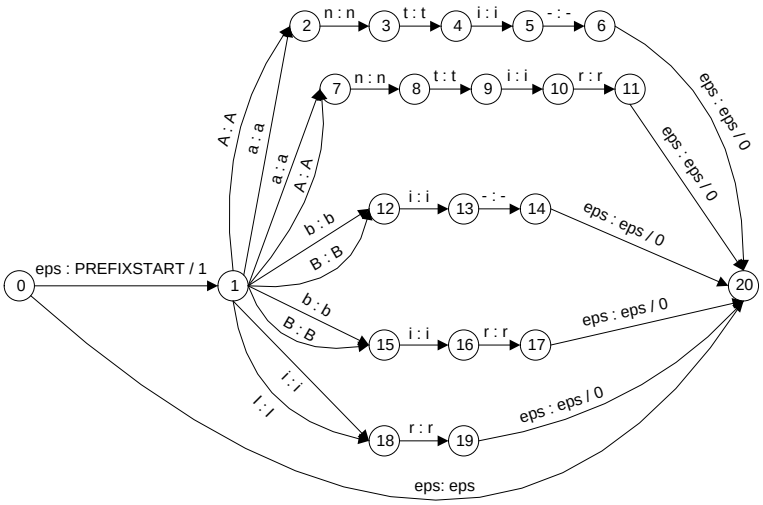

Figura 4.27: Exemplo de caminhos construídos a partir de lista de prefixos gerados

O identificador PREFIXSTART é o símbolo de saída da primeira transição do transdutor. Os caminhos de cada prefixo partem do estado de chegada dessa transição. Considera-se que palavras prefixadas (palavras resultantes da adição de um prefixo a uma base) podem começar por uma letra maiúscula, por isso, são criadas duas transições para a primeira letra do prefixo: uma para a letra minúscula e outra para a letra maiúscula.

Todos os caminhos acabam com uma transição com o símbolo *epsilon* como símbolo de entrada e símbolo de saída. O custo dessa transição é o peso de cada prefixo. Essas últimas transições acabam todas no mesmo estado. Esse estado serve de estado inicial para os caminhos das palavras às quais os prefixos são adicionados. Para permitir que essas palavras continuem a ser identificadas quando não

contêm nenhum prefixo, foi criado uma transição com o símbolo *epsilon* que começa do estado inicial do transdutor e acaba no estado onde começam a ser construídos os caminhos das palavras. Como os prefixos do transdutor representado na figura 4.27 são aqueles a adicionar aos adjetivos que começam pela letra *r*, só se constroem caminhos de palavras que cumprem esse requisito.

O resultado da geração de palavras foi dividido em categorias. Para este exemplo, as palavras que foram geradas na categoria dos adjetivos são analisadas. Neste transdutor, são construídos os caminhos dos adjetivos que começam pela letra *r*. Na figura 4.28, mostra-se um exemplo da construção dos caminhos.

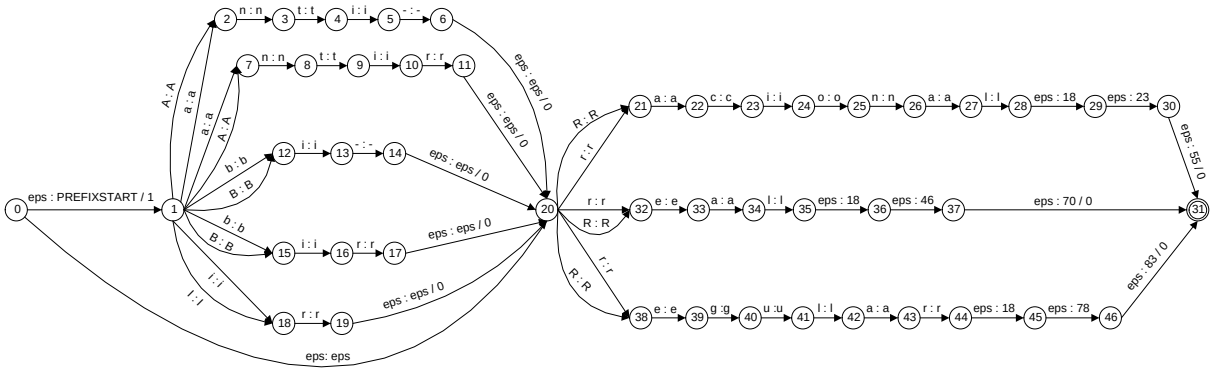

Figura 4.28: Parte do transdutor que identifica adjetivos que começam pela letra *r*

Os caminhos dos prefixos construídos podem ser parcialmente iguais, visto que nalguns casos por exemplo, um prefixo é adicionado com e sem hífen. Esses caminhos parciais são unidos quando é aplicada a operação de determinização sobre o transdutor do dicionário. A figura 4.29 representa o transdutor da figura 4.28 após a determinização.

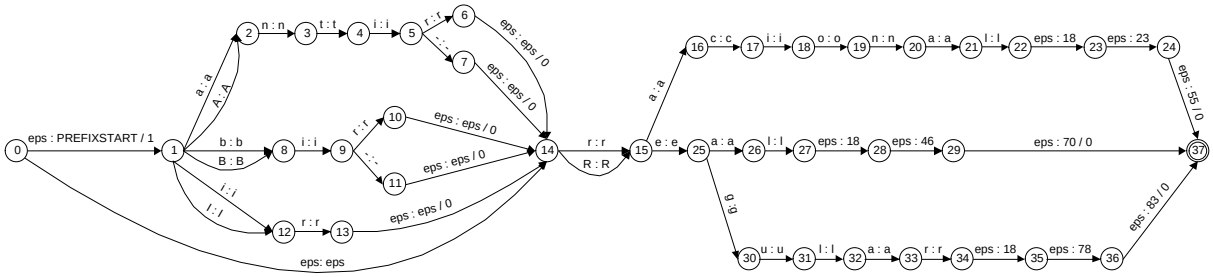

Figura 4.29: Transdutor da figura 4.28 após determinização

4.4.3.1 Junção dos Transdutores

Os transdutores criados de forma equivalente ao transdutor da figura 4.28 são usados para formar os transdutores das respetivas categorias. Usando a biblioteca *FSM*, os transdutores onde foram incluídos os prefixos, consoante a primeira letra das bases, são unidos para criar os transdutores dos adjetivos, advérbios e nomes comuns e verbos. A união dos transdutores que formam os transdutores dos adjetivos, advérbios e nomes comuns está representada na figura 4.30.

Os transdutores representados na figura 4.30, juntamente com o transdutor da categoria Outros, criado para as palavras às quais não são adicionados prefixos, formam o transdutor dos não-verbos. A união desses 4 transdutores é ilustrada na figura 4.31.

A operação de determinização da biblioteca *FSM* move os símbolos de saída associados às transições do transdutor durante a sua execução. Por isso, antes de ser determinizado, os pares entrada-saída são substituídos por um símbolo único através da operação *Encode* da biblioteca *FSM*. Desta forma, os pares entrada-saída são mantidos durante a determinização.

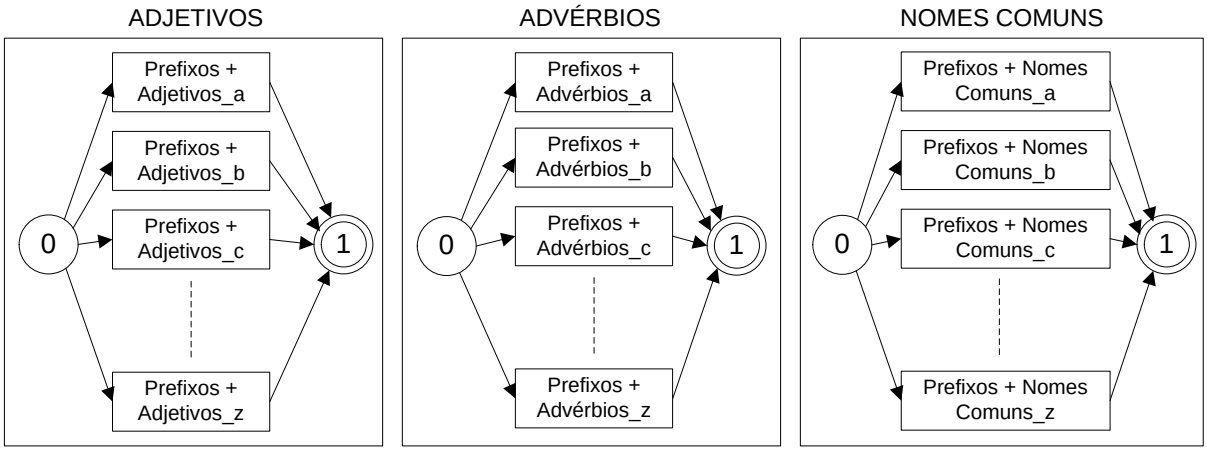

Figura 4.30: União dos transdutores de cada categoria, com prefixos

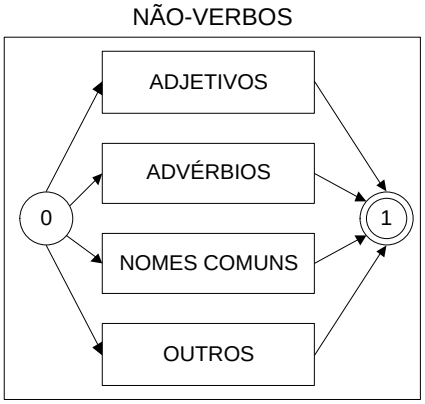

Figura 4.31: União dos transdutores que formam o transdutor dos não-verbos

Após a criação do transdutor dos não-verbos, juntam-se os transdutores que formam o transdutor dos verbos. Os transdutores que formam cada sub-categoria dos verbos são unidos e depois concatenados aos transdutores dos respetivos clíticos. Os transdutores resultantes são unidos para formar o transdutor dos verbos, que é posteriormente determinizado após a aplicação da operação *Encode*. A união dos transdutores que formam o transdutor dos verbos está representada na figura 4.32.

O transdutor dos verbos, assim como o dos não-verbos são convertidos para transdutores da biblioteca *OpenFst*. Depois da conversão, as transições com os identificadores VERBSYMBOL e WORDSYMBOL são concatenadas aos transdutores respetivos. Por fim, os 2 transdutores resultantes da concatenação são unidos ao transdutor suplementar, tal como ilustrado na figura 4.5.

O transdutor do dicionário é usado na construção do segmentador da figura 4.13 e integrado na 2ª solução (ver secção 4.3.5).

4.4.3.2 Processamento de Texto

Relativamente à solução descrita na secção 4.3.5, a análise do transdutor produzido pela operação *Prune* teve de ser alterada para permitir o processamento das palavras prefixadas. A figura 4.33 representa o transdutor criado pela operação *Prune* para a cadeia de texto *Auto-estima*.

A palavra *Auto-estima* é uma palavra prefixada, visto ser o resultado da adição do prefixo *Auto-* à base *estima*.

O transdutor resultante da operação *Prune* é percorrido para se obter as etiquetas. As três últimas transições de cada caminho são analisadas para obter o lema e a etiqueta associados ao caminho. A

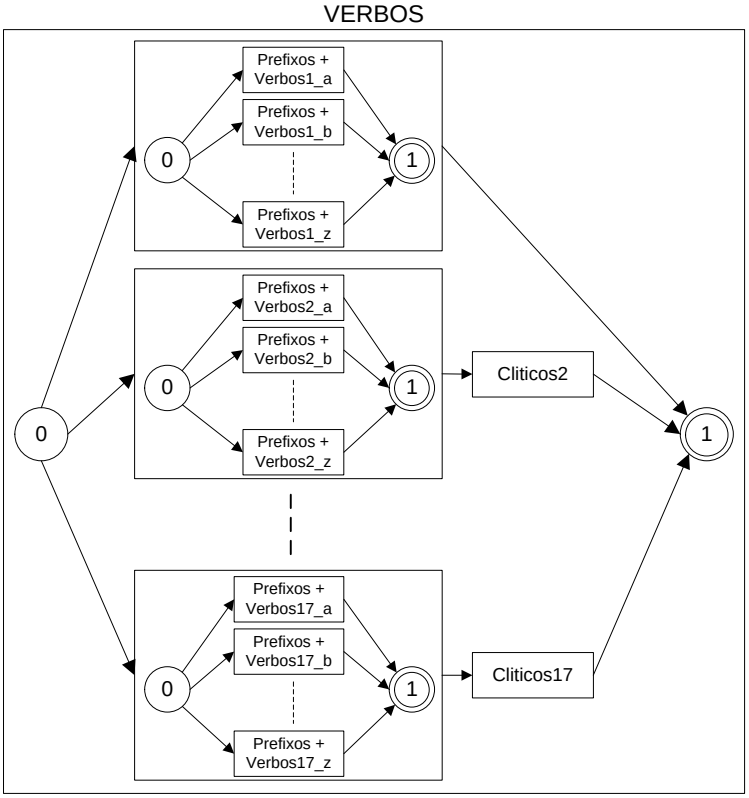

Figura 4.32: União dos transdutores que formam o transdutor dos verbos, com prefixos

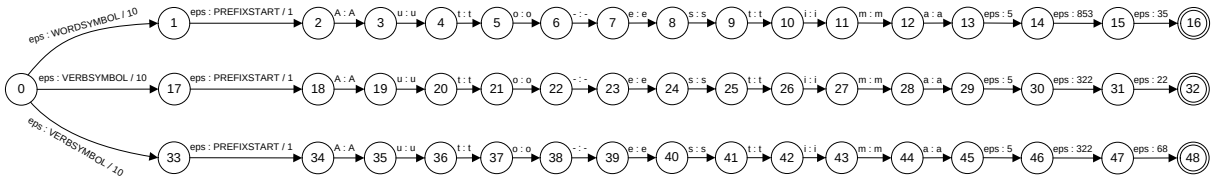

Figura 4.33: Resultado da operação *Prune* para a cadeia de texto *Auto-estima*

existência da transição com o identificador PREFIXSTART num caminho é indicador que a palavra foi identificada como sendo prefixada. Isso significa que a etiqueta obtida tem de ser alterada para refletir essa informação.

Quando se identifica uma palavra como sendo prefixada, o 10º campo da etiqueta é substituído pela letra p quando a palavra é prefixada e para a letra q quando é prefixada e reduzida. Esta é a saída do LexMan para a cadeia de texto *Auto-estima*:

Sent[0] - word[0]: |Auto-estima| [0,10] POS->[estima]Nc...sfn.p= [estimar]V.ip3s=..p= [estimar]V.m=2s=..p=

Como *Auto-estima* é uma palavra prefixada, o 10º campo das 3 etiquetas obtidas foi substituído por um p.

4.4.3.3 Ambiguidade com o Dicionário de Lemas

Antes de existir o módulo que complementa o transdutor do dicionário com prefixos, as palavras prefixadas eram diretamente inseridas no dicionário de lemas. A palavra *reescrever*, por exemplo, era gerada no transdutor do dicionário. No transdutor do dicionário, também se construía um caminho para a palavra

escrever.

Após a adição dos prefixos ao transdutor do dicionário, o prefixo *re* era adicionado à base *escrever*. Ou seja, existem 2 caminhos possíveis no transdutor do dicionário que permitem identificar a palavra *reescrever* após a composição com o segmentador com etiquetas.

No entanto, o caminho construído com o módulo de prefixos tem mais uma transição. Essa transição é a transição com o identificador PREFIXSTART, que tem um custo igual a 1. Ou seja, quando se aplica a operação *Prune* sobre o resultado da composição, o caminho com menor custo é aquele que foi totalmente construído a partir do dicionário de lemas.

O custo da transição com o identificador PREFIXSTART garante que se a palavra prefixada existir no dicionário de lemas, esse caminho do transdutor do dicionário terá sempre prioridade sobre o caminho criado após adição dos prefixos ao transdutor do dicionário.

4.4.4 Restrições às Palavras Prefixadas

Durante o desenvolvimento do módulo que permite adicionar os prefixos ao transdutor do dicionário, chegou-se à conclusão que era necessário implementar várias restrições.

4.4.4.1 Adição dos Prefixos ao Transdutor do Dicionário

Como mencionado anteriormente, as palavras geradas a partir do dicionário de lemas e dos paradigmas de flexão podem ser definidas a partir de expressões regulares. Os caminhos do transdutor do dicionário são gerados analisando o resultado da geração de palavras.

Quando se adiciona um prefixo a uma base, é preciso ter em conta a primeira letra da base. Para simplificar o processo de adição dos prefixos, decidiu-se que não se iriam adicionar prefixos a palavras cuja primeira letra estivesse definida através de uma expressão regular. Para conseguir esse objetivo, os caminhos dessas palavras são todos criados no transdutor da categoria Outros, ao qual não são adicionados prefixos.

Tal como se pode ver na figura 4.28, os caminhos dos prefixos acabam no mesmo estado onde começam a ser criados os caminhos das palavras geradas a partir do dicionário de lemas e dos paradigmas de flexão. Esta solução facilita o processo de adição dos prefixos. Se a primeira letra da base fosse dependente de uma expressão regular, a construção dos caminhos dos prefixos teria de ser incluída no processo de construção dos caminhos das respetivas bases, o que aumentaria a complexidade desse processo.

4.4.4.2 Atribuição de Etiquetas

Os testes ao transdutor do dicionário com prefixos revelaram casos particulares que obrigou à implementação de restrições na atribuição de etiquetas.

Os textos que foram testados tinham siglas que foram identificadas como sendo palavras prefixadas. A sigla *IDC*, por exemplo, era identificada através da adição do prefixo *I* à sigla *DC*, gerada a partir do dicionário de lemas. Criou-se uma restrição para evitar estes casos. Se após a análise do transdutor resultante da operação *Prune* se chegar à conclusão que a palavra foi identificada como sendo uma palavra prefixada, verifica-se se a palavra é composta totalmente por maiúsculas. Caso seja, considera-se que se trata de uma sigla e que não deve ser considerada uma palavra prefixada. As etiquetas que tinham sido obtidas na análise do transdutor são descartadas e chama-se o módulo Adivinhador para atribuir etiquetas ao segmento.

Também foi aplicada outra restrição para as palavras prefixadas cujo prefixo é *de*, *para* ou *sobre*. Quando a primeira letra da base à qual esses prefixos são adicionados é uma maiúscula, a palavra não deve ser considerada prefixada. Um caso detetado nos textos testados é *deComida*, que forma um único segmento por faltar um espaço em branco entre *de* e *Comida*. Após a obtenção das etiquetas, caso se

verifique que a palavra foi identificada como sendo uma palavra prefixada, verifica-se se o prefixo é um dos 3 mencionados acima e se a primeira letra da base é uma maiúscula. Caso isso se verifique, as etiquetas obtidas são descartadas e chama-se o módulo Adivinhador para atribuir etiquetas ao segmento.

A última restrição implementada trata dos casos em que a categoria da palavra prefixada é distinta da categoria da palavra base. Isto acontece para os prefixos *anti*, *inter*, *pós* e *pré*, quando adicionados a um nome comum. Para estes casos, a categoria da palavra prefixada passa a ser um adjetivo. Criou-se a seguinte sintaxe para definir as mudanças de categoria:

<anti>	[Nc]	[A.]
<inter>	[Nc]	[A.]
<pós>	[Nc]	[A.]
<pré>	[Nc]	[A.]

A 1ª coluna indica o prefixo. Quando o prefixo é associado a uma base que tem a categoria indicada na 2ª coluna, a categoria da etiqueta tem de ser substituída por aquela indicada na 3ª coluna. *anti-droga*, *interciclos*, *pós-crise* e *pré-crise* são exemplos de palavras prefixadas em que as etiquetas deixam de ter a categoria de nome comum, para passar a ter a categoria de adjetivo.

4.5 Pesquisa por Lema

A solução original do LexMan era usada fora do contexto da STRING. Com essa solução, era possível fazer uma pesquisa por lema, ou seja, ao colocar um lema na entrada do LexMan, era possível obter todas as palavras que tivessem aquele lema. Isso era feito usando uma versão modificada do transdutor do dicionário usado no contexto da STRING e uma interface própria[3]. A figura 4.34 representa o resultado da pesquisa para o lema *amo*, usando a interface.

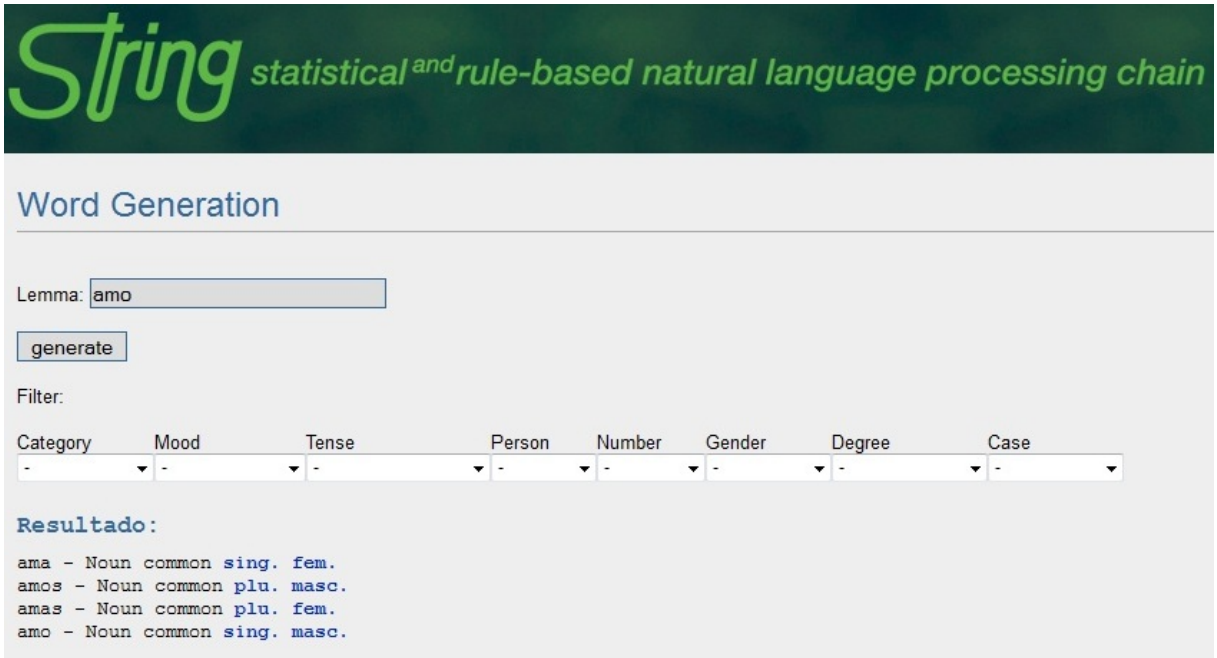

Figura 4.34: Resultado da pesquisa para o lema *amo*

Tal como descrito na secção 4.1, a geração de palavras produzia ficheiros onde estavam indicadas as

[3] `https://string.l2f.inesc-id.pt/demo/wordgenerator.pl` (data de acesso: 13-05-2013)

formas superficiais, os lemas, as etiquetas e os pesos de cada palavra gerada. A construção do transdutor do dicionário consistia na construção das transições da forma superficial, seguidas de 3 transições especiais que permitem indexar o lema e a etiqueta associados. Para esta versão modificada do transdutor do dicionário, o papel da forma superficial e do lema é invertido. As transições iniciais construídas são definidas pelo lema. Depois, as 3 transições especiais permitem indexar a forma superficial e a etiqueta correspondente. Desta forma, o lema recebido na entrada do LexMan emparelha diretamente com o transdutor do dicionário. Durante a análise morfológica, o transdutor é analisado para se obter as formas superficiais e as etiquetas. Desta forma, é possível saber quais são as palavras que têm um dado lema.

Esta funcionalidade do LexMan foi incorporada nos módulos da 2ª solução implementada. O texto recebido na entrada do LexMan é convertido num transdutor que é composto com o transdutor do dicionário. O transdutor resultante é percorrido para obter todas as formas superficiais e etiquetas.

Capítulo 5

Avaliação

A arquitetura original do LexMan foi modificada, tendo sido possível implementar 3 novas arquiteturas para o LexMan. Avaliou-se cada uma das soluções implementadas, o que permite comparar as novas soluções com a solução original do LexMan.

Usando um corpus, as saídas produzidas por cada uma das soluções foram comparadas, tal como descrito na secção 5.1. Foi também feita uma avaliação do desempenho de cada uma das soluções, descrita na secção 5.2. As modificações realizadas no dicionário de lemas após a criação do módulo que permite adicionar os prefixos ao transdutor do dicionário são descritas na secção 5.3.

5.1 Avaliação com Corpus

A avaliação com o corpus permitiu avaliar e corrigir as regras de recomposição que foram transferidas do RuDriCo para o dicionário de lemas do LexMan. Foram criados 4 ambientes de avaliação:

Ambiente A0 - versão da cadeia STRING antes do início deste trabalho.

Ambiente A1 - versão da cadeia STRING após implementação da solução do LexMan baseada na operação *ShortestPath*.

Ambiente A2 - versão da cadeia STRING após implementação da solução do LexMan baseada na operação *Prune*, sem prefixos no transdutor do dicionário.

Ambiente A3 - versão da cadeia STRING após adição dos prefixos ao transdutor do dicionário.

O corpus é enviado para a entrada da cadeia STRING de cada ambiente de avaliação para ser processado. A saída do LexMan e do RuDriCo foram guardadas para avaliação.

As saídas produzidas por esses 2 módulos em cada ambiente foram comparadas. O objetivo era verificar se, apesar das diferenças nas arquiteturas implementadas em cada ambiente, as saídas produzidas por cada um dos módulos após o processamento do corpus estavam corretas.

As diferenças encontradas foram analisadas para verificar se tal se devia a um erro ou se foi devido às melhorias introduzidas no sistema, como por exemplo, a identificação de segmentos compostos ou de palavras que contêm um prefixo. Com estes testes, foi possível definir valores adequados para os pesos dos transdutores que compõem o segmentador. Desse modo, atingiu-se a prioridade pretendida entre eles. As diferenças de segmentação que foram consideradas erros foram, assim, corrigidas.

5.2 Avaliação do Desempenho

Na secção 5.2.1, é apresentada a metodologia da avaliação do desempenho, onde se descreve os ambientes de avaliação e os resultados da avaliação original do LexMan e do RuDriCo.

5.2.1 Metodologia da Avaliação do Desempenho

O LexMan e o RuDriCo são os módulos que foram alterados no desenvolvimento deste trabalho. Para avaliar o desempenho das novas soluções implementadas relativamente à solução original, mediu-se o tempo de CPU (em segundos) necessário para processar diferentes ficheiros de texto. Cada ficheiro de texto contém frases. O objetivo é avaliar o sistema com um número variado de frases. As frases que compõem os ficheiros foram extraídas do CETEMPúblico[1]. Para caraterizar os ficheiros, usa-se o número de frases, o número de palavras do ficheiro e o tamanho do ficheiro em kilobytes (KB). Essas informações, para cada ficheiro, são apresentadas na tabela 5.1.

Ficheiro	Nº de frases	Nº de palavras	Tamanho (KB)
Parte08-1.txt	1	32	0.21
Parte08-10.txt	10	380	2.22
Parte08-100.txt	100	2.388	14.61
Parte08-500.txt	500	11.189	69.51
Parte08-1000.txt	1.000	22.403	140.17
Parte08-5000.txt	5.000	109.902	686.74
Parte08-10000.txt	10.000	218.530	1368.86

Tabela 5.1: Caraterísticas dos ficheiros utilizados na avaliação de desempenho

O número de palavras de cada ficheiro é usado para calcular o tempo médio de CPU gasto por palavra, em milissegundos por palavra (ms/p), por forma a analisar a relação entre o tempo de processamento e o número de palavras do texto processado.

Cada ficheiro é processado 3 vezes. Os tempos apresentados correspondem à média dos 3 tempos obtidos. Para avaliar da melhor maneira possível o desempenho do LexMan e do RuDriCo face à solução original, foram preparados 4 dicionários de lemas:

Dicionário D0 - versão mais recente do dicionário de lemas utilizado na solução original do LexMan, antes do início deste trabalho.

Dicionário D1 - versão do dicionário de lemas após transferência das regras de recomposição para o LexMan e da inclusão das expressões regulares que não foram convertidas manualmente para transdutores.

Dicionário D2 - versão do dicionário de lemas antes de se remover as palavras prefixadas.

Dicionário D3 - versão mais recente do dicionário de lemas após remoção das palavras prefixadas do dicionário.

Para avaliar os dicionários, foram utilizados os mesmos ambientes nos quais o corpus foi avaliado (A0, A1, A2 e A3).

Definiu-se uma estratégia para comparar as soluções implementadas. A figura 5.1 mostra em que ambientes de avaliação é que cada versão do dicionário de lemas é testada.

Para complementar a análise dos tempos obtidos em cada um dos ambientes testados, foram registados os tempos médios de geração, obtidos após 3 tentativas, e o tamanho, em megabytes (MB), dos transdutores necessários para cada solução.

[1]Corpus de Extratos de Textos Eletrónicos MCT/Público

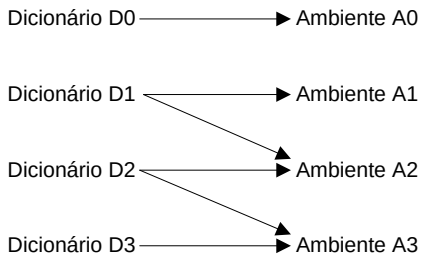

Figura 5.1: Estratégia da avaliação do desempenho

Cada uma das associações apresentadas é testada com os ficheiros de texto apresentados na tabela 5.1. Os testes foram realizados na mesma máquina do L^2F, em alturas em que a máquina não estava a ser usada por outros utilizadores.

O dicionário D0 só é testado no ambiente A0. O dicionário D1 é testado nos ambientes A1 e A2. Os resultados obtidos são comparados entre si e com os resultados do ambiente A0. Desta forma, espera-se poder determinar se as arquiteturas que foram implementadas, baseando-se nas operações *ShortestPath* e *Prune*, têm um melhor desempenho do que a solução original do LexMan. Este teste também serve para determinar qual destas duas novas soluções tem melhor desempenho.

A solução que usa a operação *Prune* é testada com e sem prefixos no transdutor do dicionário, o que permite verificar qual é o impacto no desempenho do sistema, da remoção das palavras prefixadas do dicionário de lemas.

De seguida, apresenta-se a avaliação do sistema original (D0 -> A0). Na tabela 5.2, indica-se o tempo de geração e o tamanho do transdutor, construído a partir do dicionário de lemas D0. Na tabela 5.3, apresentam-se os tempos obtidos após processamento de cada um dos ficheiros.

Construção do Transdutor (s)	Tamanho (MB)
359	181.7

Tabela 5.2: Tempo médio de construção e tamanho do transdutor da solução original

Ficheiro	Segmentador (s)	LexMan (s)	RuDriCo (s)	Soma (s)
Parte08-1.txt	0.22	2.18	1.83	4.23
Parte08-10.txt	0.36	2.34	4.49	7.19
Parte08-100.txt	1.25	3.44	16.57	21.26
Parte08-500.txt	5.10	8.22	65.26	78.58
Parte08-1000.txt	10.01	14.24	135.58	159.83
Parte08-5000.txt	51.00	62.07	683.33	796.40
Parte08-10000.txt	108.49	119.05	1322.82	1550.36

Tabela 5.3: Resultados da avaliação da solução original

Para além dos tempos registados na tabela 5.3, calculou-se o tempo médio de CPU gasto pela solução original por palavra. Esses tempos são calculados para cada um dos módulos avaliados e são apresentados na tabela 5.4.

A análise à tabela 5.4 permite concluir que os resultados obtidos para os textos mais pequenos são os mais afetados pelo tempo necessário ao arranque do programa. O transdutor usado pelo LexMan, por exemplo, tem de ser carregado para a memória do programa. É por isso que existe uma grande diferença entre os resultados obtidos, para os ficheiros mais pequenos. À medida que o número de palavras a processar aumenta, os valores obtidos variam num intervalo de valores mais pequeno, o que permite

Ficheiro	Segmentador (ms/p)	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)
Parte08-1.txt	6.88	68.13	57.19	132.20
Parte08-10.txt	0.95	6.16	11.82	18.93
Parte08-100.txt	0.52	1.44	6.93	8.89
Parte08-500.txt	0.46	0.73	5.83	7.02
Parte08-1000.txt	0.45	0.64	6.05	7.14
Parte08-5000.txt	0.46	0.56	6.22	7.24
Parte08-10000.txt	0.50	0.54	6.05	7.09

Tabela 5.4: Tempo médio de CPU gasto por palavra na solução original

obter uma estimativa mais real do tempo médio de CPU gasto por palavra.

Nas secções seguintes, os resultados apresentados nestas tabelas são posteriormente comparados com os resultados obtidos na avaliação das soluções implementadas no contexto deste trabalho.

5.2.2 1ª Solução - *ShortestPath*

Nesta secção, são apresentados os resultados da avaliação para a 1ª solução implementada, baseada na operação *ShortestPath*. Para esta avaliação, utilizaram-se o dicionário D1 e o ambiente A1.

Para esta solução, são construídos 2 transdutores, o segmentador e o analisador morfológico. O tempo médio de geração dos 2 transdutores e o seus tamanhos são apresentados na tabela 5.5.

Transdutor	Construção (s)	Tamanho (MB)
Segmentador	530	121.2
Analisador	515	210.8

Tabela 5.5: Tempo médio de geração e tamanho dos transdutores da 1ª solução

Como se pode ver nas tabelas 5.2 e 5.5, o tempo médio necessário para construir cada transdutor é maior do que o tempo necessário para gerar o transdutor usado na solução original. Isso já era esperado uma vez que foram transferidas para o dicionário de lemas as regras de recomposição que permitem unir segmentos num só. No transdutor do segmentador, estão também incluídos os transdutores construídos manualmente a partir das expressões regulares que foram transferidas do antigo segmentador.

Os 2 transdutores desta solução foram usados para processar os ficheiros de texto usados na avaliação. Os resultados desta avaliação são apresentados na tabela 5.6. São indicadas as diferenças, em percentagem, dos tempos obtidos nesta solução relativamente à solução original.

Ficheiro	LexMan (s)	RuDriCo (s)	Soma (s)	Diferença para solução original
Parte08-1.txt	3.83	0.08	3.91	-7.6%
Parte08-10.txt	4.06	0.71	4.77	-33.66%
Parte08-100.txt	5.64	3.82	9.46	-55.50%
Parte08-500.txt	12.80	16.41	29.21	-62.83%
Parte08-1000.txt	21.91	33.39	55.30	-65.40%
Parte08-5000.txt	94.07	165.09	259.16	-67.46%
Parte08-10000.txt	183.87	332.02	515.89	-66.72%

Tabela 5.6: Resultados da avaliação da 1ª solução

Comparando os resultados da tabela 5.6 com os resultados da solução original, apresentados na tabela 5.3, a soma dos tempos obtidos em cada módulo, para cada um dos ficheiros, é sempre menor do que

aquela obtida na solução original, para o mesmo ficheiro. Os tempos absolutos obtidos no RuDriCo mostram que a transferência de regras de recomposição para o LexMan melhorou o desempenho do módulo. Havendo menos regras a aplicar, o processamento do RuDriCo ficou mais rápido.

Os tempos do LexMan, que junta a segmentação e a análise morfológica, são mais lentos para os ficheiros que contêm menos frases. A partir do ficheiro que contém 500 frases, esta nova versão do LexMan revela-se mais rápida.

Somando os tempos do LexMan e do RuDriCo, é possível ver que a diferença de tempo relativamente às somas obtidas na solução original vai aumentando à medida que aumenta o número de frases dos ficheiros, estabilizando à volta dos 67% para os ficheiros com maior número de frases.

Calculou-se o tempo médio de CPU gasto por palavra, por esta solução. Os resultados são apresentados na tabela 5.7.

Ficheiro	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)	Diferença para solução original
Parte08-1.txt	119.69	2.5	122.19	-7.57%
Parte08-10.txt	10.68	1.87	12.55	-33.70%
Parte08-100.txt	2.36	1.60	3.96	-55.46%
Parte08-500.txt	1.14	1.47	2.61	-62.82%
Parte08-1000.txt	0.98	1.49	2.47	-65.40%
Parte08-5000.txt	0.85	1.50	2.35	-67.54%
Parte08-10000.txt	0.84	1.52	2.36	-66.71%

Tabela 5.7: Tempo médio de CPU gasto por palavra na 1ª solução

Como se pode ver na tabela 5.7 e comparando com os tempos apresentados na tabela 5.4 da solução original, os tempos obtidos para o RuDriCo baixaram significativamente, de 6.05 para 1.52 ms/p para o maior ficheiro processado, o que representa uma melhoria de aproximadamente 75%. O tempo médio obtido para o LexMan para esse ficheiro, 0.84, representa uma melhoria de 19.23%, relativamente à soma dos tempos obtidos para as soluções originais do segmentador e do LexMan, para o mesmo ficheiro.

Após o desenvolvimento desta solução, a ordem das regras do RuDriCo foi alterada para melhorar o seu desempenho. Para não influenciar os resultados da avaliação deste trabalho, decidiu-se não considerar as otimizações ao RuDriCo e usar os valores da tabela 5.7 em todas as restantes comparações.

5.2.3 2ª Solução - *Prune*

Nesta secção, apresentam-se os resultados das duas avaliações realizadas com o ambiente A2. Os dicionários D1 e D2 são utilizados na construção do transdutor utilizado na solução que junta a segmentação e a análise morfológica numa só etapa, ao usar a operação *Prune*.

Esta solução é avaliada com o dicionário D1 e os resultados são comparados com os resultados da solução original e com os resultados da 1ª solução. Desta forma, é possível verificar se esta 2ª solução é uma melhoria relativamente à solução original e se é melhor do que a 1ª solução implementada. Esta solução é depois avaliada com o dicionário D2. Os resultados desta 2ª avaliação são posteriormente comparados com os resultados obtidos numa solução equivalente mas à qual foram adicionados os prefixos ao transdutor do dicionário.

5.2.3.1 Avaliação do Dicionário D1

Na tabela 5.8, são apresentados os tempo de construção do transdutor para esta solução, assim como o seu tamanho.

Construção do Transdutor (s)	Tamanho (MB)
638	213.2

Tabela 5.8: Tempo médio de construção e tamanho do transdutor da 2ª solução para o dicionário D1

Esta avaliação é feita com o dicionário D1, que foi usado para avaliar a solução baseada na operação *ShortestPath*. Esta avaliação permite comparar essa abordagem com esta 2ª solução, baseada na operação *Prune*. Os resultados são também comparados com os resultados da solução original.

Uma vez que esta solução junta a segmentação e a análise morfológica numa etapa, é de esperar que se obtenham resultados melhores para esta solução, comparativamente com aqueles obtidos na avaliação da 1ª solução.

Ficheiro	LexMan (s)	RuDriCo (s)	Soma (s)	Diferença para solução original	Diferença para 1ª solução
Parte08-1.txt	4.81	0.08	4.89	+15.60%	+25.06%
Parte08-10.txt	4.97	0.71	5.68	-21.00%	+19.08%
Parte08-100.txt	5.89	3.82	9.71	-54.33%	+2.64%
Parte08-500.txt	10.97	16.41	27.38	-65.16%	-6.26%
Parte08-1000.txt	16.84	33.39	50.23	-68.57%	-9.17%
Parte08-5000.txt	70.24	165.09	235.33	-70.45%	-9.20%
Parte08-10000.txt	139.33	332.02	471.35	-69.60%	-8.63%

Tabela 5.9: Resultados da avaliação da 2ª solução com o dicionário D1

Como se pode ver pelos tempos apresentados na tabela 5.9, esta solução tem um desempenho pior do que a 1ª solução no processamento dos ficheiros com menor número de frases. A diferença entre os tempos vai diminuindo até se tornar favorável para esta solução a partir das 500 frases. À medida que o número de frases no ficheiro que é testado aumenta, a diferença entre os tempos é cada mais favorável a esta solução, estabilizando para uma vantagem de aproximadamente 9% para os ficheiros com maior número de frases.

Esta vantagem relativamente à 1ª solução nos ficheiros com maior número de frases é traduzida nas percentagens de diferença obtidas relativamente à solução original. Com esta solução, obtém-se um máximo de 70.45% de vantagem face à solução original. Com a 1ª solução, o valor máximo obtido foi 67.46%.

Na tabela 5.10, apresentam-se os tempos médios de CPU gasto por palavra, para esta solução.

Ficheiro	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)	Diferença para solução original	Diferença para 1ª solução
Parte08-1.txt	150.31	2.5	152.81	+15.59%	+25.06%
Parte08-10.txt	13.08	1.87	14.95	-21.02%	+19.12%
Parte08-100.txt	2.47	1.60	4.07	-54.22%	+2.78%
Parte08-500.txt	0.98	1.47	2.45	-65.10%	-6.13%
Parte08-1000.txt	0.75	1.49	2.24	-68.63%	-9.31%
Parte08-5000.txt	0.64	1.50	2.14	-70.44%	-8.94%
Parte08-10000.txt	0.64	1.52	2.16	-69.53%	-8.47%

Tabela 5.10: Tempo médio de CPU gasto por palavra na 2ª solução com o dicionário D1

Como se usaram os mesmos valores de tempo para o RuDriCo, os valores apresentados na tabela 5.10 para esse módulo mantêm-se. A partir do ficheiro com 500 frases, os valores médios obtidos para o

LexMan são menores do que aqueles obtidos na avaliação da 1ª solução. Para o ficheiro com 10.000 frases, e comparando com o resultado obtido na 1ª solução, houve uma melhoria de 23.8%.

Em suma, pode-se concluir que a abordagem baseada na operação *Prune* é melhor do que a solução que se baseia na operação *ShortestPath* e do que a solução original do sistema para a segmentação e a análise morfológica. Para esta abordagem, é necessário a construção de um transdutor equivalente ao segmentador da solução, o que é uma vantagem relativamente à abordagem baseada na operação *ShortestPath* que necessita de 2 transdutores. Para além disso, os resultados obtidos demonstram que, no processamento de um corpus de grandes dimensões, a abordagem baseada na operação *Prune* tem um desempenho mais eficiente.

5.2.3.2 Avaliação do Dicionário D2

Optou-se por avaliar o dicionário D2 para fazer a comparação com os resultados obtidos na avaliação da solução em que os prefixos foram adicionados ao transdutor do dicionário.

Na tabela 5.11, encontram-se o tempo de construção e o tamanho do transdutor criado para esta avaliação.

Construção do Transdutor (s)	Tamanho (MB)
653	215.5

Tabela 5.11: Tempo médio de construção e tamanho do transdutor da 2ª solução para o dicionário D2

O transdutor construído a partir do dicionário D2 é maior que aquele criado para avaliar o dicionário D1. O dicionário de lemas do LexMan vai sendo modificado ao longo do tempo e o número de palavras no dicionário tem tendência a aumentar. Como o dicionário D2 é mais recente do que o dicionário D1, ele contém mais palavras, daí que o transdutor construído seja maior.

A avaliação do dicionário D2 com a 2ª solução é apresentada na tabela 5.12.

Ficheiro	LexMan (s)	RuDriCo (s)	Soma (s)	Diferença para solução original
Parte08-1.txt	4.84	0.08	4.92	+16.31%
Parte08-10.txt	5.02	0.71	5.73	-20.31%
Parte08-100.txt	6.08	3.82	9.90	-53.43%
Parte08-500.txt	11.35	16.41	27.76	-64.67%
Parte08-1000.txt	17.27	33.39	50.66	-68.30%
Parte08-5000.txt	71.63	165.09	236.72	-70.28%
Parte08-10000.txt	141.12	332.02	473.14	-69.48%

Tabela 5.12: Resultados da avaliação da 2ª solução com o dicionário D2

Os tempos obtidos com este dicionário D2 são semelhantes àqueles obtidos com o dicionário D1 mas há uma pequena diminuição de desempenho com este dicionário. Isso pode ser explicado com o tamanho do transdutor usado na composição com o texto. Como o transdutor construído para o dicionário D2 é maior, a composição do texto com o transdutor do segmentador é menos eficiente, o que explica a diminuição de desempenho registada nesta avaliação, face aos resultados obtidos com o dicionário D1.

Essa diminuição de desempenho também é visível nos tempos médios de CPU gastos por palavra, com esta solução. Os resultados são apresentados na tabela 5.13.

Para todos os ficheiros, as médias obtidas para o LexMan são maiores que aquelas apresentadas na tabela 5.10. A diferença entre os valores obtidos nas duas avaliações vai diminuindo. Para os ficheiros

Ficheiro	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)	Diferença para solução original
Parte08-1.txt	151.25	2.5	153.75	+16.30%
Parte08-10.txt	13.21	1.87	15.08	-20.34%
Parte08-100.txt	2.55	1.60	4.15	-53.32%
Parte08-500.txt	1.01	1.47	2.48	-64.67%
Parte08-1000.txt	0.77	1.49	2.26	-68.35%
Parte08-5000.txt	0.65	1.50	2.15	-70.30%
Parte08-10000.txt	0.65	1.52	2.17	-69.39%

Tabela 5.13: Tempo médio de CPU gasto por palavra na 2ª solução com o dicionário D2

com 5.000 e 10.000 frases, a diferença é somente de 1 ms/p.

Os dados apresentados nas tabelas desta avaliação são usados na comparação com a 3ª solução, que é a abordagem baseada na operação *Prune*, em que foram adicionados os prefixos ao transdutor do dicionário.

5.2.4 3ª Solução - *Prune* com Prefixos

A 3ª solução é avaliada com o dicionário D2. Ao comparar os resultados desta avaliação com os resultados obtidos na avaliação da abordagem equivalente mas sem os prefixos, consegue-se determinar que impacto tem a adição dos prefixos no transdutor do dicionário desta solução.

Palavras que passaram a ser identificadas com a adição dos prefixos e que existiam no dicionário de lemas foram removidas do dicionário de lemas, formando o dicionário D3. Esse dicionário é a versão mais recente do dicionário de lemas até hoje. O dicionário D3 é avaliado com esta solução e os resultados obtidos são comparados com aqueles obtidos com o dicionário D2.

5.2.4.1 Avaliação do Dicionário D2

Na tabela 5.14, são indicados o tempo necessário para formar o transdutor do segmentador e o seu tamanho.

Construção do Transdutor (s)	Tamanho (MB)
716	239.1

Tabela 5.14: Tempo médio de construção e tamanho do transdutor da 3ª solução para o dicionário D2

Uma vez que o transdutor do dicionário tem mais transições após a adição dos prefixos, a construção do transdutor do segmentador demora mais tempo. O aumento do número de transições também significa que o transdutor construído é maior do que todos os transdutores avaliados anteriormente.

Os resultados da avaliação desse transdutor com o dicionário D2 são apresentados na tabela 5.15.

Tal como se esperava, a adição dos prefixos ao transdutor do dicionário fez com que os ficheiros demorassem mais tempo a serem processados. O aumento do número de transições no transdutor do dicionário torna a composição do transdutor do texto com o transdutor do segmentador mais lenta. À medida que o número de frases aumenta, o desempenho desta solução piora relativamente à solução que não tem os prefixos no transdutor do dicionário. Mesmo depois de complementar a arquitetura do LexMan com um módulo que permite identificar palavras prefixadas, o processamento conjunto do LexMan e do RuDriCo é mais rápido que o processamento da solução original para esses 2 módulos.

Na tabela 5.16, apresentam-se os tempos médios de CPU gastos por palavra, por esta solução.

Ficheiro	LexMan (s)	RuDriCo (s)	Soma (s)	Diferença para solução original	Diferença para 2ª solução com D2
Parte08-1.txt	5.36	0.08	5.44	+28.61%	+10.57%
Parte08-10.txt	5.63	0.71	6.34	-11.82%	+10.65%
Parte08-100.txt	7.22	3.82	11.04	-48.07%	+11.52%
Parte08-500.txt	14.59	16.41	31.00	-60.55%	+11.67%
Parte08-1000.txt	24.64	33.39	58.03	-63.69%	+14.55%
Parte08-5000.txt	113.3	165.09	278.39	-65.04%	+17.60%
Parte08-10000.txt	214.75	332.02	546.77	-64.73%	+15.56%

Tabela 5.15: Resultados da avaliação da 3ª solução com o dicionário D2

Ficheiro	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)	Diferença para solução original	Diferença para 2ª solução com D2
Parte08-1.txt	167.50	2.5	170.00	+28.59%	+10.57%
Parte08-10.txt	14.82	1.87	16.69	-11.83%	+10.68%
Parte08-100.txt	3.02	1.60	4.62	-48.03%	+11.33%
Parte08-500.txt	1.30	1.47	2.77	-60.54%	+11.69%
Parte08-1000.txt	1.10	1.49	2.59	-63.73%	+14.60%
Parte08-5000.txt	1.03	1.50	2.53	-65.06%	+17.67%
Parte08-10000.txt	0.98	1.52	2.50	-64.74%	+15.21%

Tabela 5.16: Tempo médio de CPU gasto por palavra na 3ª solução com o dicionário D2

Os tempos obtidos para o LexMan pioraram relativamente à solução sem os prefixos, o que é um reflexo dos dados apresentados na tabela 5.15. Para o ficheiro com 10.000 frases, houve uma baixa de desempenho de 50%. Apesar disso, a média obtida para esse ficheiro é mais baixa que aquela obtida na solução original, que é de 1.04 (somando os valores obtidos para a segmentação e para a análise morfológica), o que representa uma melhoria de 5.77%.

5.2.4.2 Avaliação do Dicionário D3

A remoção de palavras prefixadas do dicionário de lemas permitiu formar o dicionário D3. Tal como se pode ver na tabela 5.17, ao reduzir o número de palavras do dicionário, a construção do transdutor ficou mais rápida e o transdutor construído ficou mais pequeno.

Construção do Transdutor (s)	Tamanho (MB)
652	215.8

Tabela 5.17: Tempo médio de construção e tamanho do transdutor da 3ª solução para o dicionário D3

Os resultados desta avaliação são apresentados na tabela 5.18, onde se pode ver que a diminuição do número de palavras do transdutor do dicionário permitiu um processamento mais rápido dos ficheiros.

Os tempos médios de CPU gastos por palavra são apresentados na tabela 5.19.

A remoção de palavras do dicionário permitiu obter com o LexMan valores médios de CPU gastos por palavra mais baixos do que na avaliação com o dicionário D2. Ao avaliar o ficheiro com 10.000 frases, houve uma melhoria de 7.69% no resultado do LexMan relativamente à solução original.

Ficheiro	LexMan (s)	RuDriCo (s)	Soma (s)	Diferença para solução original	Diferença para 3ª solução com D2
Parte08-1.txt	4.84	0.08	4.92	+16.31%	-9.56%
Parte08-10.txt	5.08	0.71	5.79	-19.47%	-8.68%
Parte08-100.txt	6.68	3.82	10.50	-50.61%	-4.89%
Parte08-500.txt	14.07	16.41	30.48	-61.21%	-1.68%
Parte08-1000.txt	24.14	33.39	57.53	-64.01%	-0.86%
Parte08-5000.txt	103.59	165.09	268.68	-66.26%	-3.49%
Parte08-10000.txt	208.92	332.02	540.94	-65.11%	-1.07%

Tabela 5.18: Resultados da avaliação da 3ª solução com o dicionário D3

Ficheiro	LexMan (ms/p)	RuDriCo (ms/p)	Soma (ms/p)	Diferença para solução original	Diferença para 3ª solução com D2
Parte08-1.txt	151.25	2.5	153.75	+16.30%	-9.56%
Parte08-10.txt	13.37	1.87	15.24	-19.49%	-8.69%
Parte08-100.txt	2.80	1.60	4.40	-50.51%	-4.76%
Parte08-500.txt	1.26	1.47	2.73	-61.11%	-1.44%
Parte08-1000.txt	1.08	1.49	2.57	-64.01%	-0.77%
Parte08-5000.txt	0.94	1.50	2.44	-66.30%	-3.56%
Parte08-10000.txt	0.96	1.52	2.48	-65.02%	-0.80%

Tabela 5.19: Tempo médio de CPU gasto por palavra na 3ª solução com o dicionário D3

5.2.5 Evolução dos Resultados

Nesta secção, é feita uma análise da evolução dos resultados obtidos nas avaliações realizadas.

As barras do gráfico representado na figura 5.2 são calculadas usando a soma, numa escala logarítmica de base 2, dos tempos de processamento dos módulos avaliados em cada solução (Segmentador (na solução original), LexMan e RuDriCo), para cada ficheiro.

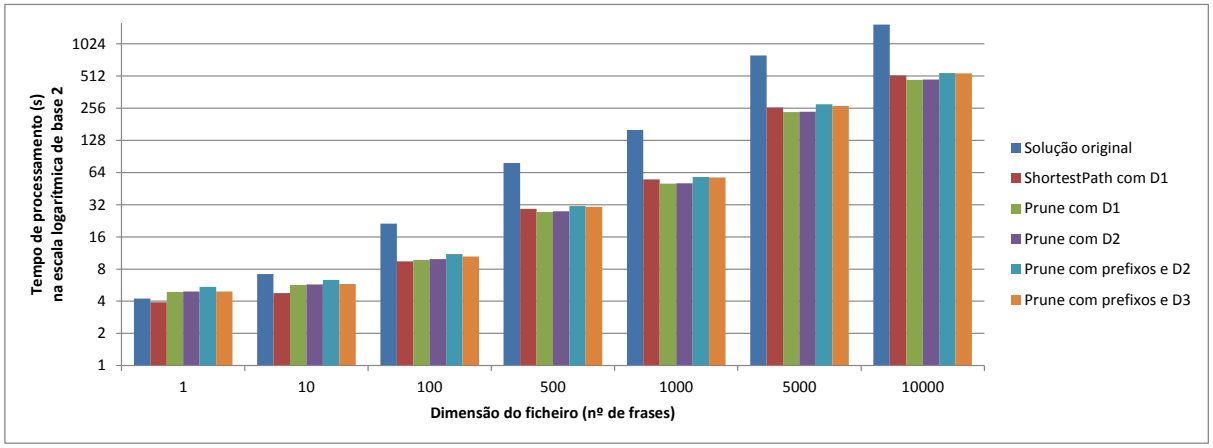

Figura 5.2: Soma dos tempos do Segmentador (na solução original), do LexMan e RuDriCo

O tempo de processamento é proporcional ao tamanho do texto. Quanto maior for o texto, mais tempo demora o seu processamento. Isso é visível nos resultados de todas as soluções. A análise do gráfico também permite concluir que, para os ficheiros com menor número de frases (1 e 10), havia pouca diferença entre os resultados obtidos em cada solução. A partir do ficheiro com 10 frases e à medida

que o número de frases aumenta, as soluções construídas neste trabalho revelaram-se sempre melhores do que a solução original. Para processar um corpus com dimensões semelhantes a esses ficheiros, todas as soluções implementadas são mais eficientes do que a solução original. A solução baseada na operação *Prune* é a que se revela mais eficiente nessa tarefa.

Na figura 5.3, apresenta-se a evolução da soma dos tempos médios de CPU gastos por palavra, por parte do Segmentador (só na solução original), LexMan e do RuDriCo, para cada ficheiro avaliado, em cada solução. Para facilitar a leitura dos resultados, o tempo de processamento é apresentado numa escala logarítmica de base 2.

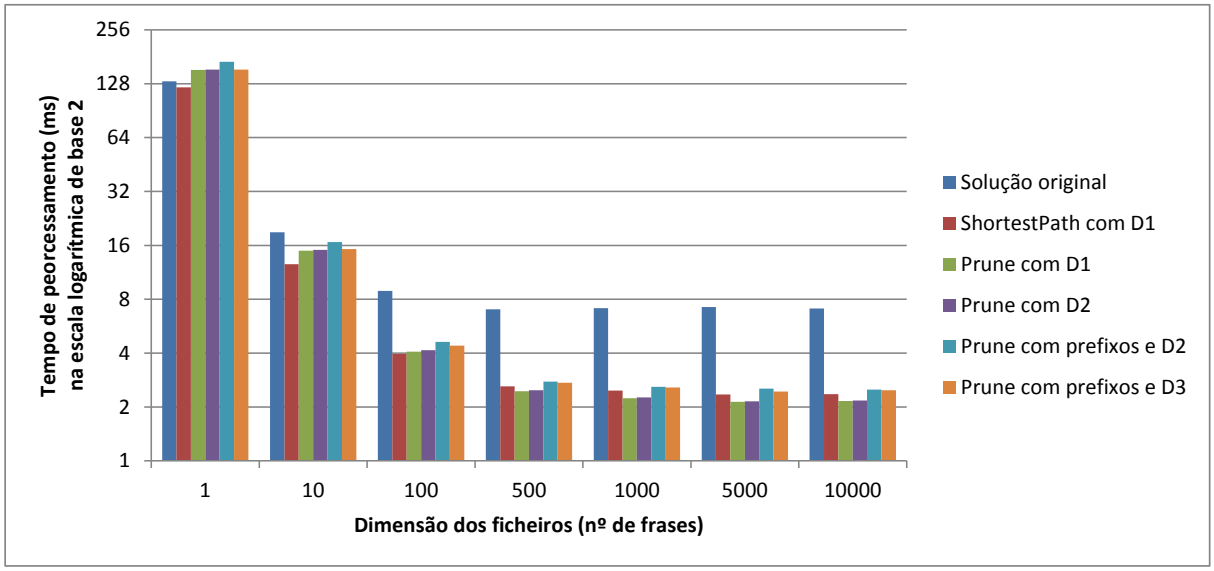

Figura 5.3: Soma dos tempos médios de CPU gastos por palavra, pelo LexMan e RuDriCo

À medida que o número de palavras a analisar aumenta, a diferença entre os valores obtidos em cada ficheiro vai diminuindo até variar em intervalos muito pequenos para os ficheiros com maior número de palavras. Os resultados obtidos para as soluções construídas neste trabalho são piores que os resultados da solução original para o ficheiro mais pequeno (exceto para a solução baseada na operação *ShortestPath*). A tendência inverte-se a partir do ficheiro com 10 frases. Uma vez que os tempos do RuDriCo foram normalizados para as novas soluções, a variação de valores é influenciada pelos resultados do LexMan nessas soluções. A análise do gráfico permite concluir que a solução baseada na operação *Prune* é aquela que permite obter as melhores médias para ficheiros de maior dimensão. O impacto da adição dos prefixos ao transdutor do dicionário é também visível no gráfico.

Nos resultados apresentados nas tabelas das diversas avaliações, nota-se que por vezes, a evolução dos tempos obtidos entre ficheiros aproximava-se de uma evolução linear.

5.2.6 Avaliação da Memória

Nesta secção, é feita uma comparação da quantidade de memória que é necessário que esteja disponível para processar os ficheiros de avaliação com a STRING. A comparação é feita entre a solução original com o dicionário D0 e a solução que se baseia na operação *Prune* após adição dos prefixos ao transdutor do dicionário. Esta solução é avaliada com o dicionário mais recente, D3. A unidade utilizada é o megabyte (MB). Os resultados desta avaliação são apresentados na tabela 5.20.

Foi durante a operação da composição que os valores registados na tabela 5.20 foram atingidos. A quantidade de memória necessária aumenta à medida que o tamanho do texto a processar aumenta.

Na solução original, os segmentos por etiquetar são compostos com o transdutor utilizado por essa

Ficheiro	Memória para a solução original (MB)	Memória para a solução mais recente (MB)
Parte08-1.txt	533	1.006
Parte08-10.txt	582	1.337
Parte08-100.txt	652	1.479
Parte08-500.txt	677	2.148
Parte08-1000.txt	738	2.694
Parte08-5000.txt	869	6.677
Parte08-10000.txt	991	11.668

Tabela 5.20: Memória necessária para processar os ficheiros com a solução original e a mais recente

solução. Na solução mais recente, o texto processado é transformado num transdutor que é composto com o transdutor do segmentador. Nessa solução, quanto maior for o texto a processar, maior é o transdutor resultante da composição, o que provoca um aumento de memória ocupada pelo processo.

5.3 Avaliação dos Prefixos

Nesta secção, são descritas as alterações efetuadas no dicionário de lemas que permitiram obter a versão D3 do dicionário.

5.3.1 Análise do Dicionário

Antes de o módulo que complementa o transdutor do dicionário com os transdutores dos prefixos existir, as palavras prefixadas eram adicionadas diretamente ao dicionário de lemas do LexMan. As palavras *refazer* e *reescrever* são exemplos de duas palavras prefixadas que estavam no dicionário. Decidiu-se que as palavras prefixadas que passaram a poder ser identificadas com o novo módulo seriam removidas do dicionário de lemas.

Foi feita uma análise ao resultado da geração de palavras e foram extraídas todas as palavras cujo início de palavra coincidia com um prefixo. Cada palavra foi analisada para verificar se era de facto uma palavra prefixada e se devia ser removida do dicionário. Trata-se, de facto, de um conjunto complexo de decisões já que se assume que, quando uma palavra é analisada por prefixação, esta adquire os traços sintáticos e semânticos da palavra de base, o que não acontece quando a forma mais complexa já se autonomizou da forma original. Assim, por exemplo, os pares *levar/relevar* ou *canto/recanto* não estão ligados por prefixação (pelo menos do ponto de vista sincrónico).

Foram removidas 4.978 palavras do dicionário. A remoção dessas palavras provocou erros na geração de compostos que deixavam de poder ser gerados pois não existiam no dicionário de base. Para resolver esse problema, foram introduzidos no dicionário 149 dos adjetivos retirados inicialmente.

Para uma palavra prefixada poder ser identificada usando o novo módulo, é necessário que a base à qual o prefixo é adicionado seja gerada no transdutor do dicionário. Após a remoção das 4.978 palavras, foram adicionadas 158 palavras, correspondentes a bases que não estavam a ser geradas.

5.3.2 Lista de Palavras Desconhecidas

Antes do início deste trabalho, foi criada uma lista que contém 523.529 palavras do corpus CETEMPúblico [Santos e Rocha, 2001] que não eram identificadas pela solução original do LexMan. Essa lista foi processada pela nova versão do LexMan para verificar se havia palavras que passavam a ser identificadas graças à criação do módulo que adiciona os prefixos ao transdutor do dicionário.

Das 523.529 palavras desconhecidas, 44.974 foram identificadas. Dessas 44.974, 33.440 palavras eram palavras reconhecidas como tendo um prefixo. A lista das 33.440 palavras foi dividida em duas, formando-se uma lista de 17.196 palavras prefixadas cujo prefixo acaba em hífen e outra com 16.244 palavras prefixadas cujo prefixo não acaba em hífen.

A lista de palavras prefixadas cujo prefixo não acaba em hífen foi analisada. Os resultados dessa análise permitiu modificar o dicionário de lemas do LexMan. Concluiu-se que 10.391 palavras foram corretamente identificadas como sendo palavras prefixadas. *antibetão* e *recolorir* são duas palavras que pertencem a esse conjunto.

Também se chegou à conclusão que 2.379 palavras deveriam fazer parte do dicionário de lemas. A palavra *criolite* foi adicionada ao dicionário para impedir que seja identificada como a palavra resultante da adição do prefixo *crio* à base *lite*. *degladia* é outra palavra que foi adicionada ao dicionário. Era uma palavra prefixada resultante da adição do prefixo *de* à base *gladia*.

Por fim, 1.965 foram identificadas como contendo um erro ortográfico. Ou seja, palavras que antes eram desconhecidas por terem erros ortográficos podem vir a ser identificadas por esta nova versão do LexMan. Apesar de terem erros ortográficos, essas palavras são identificadas por causa da adição dos prefixos ao transdutor do dicionário. O facto de isso acontecer é pior do que o não identificar a palavra, porque, se esta for desconhecida, o problema pode ser detetado/corrigido mais facilmente. *aeroporo* e *dipensas* são duas palavras que contém um erro ortográfico e que foram identificadas como palavras prefixadas. No 1º exemplo, falta o *t* para formar a palavra *aeroporto*. *aeroporo* está agora a ser identificada como o resultado da adição do prefixo *aero* à base *poro*. No 2º exemplo, falta um *s* para formar a palavra *dispensas*. *dipensas* está agora a ser identificada como o resultado da adição do prefixo *di* à base *pensas*.

Após a conclusão das alterações ao dicionário de lemas, a lista de palavras foi novamente processada. Nesse 2º processamento, foram identificadas um total de 50.047 palavras. Dessas, 33.989 foram consideradas palavras prefixadas. A divisão em 2 conjuntos criou uma lista de 17.575 palavras prefixadas cujo prefixo acaba com hífen. O conjunto final das palavras prefixadas cujo prefixo não acaba com hífen tem um total de 16.414 palavras.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Contribuições

Foram desenvolvidas de forma autónoma as seguintes tarefas:

- Transformação das expressões regulares mais complexas (endereços HTTP, IP e de correio eletrónico, números romanos, números por extenso e referências bíblicas) para transdutores;
- Criação do módulo responsável por efetuar a normalização do texto recebido na entrada do LexMan;
- Criação do transdutor responsável pela identificação de palavras desconhecidas;
- Definição da arquitetura do segmentador e do módulo que forma o segmentador a partir de transdutores previamente gerados;
- Alteração e otimização do processo de construção do transdutor do dicionário;
- Modificação do processo de obtenção das etiquetas durante a análise morfológica, para a 2ª arquitetura implementada. O processo teve de ser modificado para tornar possível a análise de um transdutor correspondente a uma sequência de segmentos;
- Criação do módulo que gera os prefixos a partir de um ficheiro do dicionário de lemas e de um ficheiro que define as regras de junção dos prefixos;
- Alteração do processo de construção dos caminhos do transdutor do dicionário, por forma a adicionar os prefixos ao transdutor;
- Alterações ao processo de obtenção das etiquetas durante a análise morfológica, para que fossem feitas as substituições necessárias nas etiquetas das palavras prefixadas;
- Implementação de restrições na etiquetação de palavras prefixadas;
- Processamento da lista de palavras do CETEMPúblico que eram consideradas desconhecidas quando processadas pela solução original;
- Adaptação do processo de pesquisa por lema à nova arquitetura do LexMan.

Para além disso, foram desenvolvidas, em colaboração com o Cláudio Diniz, as seguintes tarefas:

- Transferência para o dicionário de lemas das expressões regulares mais simples e de regras de recomposição que estavam no RuDriCo.

- Criação de transdutores necessários para o funcionamento da solução que se baseia na utilização da operação *ShortestPath*;
- Testes das soluções implementadas;
- Avaliação das diferentes soluções com um corpus.

O principal objetivo foi cumprido. As tarefas de segmentação e análise morfológica da cadeia STRING são agora realizadas num único módulo, o LexMan. Foram criadas duas abordagens, baseadas em transdutores, que se revelaram mais eficientes do que a solução original. As vantagens das abordagens criadas podem ser resumidas nos seguintes aspetos:

- A diminuição do número de regras a aplicar no RuDriCo permite um aumento do desempenho desse módulo;
- A sintaxe de formação dos segmentos compostos é agora mais simples;
- A definição de prioridades para o processo de segmentação é mais flexível, bastando alterar os valores de custo dos transdutores envolvidos no processo de segmentação.

Além disso, as palavras do transdutor do dicionário são agora complementadas com prefixos. A solução escolhida para a adição de um prefixo a uma base tem as seguintes vantagens:

- A sintaxe escolhida para a geração de prefixos é semelhante à sintaxe que define a geração de palavras para o transdutor do dicionário do LexMan;
- A adição de uma nova palavra no dicionário de lemas pode permitir a identificação de várias palavras prefixadas, sem ser necessário adicionar cada uma delas ao dicionário;
- Uma única entrada na lista de lemas dos prefixos permite a adição de prefixos a várias palavras de diferentes categorias;
- Foi possível remover do dicionário de lemas palavras prefixadas que passaram a ser identificadas através do novo módulo.

De uma forma geral, este trabalho permite ter uma maior cobertura de palavras identificáveis, mais eficiência e mais flexibilidade.

Este trabalho foi usado no projeto QREN OOBIAN[1] na empresa Maisis, numa colaboração com o INESC-ID.

6.2 Trabalho Futuro

Relativamente ao trabalho futuro, são sugeridas duas linhas de ação: continuar a aumentar a cobertura do dicionário do LexMan e desenvolver estratégias que permitam identificar corretamente palavras desconhecidas.

O aumento da cobertura do dicionário do LexMan é importante para melhorar o desempenho global do sistema. Uma forma de atingir esse objetivo é através da integração de outro módulo de processamento de afixos: os sufixos. Uma solução passaria por adicionar sufixos ao módulo Adivinhador, o qual analisaria a terminação de palavras desconhecidas a fim de verificar se uma dada palavra poderia ser o resultado de um processo de derivação, a partir de uma forma de base através adjunção de um sufixo.

Outra solução consiste em adicionar transdutores de sufixos ao transdutor do dicionário. O processo de adição dos sufixos pode, contudo, não ser tão independente do processo de construção dos caminhos do

[1] `http://www.oobian.com/` (data de acesso: 13-05-2013)

transdutor do dicionário como foi o do tratamento dos prefixos. Efetivamente, os sufixos podem introduzir várias transformações na palavra de base à qual são adicionados. Por exemplo, a adição do sufixo *inho* à palavra base *mesa* provoca a remoção da letra *a* do final da palavra base. Por isso, os transdutores dos sufixos não podem ser simplesmente concatenados aos transdutores que formam o transdutor do dicionário, sendo necessário descrever os processos morfotáticos envolvidos na derivação por sufixação. Por outro lado, verificam-se várias restrições de ordem lexical: A par de *mesa* e *mesinha*, verifica-se que a forma com a consoante de ligação *z* não é natural: **mesazinha*. Em contrapartida, para a palavra base *empresa*, verifica-se o contrário: *empresazinha*/**empresinha*.

Relativamente à outra linha de ação, a etapa da normalização ou o módulo Adivinhador podem vir a ser usados como um corretor ortográfico do módulo LexMan. Por exemplo certos carateres, como <*ñ*>, fazem parte da adaptação particular do alfabeto latino a algumas línguas mas que não pertencem ao alfabeto português. Durante a normalização, podem ser feitas substituições de carateres com diacríticos para carateres sem diacríticos e vice-versa. A letra *ñ* poderia ser substituída, por exemplo, por <*nã*>, para dar conta do erro de digitação **ñao*.

O módulo Adivinhador já aplica transformações aos segmentos das palavras desconhecidas, nomeadamente convertendo os carateres maiúsculos e minúsculos que compõem a palavra. Após as transformações, a palavra é composta com o transdutor. A aplicação destas transformações tem como objetivo conseguir emparelhar a nova palavra com um dos caminhos do transdutor, durante a composição. No futuro, podem ser criadas novas estratégias de transformação das palavras desconhecidas.

Agora, o LexMan é o primeiro módulo da STRING, ou seja, o resultado do seu processamento afeta o processamento e o desempenho de todos os outros módulos. Qualquer mudança na arquitetura do LexMan tem de ser gerida com muito cuidado, por forma a não comprometer os restantes módulos da cadeia.

Bibliografia

[Aït-Mokhtar, 1998] Aït-Mokhtar, S. (1998). *L'analyse présyntaxique en une seule étape.* PhD Thesis, Université Blaise Pascal, Clermont-Ferrand.

[Alencar, 2009] Alencar, L. F. (2009). Produtividade morfológica e tecnologia do texto: aspectos da construção de um transdutor lexical do português capaz de analisar neologismos. *Calidoscópio (UNISINOS)*, 7:199 – 220.

[Attia, 2006] Attia, M. A. (2006). An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks, The Challenge of Arabic for NLP/MT Conference. The British Computer Society, London, UK.

[Baptista e Faísca, 2007] Baptista, J. e Faísca, L. (2007). Mapping, filtering and measuring impact of ambiguous words in portuguese. In *Koeva, S.; Maurel, D.; Silberztein, M.; Formaliser les langues avec l'ordinateur: De INTEX à Nooj*, pages 305 – 324. Presses Universitaires de Franche-Comté.

[Beesley, 1998] Beesley, K. R. (1998). Arabic morphology using only finite-state operations. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, pages 50–57, Montreal, Quebec, Canada.

[Beesley, 2001] Beesley, K. R. (2001). Finite-state morphological analysis and generation of Arabic at Xerox Research: Status and plans in 2001. In *ACL Workshop on Arabic Language Processing: Status and Perspective*, volume 1, pages 1–8.

[Bick, 2000] Bick, E. (2000). *The Parsing System Palavras: Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework.* Aarhus University Press.

[Clark, 2002] Clark, A. (2002). Memory-based learning of morphology with stochastic transducers. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 513 – 520, Philadelphia, Pennsylvania.

[Cunha e Cintra, 1986] Cunha, C. e Cintra, L. (1986). *Nova Gramática do Português Contemporâneo.* Edições João Sá da Costa.

[Diniz, 2010] Diniz, C. (2010). *RuDriCo2 - Um Conversor baseado em Regras de Transformação Declarativas.* Master's Thesis, Instituto Superior Técnico – Universidade Técnica de Lisboa.

[Diniz e Mamede, 2011] Diniz, C. e Mamede, N. (2011). *LexMan - Lexical Morphological Analyzer.* Technical report, INESC ID Lisboa.

[Garrido-Alenda et al., 2002] Garrido-Alenda, A., Forcada, M. L., e Carrasco, R. C. (2002). Incremental construction and maintenance of morphological analysers based on augmented letter transducers. In *Proceedings of TMI 2002 (Theoretical and Methodological Issues in Machine Translation, Keihanna/Kyoto, Japan)*, pages 53–62.

[Jurafsky e Martin, 2009] Jurafsky, D. e Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

[Karttunen, 2001] Karttunen, L. (2001). Applications of finite-state transducers in natural language processing. In *Revised Papers from the 5th International Conference on Implementation and Application of Automata*, pages 34 – 46, London, UK. Springer-Verlag.

[Karttunen et al., 1996] Karttunen, L., Chanod, J.-P., Grefenstette, G., e Schiller, A. (1996). Regular expressions for language engineering. *Natural Language Engineering*, 2:305 – 328.

[Levine et al., 1992] Levine, J. R., Mason, T., e Brown, D. (1992). *Lex & yacc.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition.

[Medeiros, 1995] Medeiros, J. C. D. (1995). *Processamento Morfológico e Correcção Ortográfica do Português.* Master's Thesis, Instituto Superior Técnico – Universidade Técnica de Lisboa.

[Mira Mateus et al., 2003] Mira Mateus, M. H., Brito, A. M., Duarte, I., e Faria, I. H. (2003). *Gramática da Língua Portuguesa.* Caminho.

[Mohri, 1996] Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:61 – 80.

[Mota, 1999] Mota, C. (1999). *Enhancement of the morphological analyser of INTEX.* Master's Thesis, Laboratoire d'Automatique Documentaire et Linguistique - Université de Marne-la-Vallée.

[Portela, 2011] Portela, R. (2011). *Identificação Automática de Nomes Compostos.* Master's Thesis, Instituto Superior Técnico – Universidade Técnica de Lisboa.

[Ribeiro, 2003] Ribeiro, R. (2003). *Anotação Morfossintáctica Desambiguada em Português.* Master's Thesis, Instituto Superior Técnico, Lisboa.

[Roche e Schabes, 1995] Roche, E. e Schabes, Y. (1995). Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227 – 253.

[Santos e Rocha, 2001] Santos, D. e Rocha, P. (2001). Evaluating cetempúblico, a free resource for portuguese. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 450–457.

[Silva, 2007] Silva, J. R. (2007). *Shallow processing of Portuguese: From sentence chunking to nominal lemmatization.* Master's Thesis, Faculdade de Ciências da Universidade de Lisboa.

[Simões e Almeida, 2002] Simões, A. M. e Almeida, J. J. (2002). Jspell.pm – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas do XVII Encontro da Associação Portuguesa de Linguística*, pages "485–495", Lisboa 2001.

[Viterbi, 1967] Viterbi, A. J. (1967). Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260 – 269.

[Xerox, 2003] Xerox (2003). Xerox Incremental Parser – Reference Guide. `https://open.xerox.com/Repo/service/XIPParser/public/XIPReferenceGuide.pdf`.

Apêndice A

Expressões Regulares

A.1 Endereços HTTP

```
$entrada =~
   /^((http[s]?:\/\/|ftp:\/\/)|(www.))+(([a-zA-Z0-9\.-]+(([.][a-zA-Z]{2,4})|
     (:[0-9]+))?)|([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}))
     (\/[a-zA-Z0-9\%:\/-_\?\.\'\~#&\+,;=!]\*)?/o
```

A.2 Endereços IP

```
$entrada =~
   /^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
     [.](25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
     [.](25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
     [.](25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
     (\/([12][0-9]|3[012]|[1-9]))?/o
```

A.3 Endereços de Correio Eletrónico

```
$entrada =~
   /^[a-zA-Z0-9._\-\! \$\%&'\*\+\/=?\^\`\{\|\}\~]+\@
     [a-zA-Z0-9-_]+([.][a-zA-Z0-9\-_]+)*[.][a-zA-Z]{2,}/o
```

A.4 Números Romanos

```
$entrada =~
   /^(MM{0,2}((CM)?|(CD)?|D?C{0,3})((XC)?|(XL)?|L?X{0,3})((IX)?|(IV)?|V?I{0,3})|
     C(M|D|C{0,2})((XC)?|(XL)?|L?X{0,3})((IX)?|(IV)?|V?I{0,3})|
     DC{0,3}((XC)?|(XL)?|L?X{0,3})((IX)?|(IV)?|V?I{0,3})|
     X(C|L|X{0,2})((IX)?|(IV)?|V?I{0,3})|
     LX{0,3}((IX)?|(IV)?|V?I{0,3})|
     I(X|V|I{0,2})|
     VI{0,3})
     |
```

```
(mm{0,2}((cm)?|(cd)?|d?c{0,3})((xc)?|(xl)?|l?x{0,3})((ix)?(iv)?|v?i{0,3})|
c(m|d|c{0,2})((xc)?|(xl)?|l?x{0,3})((ix)?|(iv)?|v?i{0,3})|
dc{0,3}((xc)?|(xl)?|l?x{0,3})((ix)?|(iv)?|v?i{0,3})|
x(c|l|x{0,2})((ix)?|(iv)?|v?i{0,3})|
lx{0,3}((ix)?|(iv)?|v?i{0,3})|
i(x|v|i{0,2})|
vi{0,3})/o
```

A.5 Referências Bíblicas

```
my $pentateuco = qr/G(é|ê)n(esis)?|Êx(odo)?|Lev(ítico)?|Núm(eros)?|
                Deut(eron(ó|ô)mio)?/o;

my $historicos = qr/Jos(ué)?|Juí(zes)?|R(t|ute)|Sam(uel)?|Re(is)?|
                Cr(ó|ô)n(icas)?| Par(alip(ó|ô)menos)?|Esd(ras)?|
                Ne(emias)?|Tob(ias)?|J(dt|udite)|Est(er)?|Mac(abeus)?/o;

my $sapienciais = qr/Jó|Sal(mos)?|Prov(érbios)?|Ecle(siastes)?|Cânt(icos)?|
                 Sab(edoria)?|Ecl(o|esiástico)/o;

my $profeticos = qr/Is(aías)?|Jer(emias)?|Lam(entações)?|Bar(uc)?|
                Ez(equiel)?|Dan(iel)?|Oséias|J(oe)?l|Am(ós)?|Abd(ias)?|
                Jon(as)?|Miq(uéias)?|Na(um)?|Hab(acuc)?|Sof(onias)?|
                Ag(eu)?|Zac(arias)?|Mal(aquias)?/o;

my $novo_testamento = qr/M(t|ateus|c|arcos)|L(c|ucas)|Jo(ão)?|
                      A(t|póstolos)|Rom(anos)?|Cor(íntios)?|Gál(atas)?|
                      Ef(ésios)?|F(lp|ilipenses)|Col(ossenses)?|
                      Tes(salonicenses)?|Tim(óteo)?|T(t|ito)|
                      F(lm|il(é|ê)mon)|Heb(reus)?|T(g|iago)|Pe(dro)?|
                      Jud(as)?|Apoc(alipse)?/o;

my $livros = qr/$pentateuco|$historicos|$sapienciais|$profeticos|$novo_testamento/o;

my $livros_romanos = qr/Sam(uel)?|Re(is)?|Cr(ó|ô)n(icas)?|Esd(ras)?|
                     Mac(abeus)?|Cor(íntios)?|Tes(salonicenses)?|Tim(óteo)?|
                     Pe(dro)?|Jo(ão)?|Par(alip(ó|ô)menos)?/o;

my $versiculo = qr/\d+((\.\d+-\d+)|(\.\d+))*/o;

my $intervalo = qr/\d+-\d+/o;

my $intervalo_intercalado = qr/\d+([\.-]\d+)*/o;

my $capitulo = qr/\d+/o;

$entrada =~
```

```
   /^((I{1,3}|IV)\s+$livros_romanos\s+($intervalo_intercalado|$intervalo|$versiculo))|
     ($livros\s+$capitulo\s*,\s*\d+\s*-\s*$capitulo\s*,\s*\d+) |
     ($livros\s+$capitulo\s*,\s*($intervalo|$versiculo)((\s*;\s*)?
     $capitulo\s*,\s*($intervalo_intercalado|$intervalo|$versiculo))*) |
     ($livros\s+$intervalo(\s*,\s*$intervalo)*) |
     ($livros\s+$capitulo)/o
```

A.6 Números Cardinais

```
my $cards_Units = qr/um(a?|as?)|uns|dois|duas|três|quatro|cinco|seis|sete|oito|
                  nove/io;

my $cards_Teens = qr/onze|doze|treze|(qua|ca)torze|quinze|dez[ae]sseis|dez[ae]ssete|
                  dezoito|dez[ae]nove/io;

my $cards_Dozens = qr/vinte|trinta|quarenta|cinquenta|sessenta|setenta|oitenta|
                   noventa/i;

my $cards_Hundreds = qr/duzent[oa]s|trezent[oa]s|quatrocent[oa]s|quinhent[oa]s|
                     seiscent[oa]s|setecent[oa]s|oitocent[oa]s|novecent[oa]s/io;

my $cards_MaxDozens = qr/($cards_Dozens\s+e{1}\s+$cards_Units)|$cards_Dozens|
                      $cards_Teens|dez|$cards_Units/io;

my $cards_OneHundreds = qr/cento\s+e{1}\s+$cards_MaxDozens|cem/io;

my $cards_MaxHundreds = qr/$cards_Hundreds\s+e{1}\s+$cards_MaxDozens|
                        $cards_Hundreds/io;

my $cards_OneThousands = qr/(($cards_MaxHundreds|$cards_OneHundreds|
                         $cards_MaxDozens)\s+mil(\s+|,\s+|\s+e{1}\s+)
                         ($cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens))|
                         (($cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens)
                         \s+mil)|mil(\s+|,\s+|\s+e{1}\s+)($cards_MaxHundreds|
                         $cards_OneHundreds|$cards_MaxDozens)|mil/io;

my $cards_OneMillions = qr/(um milhão(\s+|,\s+|\s+e{1}\s+)($cards_OneThousands|
                        $cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens))/io;

my $cards_MaxMillions = qr/(($cards_OneThousands|$cards_MaxHundreds|
                        $cards_OneHundreds|$cards_MaxDozens)\s+milhões
                        (\s+|,\s+|\s+e{1}\s+)($cards_OneThousands|
                        $cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens))/io;

$entrada =~
   /^($cards_MaxMillions|$cards_OneMillions|$cards_OneThousands|$cards_MaxHundreds|
     $cards_OneHundreds|$cards_MaxDozens)|
```

```
     ([0-9]+([.\s+][0-9]{3})*([,][0-9]+)*\s+mil)/i
```

A.7 Números Enas

```
my $enas = qr/dúzias?|vintenas?|trintenas?|dezenas?|centenas?|bil(i|h)(ão|ões)|
           deci(li|lh)(ão|ões)|milh(ão|ões|ar(es)?)|noni(li|lh)(ão|ões)|
           oc?ti(li|lh)(ão|ões)|quatri(li|lh)(ão|ões)|quinti(li|lh)(ão|ões)|
           sep?ti(li|lh)(ão|ões)|sexti(li|lh)(ão|ões)|tri(li|lh)(ão|ões)/io;

$entrada =~
   /^([0-9]+([.\s+][0-9]{3})*([,][0-9]+)*\s*(mil)?\s*$enas(\s+de\s+$enas)*)|
   ($enas(\s+de\s+$enas)*)|
   ($cards_MaxMillions|$cards_OneMillions|$cards_OneThousands|$cards_MaxHundreds|
   $cards_OneHundreds|$cards_MaxDozens)(\s+$enas(\s+de\s+$enas)*)/i
```

A.8 Números Ordinais

```
my $ords_Units = qr/primeir[ao]s?|segund[ao]s?|terceir[ao]s?/io;

my $ords_Teens = qr/décim[ao]s?\s+$ords_Units/io;

my $ords_Dozens = qr/vigésim[ao]s?|trigésim[ao]s?|quadragésim[ao]s?|
                  quinquagésim[ao]s?|sexagésim[ao]s?|sep?tuagésim[ao]s?|
                  octogésim[ao]s?|nonagésim[ao]s?/io;

my $ords_Hundreds = qr/ducentésim[ao]s?|tr[ie]centésim[ao]s?|quadringentésim[ao]s?|
                    quingentésim[ao]s?|se(x|is)centésim[ao]s?|sep?tingentésim[ao]s?|
                    oct(o|in)gentésim[ao]s?|non(in)?gentésim[ao]s?/io;

my $ords_MaxDozens = qr/($ords_Dozens\s+$ords_Units)|$ords_Teens|$ords_Units/io;

my $ords_OneHundreds = qr/centésim[ao]s?\s+$ords_MaxDozens/io;

my $ords_MaxHundreds = qr/$ords_Hundreds\s+$ords_MaxDozens/io;

my $ords_OneThousands = qr/(milésim[ao]s?\s+($ords_MaxHundreds|
                        $ords_OneHundreds|$ords_MaxDozens))|
                        ((centésim[ao]s?\s+milésim[ao]s?)\s+($ords_MaxHundreds|
                        $ords_OneHundreds|$ords_MaxDozens))|
                        (($ords_Dozens\s+$ords_Units)|$ords_Dozens|$ords_Teens|
                        décim[ao]s?)\s+milésim[ao]s?\s+($ords_MaxHundreds|
                        $ords_OneHundreds|$ords_MaxDozens))|/io;

my $ords_Millions = qr/(($ords_Hundreds|centésimos?|($ords_Dozens\s+$ords_Units)|
```

```
                    $ords_Dozens|$ords_Teens|décim[ao]s?)\s+)?milionésim[ao]s?\s+
                    ($ords_OneThousands|$ords_MaxHundreds|$ords_OneHundreds|
                    $ords_MaxDozens)/io;

$entrada =~
   /^($ords_Millions|$ords_OneThousands|$ords_MaxHundreds|$ords_OneHundreds|
     $ords_MaxDozens)/i
```

A.9 Números Fracionários

```
my $fracs = qr/terços?|(($cards_Dozens\s+e{1}\s+$cards_Units)|$cards_Dozens|
            $cards_Teens)\s+avos)/io;

$entrada =~
   /^(($cards_MaxMillions|$cards_OneMillions|$cards_OneThousands|
     $cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens)\s+$fracs)|
     \d{1,3}\s+$fracs/i
```

A.10 Números Ordinais/Fracionários

```
my $ordsfracs_Units = qr/quart[ao]s?|quint[ao]s?|sext[ao]s?|sétim[ao]s?|
                      oitav[ao]s?|non[ao]s?/io;

my $ordsfracs_Teens = qr/décim[ao]s?\s+$ordsfracs_Units|undécim[ao]s?|
                      duodécim[ao]s?|tredécim[ao]s?/io;

my $ordsfracs_Dozens = qr/vigésim[ao]s?|trigésim[ao]s?|quadragésim[ao]s?|
                       quinquagésim[ao]s?|sexagésim[ao]s?|sep?tuagésim[ao]s?|
                       octogésim[ao]s?|nonagésim[ao]s?/io;

my $ordsfracs_Hundreds = qr/ducentésim[ao]s?|tr[ie]centésim[ao]s?|
                         quadringentésim[ao]s?|quingentésim[ao]s?|
                         se(x|is)centésim[ao]s?|sep?tingentésim[ao]s?|
                         oct(o|in)gentésim[ao]s?|non(in)?gentésim[ao]s?/io;

my $ordsfracs_MaxDozens = qr/($ordsfracs_Dozens\s+$ordsfracs_Units)|
                          $ordsfracs_Dozens|$ordsfracs_Teens|décim[ao]s?|
                          $ordsfracs_Units/io;

my $ordsfracs_OneHundreds = qr/centésim[ao]s?\s+$ordsfracs_MaxDozens|
                            centésim[ao]s?/io;

my $ordsfracs_MaxHundreds = qr/$ordsfracs_Hundreds\s+$ordsfracs_MaxDozens|
                            $ordsfracs_Hundreds/io;
```

```
my $ordsfracs_OneThousands =
   qr/(milésim[ao]s?\s+($ordsfracs_MaxHundreds|$ordsfracs_OneHundreds|
   $ordsfracs_MaxDozens))|
   ((centésim[ao]s?\s+milésim[ao]s?)\s+($ordsfracs_MaxHundreds|$ordsfracs_OneHundreds|
   $ordsfracs_MaxDozens))|
   (centésim[ao]s?\s+milésim[ao]s?)|
   (($ordsfracs_Dozens\s+$ordsfracs_Units)|$ordsfracs_Dozens|
   $ordsfracs_Teens|décim[ao]s?)\s+milésim[ao]s?\s+($ordsfracs_MaxHundreds|
   $ordsfracs_OneHundreds|$ordsfracs_MaxDozens))|
   (décim[ao]s?\s+milésim[ao]s?)|milésim[ao]s?/io;

my $ordsfracs_Millions =
   qr/ (($ordsfracs_Hundreds|centésimos?|($ordsfracs_Dozens\s+$ordsfracs_Units)|
   $ordsfracs_Dozens|$ordsfracs_Teens|décim[ao]s?)\s+)?milionésim[ao]s?\s+
   ($ordsfracs_OneThousands|$ordsfracs_MaxHundreds|$ordsfracs_OneHundreds|
   $ordsfracs_MaxDozens))|
   (($ordsfracs_Hundreds|centésimos?|($ordsfracs_Dozens\s+$ordsfracs_Units)|
   $ordsfracs_Dozens|$ordsfracs_Teens|décim[ao]s?)\s+milionésim[ao]s?)|
   milionésim[ao]s?/io;

my $fracs_Aux = qr/quartas?|quint[ao]s?|sext[ao]s?|sétim[ao]s?|
                oitav[ao]s?|non[ao]s?|décim[ao]s?|centésim[ao]s?|
                mil(ion)?ésim[ao]s?/io;

my $ordsfracs_Quartos = ((um|1)\s+quarto)|((dois|três|2|3)\s+quartos)/i

$entrada =~
   /^($ordsfracs_Millions|$ordsfracs_OneThousands|$ordsfracs_MaxHundreds|
     $ordsfracs_OneHundreds|$ordsfracs_MaxDozens)|
     (($cards_MaxMillions|$cards_OneMillions|$cards_OneThousands|
     $cards_MaxHundreds|$cards_OneHundreds|$cards_MaxDozens)\s+$fracs_Aux)|
     \d{1,3}\s+$fracs_Aux|
     $ordsfracs_Quartos/i
```