Ferramenta Preliminar para Auxílio à formalização de Textos em Lógica

Lucas Martins de Souza

Projeto Final de Graduação

Centro Técnico Científico
Departamento de Informática
Curso de Engenharia de Computação

Orientador: Prof. Edward Hermann Haeusler

Rio de Janeiro
junho de 2014

Lucas Martins de Souza

Ferramenta Preliminar para Auxílio à formalização de Textos em Lógica

Relatório de Projeto Final, apresentado ao programa do Curso de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação.

Prof. Edward Hermann Haeusler
Orientador
Departamento de Informática — PUC–Rio

Rio de Janeiro, 3 de junho de 2014

Agradecimentos

Agradeço à minha família por ter me dado suporte durante minha jornada até o presente momento sem eles nada disso seria possível.

Ao meu orientador Edward Hermann, por sua paciência, apoio e ideias ao longo deste trabalho.

À minha namorada Sofia, que é uma pessoa maravilhosa. Sua compania e compreensão foram essenciais.

E principalmente aos meus amigos. Sem eles eu não teria forças para chegar onde cheguei. Espero conseguir ser tão bom para eles quanto eles são para mim.

Resumo

Martins de Souza, Lucas; Haeusler, Edward Hermann. **Ferramenta Preliminar para Auxílio à formalização de Textos em Lógica** . Rio de Janeiro, 2014. Relatório de Projeto Final — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho consiste no estudo e na implementação de um protótipo capaz de formalizar sentenças escritas em português através do uso de uma ferramenta léxica. A lógica escolhida para a formalização é a lógica proposicional, muito usada para fins didáticos.

Palavras–chave

Lógica Proposicional. Processamento de Linguagem Natural. Expressões Regulares. Formalização de Sentenças.

Abstract

Martins de Souza, Lucas; Haeusler, Edward Hermann. **Preliminary Tool for Sentences Formalization**. Rio de Janeiro, 2014. — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work consists in studying and developing a prototype software capable of formalizing simple sentences written in Portuguese, using an online tool for support on natural language processing. The logic chosen for the formalization process is the propositional logic, widely used for introducing students into the logics field of study in computer science.

Keywords

Propositional Logic. Natural Language Processing. Regular Expressions. Sentences Formalization.

Sumário

Lista de Figuras

Lista de tabelas

1
Introdução

Este capítulo tem o objetivo de apresentar o leitor ao projeto, assim como familiarizá-lo com as ferramentas e tecnologias utilizadas para a construção de um simples formalizador de sentenças, produto deste trabalho.

Na primeira seção, é apresentado o projeto sob uma visão geral, introduzindo a motivação que levou ao desenvolvimento deste trabalho. Na seção dois o foco é apresentar a lógica proposicional e exemplos de formalização. Na seção três é discutida a ferramenta léxica F-EXT-WS. Na quarta seção é discutida a ferramenta léxica LX Suite.

1.1
Introdução e Visão Geral do Projeto

O principal meio de comunição e registro de informações desde os tempos antigos é a linguagem escrita. Por este meio nossa espécie foi capaz de transmitir informações para as gerações posteriores. Hoje em dia, muita informação que geramos se encontra armazenada em relatórios e textos. A proposta desse trabalho é caminhar na direção de estruturar essas informações para tornar possível o seu processamento por um computador.

A metodologia usada para obter este resultado é a transformação da linguagem escrita para uma estrutura de dados geradas a partir de tags dados por uma ferramenta semântica e a partir desses dados gerar expressões lógicas. Estas expressões poderão ser capazes de inferir asserções sobre o texto. Quanto mais complexa a linguagem lógica escolhida para essa transformação, maior a expressividade do texto traduzido sendo possível inferir asserções de forma mais adequada.

A linguagem lógica escolhida para este trabalho foi a lógica proposicional, a mais simples disponível. O motivo desta escolha foi tornar o projeto mais simples servindo como introdução ao problema, facilitando que outros estudantes usem-o como degrau para o aprendizado e introdução a lógicas mais complexas.

A lógica proposicional é ensinada em todos os cursos do Departamento de Informática e utilizada para apresentar aos alunos de Informática as

possibilidades e o poder computacional das ferramentas que é possível criar usando lógica como ferramenta.

Além da conversão para a lógica proposicional, um dos problemas abordados por este trabalho é a complexidade da interpretação de sentenças em português.

Para resolver este problema foram usadas duas ferramentas: uma desenvolvida pelo laboratório LEARN (PUC-RIO), que se chama FE-EXT-WS; e outra desenvolvida pelo grupo NLX do Departamento de Informática da Universidade de Lisboa.

Estas ferramentas realizam diversas análises. Dentro da gama de opções fornecida pelas ferramentas para análise, foi utilizado o POS Tagger, opção fornecida tanto pelo F-EXT-WS quanto pelo LX Suite. Esta função consegue enriquecer sentenças em português associando a cada palavra sua possível classe gramatical (verbo, advérbio, pronome etc.).

As sentenças enriquecidas por essa análise são usadas como entrada para o algoritmo responsável por construir as expressões em lógica proposicional. Deve-se ressaltar que, quando nos utilizamos de algoritmos para a análise de linguagem natural não podemos garantir o reconhecimento de todo o conjunto de expressões da língua. É preciso utilizar um conjunto de controle para que seja possível garantir, para esse grupo, o reconhecimento das expressões e assim a geração de expressões lógicas corretas.

1.2 A Lógica Proposicional

"Em lógica e matemática, uma lógica proposicional (ou cálculo sentencial) é um sistema formal no qual as fórmulas representam proposições que podem ser formadas pela combinação de proposições atômicas usando conectivos lógicos e um sistema de regras de derivação, que permite que certas fórmulas sejam estabelecidas como 'teoremas' do sistema formal." (Van Dalen)

1.2.1 O que é uma proposição?

Uma proposição é uma expressão declarativa que pode ser classificada como verdadeira ou falsa em um certo contexto. Podemos ressaltar os seguintes exemplos:

Todo número inteiro é racional.

A casa é bonita.

Paulo é flamenguista.

Regina tem olhos verdes.

Exemplos de sentenças que não são proposições:

Nossa!

A casa bonita.

Não fume!

Será que vai chover?

1.2.2 Proposições atômicas e moleculares

As proposições podem ser divididas em dois grupos, moleculares e atômicas. As proposições atômicas são aquelas em que não é possível quebrar em proposições menores (Correia). Por exemplo:

31 é um número primo.

Roberto é brasileiro.

Proposições moleculares são compostas por proposições atômicas ou por outras proposições moleculares. São exemplos de proposições moleculares:

Maria e Jorge gostam de novela.

Se Jorge está em casa então Jorge está vendo novela.

Maria e Angela estão jantando, se e somente se, Angela foi ao mercado.

1.2.3 Operadores

Para relacionar proposições e sermos capazes de criar formas lógicas é necessário o uso de operadores lógicos. Estes podem ser os seguintes:

Operador	Símbolo	Exemplo
e	$\wedge$	q $\wedge$ r
ou	$\vee$	q $\vee$ r
se ... então..	$\Rightarrow$	q $\Rightarrow$ r
...se e somente se..	$\Leftrightarrow$	q $\Leftrightarrow$ r
não	$\neg$	$\neg$q

Tabela 1.1: Tabela mostrando os operadores usados na lógica proposicional.

Cada operador acima possui uma tabela verdade que serve para verificarmos as fórmulas criadas. Abaixo segue a tabela verdade de cada operador.

q	r	q $\wedge$ r
Verdade	Verdade	Verdade
Verdade	Falso	Falso
Falso	Verdade	Falso
Falso	Falso	Falso

q	r	q $\leftrightarrow$ r
Verdade	Verdade	Verdade
Verdade	Falso	Falso
Falso	Verdade	Falso
Falso	Falso	Verdade

q	r	q $\vee$ r
Verdade	Verdade	Verdade
Verdade	Falso	Verdade
Falso	Verdade	Verdade
Falso	Falso	Falso

q	r	q $\rightarrow$ r
Verdade	Verdade	Verdade
Verdade	Falso	Falso
Falso	Verdade	Verdade
Falso	Falso	Verdade

q	$\neg$q
Verdade	Falso
Falso	Verdade

Figura 1.1: As tabelas verdade dos operadores usados na lógica proposicional.

1.2.4 Formalização.

Podemos agora formalizar sentenças usando o que foi descrito acima. Para fins de demonstração, abaixo encontra-se dois exemplos.

Exemplo 1 : Um caso direto.

A sentença:

Maria e Jorge gostam de novela.

Pode ser traduzida para :

$$q \wedge r$$

Onde:

q = Maria gosta de novela.

r = Jorge gosta de novela.

Exemplo 2: Um caso mais complexo.
A sentença:

Uma condição necessária para que duas retas r1 e r2 sejam paralelas é que sejam coplanares e não se interceptem.

Primeiramente, devemos reescrever essa sentença como:

As retas r1 e r2 serem complanares e não se interceptarem é uma condição necessária para que elas sejam paralelas

Se uma sentença é do tipo 'A condição é necessária para B' então ela deve ser reescrita como 'Se B então A'. Desta forma obtemos:

Se as retas r1 e r2 são paralelas, então r1 e r2 são coplanares e não se interceptam.

Por fim obtemos:

Se as retas r1 e r2 são paralelas, então r1 e r2 são coplanares e r1 e r2 não se interceptam.

Temos finalmente:

$$q \rightarrow (r \wedge \neg(s))$$

Onde:

q = as retas r1 e r2 são paralelas.

r = as retas r1 e r2 são coplanares.

s = as retas r1 e r2 se interceptam.

1.3 O F-EXT-WS

Um sistema de classificação léxica é capaz de classificar palavras de um texto baseado em sua semâtica e contexto, um exemplo é a ferramenta F-EXT-WS (FEXTWSDoc). Esta é desenvolvida e mantida pelo grupo de pesquisa LEARN no Departamento de Informática da PUC– Rio. Ele recebe como entrada um texto em linguagem natural e produz como saída um texto onde cada palavra recebe um rótulo, que classifica esta palavra de alguma forma. O texto de entrada pode ser escrito tanto em inglês quanto em português. O português foi a linguagem escolhida. Para o português, os seguintes tipos de funcionalidade são suportados:

Part-of-speech (POS) tagging

Phrase chunking

Clause

Semantic role labeling

A ferramenta desenvolvida utiliza o Part-of-speech (POS) tagging. Por esta razão, a seguir só será explicada esta funcionalidade.

1.3.1 Part-of-speech (POS) tagging

É o processo de etiquetagem de cada palavra de um texto com o objetivo de representar a sua função gramatical. Etiquetas POS classificam palavras em categorias, baseadas na regra que elas participam no contexto em que elas aparecem. Para exemplificar o funcionamento veja este exemplo:

Jorge	NPROP
vai	V
a	PREP
o	ART
baile	N
se	PROPESS
e	KC
somente	PDEN
se	KS
Maria	NPROP
for	V
a	PREP
o	ART
baile	N

Tabela 1.2: Tabela mostrando um exemplo de classificação POS.

A primeira coluna corresponde a cada palavra do texto, e a segunda coluna corresponde as etiquetas POS. As etiquetas POS para o português que o F–EXT–WS usa são as dispostas no site do projeto Lácio-Web(LacioWeb).

A ferramenta esta disponível para uso pelo site, ou através de um web service. A ferramenta proposta neste trabalho utiliza o F-EXT-WS através do web service. (Deitel). Na próxima página é apresentado uma tabela com o significado de todas as etiquetas oferece.

POS	TAG
ADJETIVO	ADJ
ADVÉRBIO	ADV
ADVÉRBIO CONECTIVO SUBORDINATIVO	ADV-KS
ADVÉRBIO RELATIVO SUBORDINATIVO	ADV-KS-REL
ARTIGO (def. ou indef.)	ART
CONJUNÇÃO COORDENATIVA	KC
CONJUNÇÃO SUBORDINATIVA	KS
INTERJEIÇÃO	IN
NOME	N
NOME PRÓPRIO	NPROP
NUMERAL	NUM
PARTICÍPIO	PCP
PALAVRA DENOTATIVA	PDEN
PREPOSIÇÃO	PREP
PRONOME ADJETIVO	PROADJ
PRONOME CONECTIVO SUBORDINATIVO	PRO-KS
PRONOME PESSOAL	PROPESS
PRONOME RELATIVO CONECTIVO SUBORDINATIVO	PRO-KS-REL
PRONOME SUBSTANTIVO	PROSUB
VERBO	V
VERBO AUXILIAR	VAUX
SÍMBOLO DE MOEDA CORRENTE	CUR
CONTRAÇÕES e ÊNCLISES	—+
MESÓCLISES	—!
ADDITIONAL TAGS	
Estrangeirismos	—EST
Apostos	—AP
Dados	—DAD
Números de Telefone	—TEL
Datas	—DAT
Horas	—HOR
Disjunção	—[—]

Tabela 1.3: Todos os tipos de TAGS do F-EXT-WS.

1.4

O LX Suite

O projeto LX Suite é outro sistema de classificação léxica, desenvolvido pelo grupo NLX (Grupo NLX) do Departamento de Informática da Universidade de Lisboa. Assim como o F-EXT-WS, esta ferramenta recebe como entrada um texto em português e produz como saída um texto rotulado. O LX Suite é capaz de fazer diversas análises em textos em português, mas para este projeto será usada somente a análise POS (Part-of-Speech). As outras análises possíveis são:

Phrase chunking

Tokenizer

Featurizer

Lemmatizer

Lemmatizer and Featurizer

Part-of-speech (POS) tagging

1.4.1 Part-of-speech (POS) tagging com LX Suite

O LX suite é capaz de fazer uma análise bastante aprofundada em textos, possuindo um número superior de tags, sendo capaz de discernir inclusive tempos verbais em frases. Como exemplo é possível citar a frase abaixo:

Maria e Jorge gostam de novela.

Para esta frase temos:

Maria	PNM
e	CJ
Jorge	PNM
gostam	GOSTAR/V#pi-3p
de	N
novela	CN
.	PNT

Tabela 1.4: Frase com rótulos LX Suite.

Note que o LX Suite não só informa a forma infinitiva do verbo como também em que pessoa ele está sendo conjugado. No caso, o verbo 'gosta', tem seu infinitivo, 'GOSTAR', e está sendo conjugado na terceira pessoa do plural no presente do indicativo ('pi-3p').

Abaixo está uma tabela com algumas das etiquetas que o LX Suite usa para sua análise. Existem mais etiquetas que as exemplificadas abaixo (EtiquetasPOSLXSuite).

Tag	**Categoria**	**Examplo**
ADJ	Adjectives	bom, brilhante, eficaz, ...
ADV	Adverbs	hoje, já, sim, felizmente, ...
CARD	Cardinals	zero, dez, cem, mil, ...
CJ	Conjunctions	e, ou, tal como, ...
CL	Clitics	o, lhe, se, ...
CN	Common Nouns	computador, cidade, ideia, ...
DA	Definite Articles	o, os, ...
DEM	Demonstratives	este, esses, aquele, ...
DFR	Denominators of Fractions	meio, terço, décimo, %, ...
DGTR	Roman Numerals	VI, LX, MMIII, MCMXCIX, ...
DGT	Digits	0, 1, 42, 12345, 67890, ...
DM	Discourse Marker	olá, ...
EADR	Electronic Addresses	http://www.di.fc.ul.pt, ...
EOE	End of Enumeration	etc
EXC	Exclamative	ah, ei, etc.
GER	Gerunds	sendo, afirmando, vivendo, ...
GERAUX	Gerund "ter"/"haver"in compound tenses	tendo, havendo ...
IA	Indefinite Articles	uns, umas, ...
IND	Indefinites	tudo, alguém, ninguém, ...
INF	Infinitive	ser, afirmar, viver, ...
INFAUX	Infinitive "ter"/"haver"in compound tenses	ter, haver ...
INT	Interrogatives	quem, como, quando, ...
ITJ	Interjection	bolas, caramba, ...
LTR	Letters	a, b, c, ...
MGT	Magnitude Classes	unidade, dezena..
MTH	Months	Janeiro, Dezembro, ...
NP	Noun Phrases	idem, ...

Tabela 1.5: Etiquetas LX Suite

2
Proposta

Neste capítulo é descrita a proposta do projeto, é exposto um fluxograma de como ele está feito, assim como os seus requisitos funcionais e não funcionais. Também é descrito o estado da arte em que a aplicação se encaixa.

Na primeira seção é apresentada a proposta do projeto; a segunda seção trata de projetos similares; a terceira seção apresenta um fluxograma que representa o processo todo da aplicação em uma forma geral. Por fim, na quarta seção, são apresentados os requisitos funcionais que se espera da aplicação gerada por este projeto.

2.1
A Proposta

O presente trabalho tem como proposta ser um estudo de caso sobre formalização de sentenças escritas em linguagem natural, servindo para o compreendimento da complexidade de se estruturar sentenças em português usando lógica.

Devido à sua complexidade e por estar fora do escopo deste projeto, não foi construido um analisador semântico para cuidar do processamento de linguagem natural envolvido no projeto.

O projeto usará como base a análise Part-of-Speech (POS), que será feita pela ferramenta criada pelo laboratório LEARN da PUC–RIO, o POS Tagger, e pela ferramenta criada pelo grupo NLX da Universidade de Lisboa, o LX Suite (LXSuite). Ambas são ferramentas léxicas capazes de receber uma string de caracteres e retornar uma outra string que relaciona cada palavra a uma etiqueta com um significado gramatical.

O escopo do projeto abrangerá, além do estudo e a aplicação de lógica proposicional, a criação de uma metodologia para transformar textos, com suas palavras devidamente categorizadas, em fórmulas lógicas de forma genérica. Isto tudo será implementado em uma ferramenta que receberá um texto e retornará as expressões lógicas referentes a cada uma de suas frases. A comunicação com a ferramenta léxica ocorrerá pela internet a partir de um web server. A ferramenta final criada será open source e será disponibilizada

em um repositório online (Google Code) para poder ser utilizada como base para outros trabalhos futuros.

Além destes objetivos propostos, a ferramenta também poderá ser usada como verificador para checar a formalização de sentenças simples e suas consequências, por alunos que estejam aprendendo lógica proposicional.

Sendo assim, os principais objetivos deste projeto são apresentar e documentar uma solução para formalização de sentenças e interpretação de linguagem natural, assim como produzir um trabalho piloto que possa ser estudado e expandido por alunos que queiram conhecer e se aprofundar mais nesta área.

2.2 Análise de Projetos Similares

Atualmente não existem trabalhos de formalização de textos em forma lógica. Porém existem algumas iniciativas em ferramentas de apoio ao aprendizado de lógica, como a ferramenta do grupo de lógica (LoLITA), da Universidade Federal do Rio Grande do Norte, que desenvolveu uma ferramenta de auxilio ao aprendizado de lógica, o Logicamente.

O Logicamente possui o objetivo de auxiliar o aluno de lógica no seu aprendizado. É uma ferramenta online onde o aluno é capaz de:

- Gerar automaticamente fórmulas em uma complexidade dada.
- Definir uma linguagem assim como seus conectivos.
- Traduzir sentenças entre duas linguagens.
- Contruir tabelas verdade e Tableaux.

Tudo isto se encontra disponível em um ambiente online no seguinte endereço: http://lolita.dimap.ufrn.br/logicamente/.

2.3 Fluxograma

Na página seguinte é mostrado um fluxograma simplificado de como o processo todo se dará. Note que o capítulo 3 descreve melhor cada parte do diagrama.

Na imagem abaixo cada forma geométrica possui um significado (por exemplo quadrados significam processos, losangos perguntas). Estes significados podem ser aprendidos no site do Wikipedia (Flowcharts).Este fluxograma foi feito usando a ferramenta online gliffy, que encontra-se disponível no site (Gliffy).

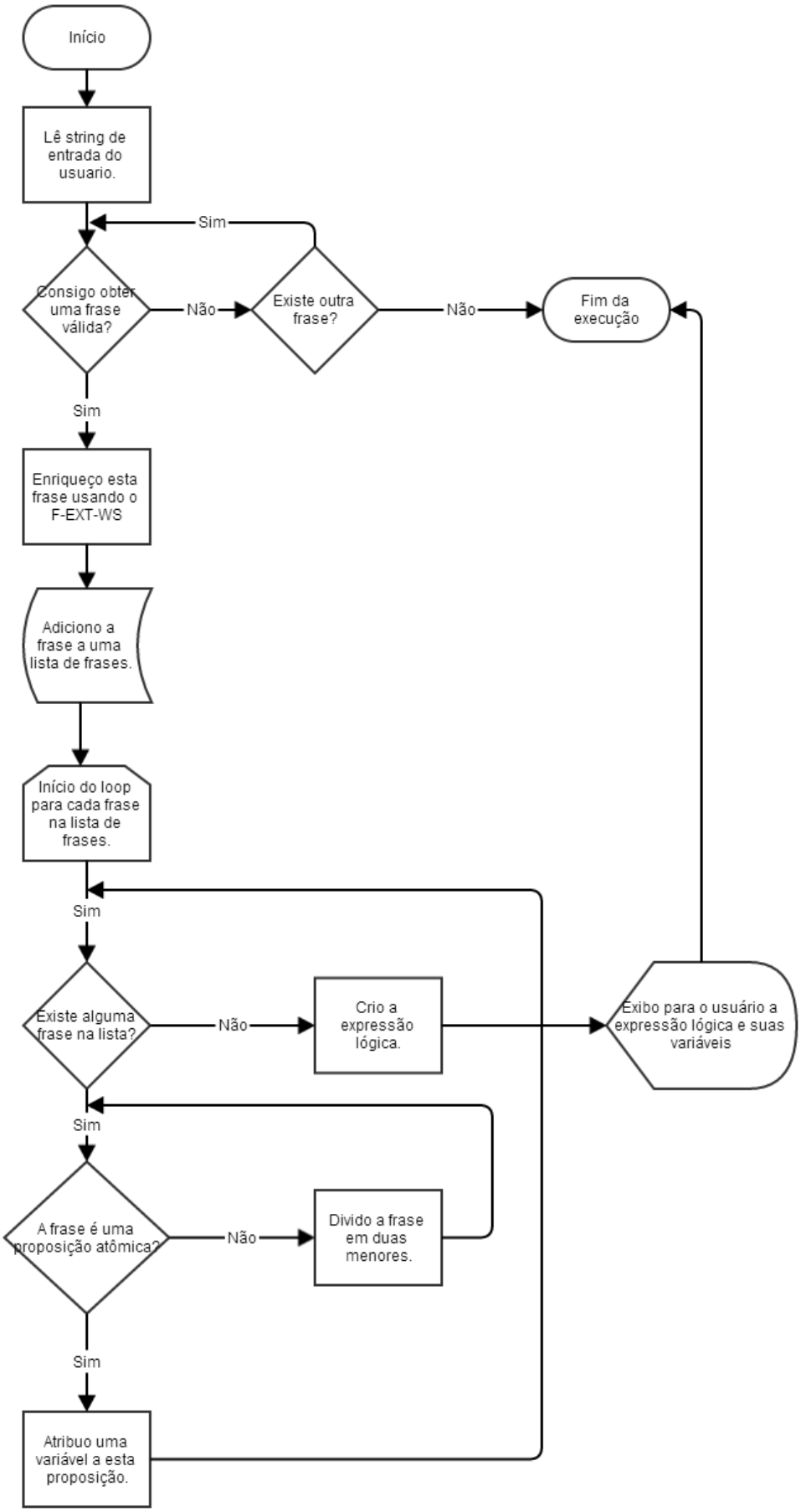

Figura 2.1: Fluxograma simplificado da aplicação proposta.

2.4 Requisitos

Nas subseções seguintes serão expostos os requisitos funcionais, e não funcionais da aplicação gerada por este projeto.

2.4.1 Escopo

Como o projeto visa construir uma plataforma simples de tradução de sentenças, ela contemplará tanto **requisitos funcionais** quanto **requisitos não funcionais**. Além disso compreende o escopo deste projeto estudar as ferramentas que existem no mercado e uma metodologia a ser empregada para resolver este problema.

2.4.2 Público Alvo

Em um primeiro momento, a plataforma em questão se destina a estudantes da Pontifícia Universidade Católica do Rio de Janeiro que estejam interessados em se aprofundar em assuntos como lógica e argumentação formal.

2.4.3 Requisitos Não Funcionais

[**R1**] Documentação do programa usando técnicas conhecidas, como diagrama de classes, exemplos de uso e definição formal de entradas e saídas.

[**R2**] Utilização de documentação interna do código, com o intuito de facilitar a modificação da plataforma no futuro. Geração de uma documentação baseada em Javadoc(Javadoc).

[**R3**] Gerenciamento de versões do código utilizando a ferramenta SVN com a plataforma fornecida pelo Google Code. (Google Code)

2.4.4 Requisitos Funcionais

[**R4**] O programa deve ser capaz de traduzir do português sentenças simples com os operadores 'e', 'ou', 'se... então' e 'se e somente se' para lógica proposicional.

[**R5**] O programa deve ser capaz de traduzir os casos documentados de sentenças compostas com mais de um operador lógico.

[**R6**] O programa deve ser capaz de traduzir um texto inteiro.

[**R7**] O programa deve ser capaz de, para cada sentença, listar suas proposições atômicas, assim como sua formalização em lógica proposicional.

[**R8**] O programa deve ser capaz de ser expansível sem ser necessário mudar o algoritmo principal. Caso seja necessário expandi-lo, basta adicionar casos novos ao seu dicionário interno.

[**R9**] O programa não deve gerar dois símbolos diferentes para uma mesma frase atômica. Deve manter a consistencia da simbologia.

[**R10**] Deve ser possível a troca da ferramenta de análise léxica apenas modificando as etiquetas usadas e o módulo de comunicação com o web service da ferramenta.

3
Terminologia

Neste capítulo será descrito o que são expressões regulares e a terminologia usada na linguagem de programação Java para estas expressões. Assim como a descrição dos casos usados.

Na primeira seção é apresentado o que são expressões regulares e o que é possivel realizar com o seu uso das mesmas. Na segunda seção é avaliado alguns artifícios que a linguagem de programação Java proporciona.

3.1
Expressões Regulares

Em teoria da computação, expressão regular é uma sequência de caracteres que formam um padrão de busca. É comumente usada para comparar duas sequência de caracteres ou para verificar se uma frase se encaixa em um padrão pré-estabelecido.

Cada caractere em uma expressão regular é considerado um metacaractere (Metacaractere) com um sentido especial ou um caracter comum com seu sentido literal. Juntos eles são usados para identificar material textual de um determinado padrão ou processar um certo número de exemplos, que podem variar desde uma igualdade exata a uma semelhança bem geral.

Um exemplo de uso de expressão regular poderia ser: Obter todas as palavras de um texto que começam com 'casa'. Poderiamos ter: 'casa', 'casamento', 'casal' e etc..

Um processador de expressões regulares é capaz de traduzi-las para um autômato finito não determinístico (NFA), a cadeia de caracteres que queremos reconhecer é então inserida neste autômato. Se ele a reconhecer significa que a expressão regular captura esta cadeia de caracteres.

3.1.1
Sintaxe

A sintaxe de uma expressão regular depende do processador usado para sua compilação. Sua sintaxe é subdividida em duas categorias, átomos

e quantificadores. A primeira consiste de metacaracteres que possuem um sentido, podendo ou não ser "escapados" para significar seu sentido literal.

Quantificadores representam quantas vezes é esperado um determinado átomo no texto; por exemplo, podemos querer capturar uma palavra que contenha um certo átomo 'ba' que venha acompanhado de duas vezes o átomo 'na'. Assim capturando a palavra 'banana'.

3.1.2 Padrão POSIX

O padrão POSIX possuí três conjuntos de sintaxe: BRE, ERE e SRE para, respectivamente, expressões regulares, básicas, estendidas e simples. Abaixo se encontra uma tabela com os operadores do conjunto BRE, o mais usado.

Metacharacter	Description
.	Matches any single character
[ ]	A bracket expression. Matches a single character that is contained within the brackets.
[^]	Matches a single character that is not contained within the brackets.
^	Matches the starting position within the string.
$	Matches the ending position of the string.
( )	Defines a marked subexpression.
\n	Matches what the nth marked subexpression matched, where n is a digit from 1 to 9.
*	Matches the preceding element zero or more times.
{m,n}	Matches the preceding element at least m and not more than n times.

Tabela 3.1: Padrão BRE Posix.

3.2 Recursos Especiais

Nesta seção serão explicados alguns recursos adicionais que existem no processador Java e que foram usados para o tratamento das frases neste projeto.

3.2.1 Lookahead Positivo e Negativo

O Lookahead é sub-dividido em duas formas, positiva e negativa. O Looakahead negativo é indispensável caso estejamos interessados em capturar algo não seguido de algo. Como acontece na maioria dos processadores de

expressão regular, não podemos usar uma classe de caractere negada para capturar um 'q' não seguido de um 'u'.

A solução para este problema é o Lookahead negativo, ele é construido de um par de parênteses com o primeiro parênteses seguido de um ponto de interrogação '?' e um de exclamação '!'. Por exemplo para o caso descrito no paragrafo anterior teriamos : 'q(?!u)'.

O lookahead positivo funciona da mesma forma, porém com sintaxe um pouco diferente. A expressão 'q(?=u)' captura, um 'q' seguido de um 'u', porém sem incluir o 'u' no conjunto capturado. O lookahead positivo é constituído de um parênteses seguido de um ponto de interrogação e um sinal de igual. Um exemplo seria:

"Alberto gosta de jogar cartas."

Aplicando o lookahead negativo, **ar(?!t)**, capturamos o 'ar' da palavra 'jogar' e não da palavra 'cartas'. Isto é, capturamos um 'ar' que não está seguido de um 't'.

Aplicando o lookahead positivo, **ar(?=t)**, capturamos o 'ar' da palavra 'cartas' da palavra 'jogar'. Isto é, capturamos um 'ar' que está seguido de um 't'.

Um detalhe importante é que qualquer expressão regular pode ser inserida dentro de um lookahead positivo, porém não dentro de um negativo. Somente pode ser inserida uma expressão regular se e somente se esta capturar ao menos um caractere.

3.2.2 Lookbehind Positivo e Negativo

Assim como o Lookahead é dividido em duas formas, positiva e negativa, o Lookbehind também. Novamente queremos capturar algo, porém desta vez queremos capturar algo somente se existe um conjunto de caracteres que antecede este que queremos capturar.

Analogamente, se queremos capturar somente se uma sequência de caracteres antecede outra, usamos o lookbehind **positivo**, caso o objetivo seja capturar somente se uma sequência **não** precede outra dada, usamos o lookbehind **negativo**.

A sintaxe do lookbehind positivo consiste em um parênteses seguido de um ponto de interrogação, um sinal de "menor igual", um sinal de igual, o texto que antecede o que queremos capturar, outro parêntese e a sequência de caracteres que queremos capturar.

Para o negativo é a mesma sintaxe, porém trocando o sinal de igual por um ponto de exclamação. Por exemplo no texto:

"João vai comprar bananas."

Aplicando o lookbehind negativo, **(?<!ba)na**, capturamos o segundo 'na' da palavra banana. Isto é, capturamos um 'na' que não vem depois de um 'ba'.

Aplicando o lookbehind positivo, **(?=!ba)na**, capturamos o primeiro 'na' da palavra banana. Isto é, capturamos um 'na' que vem depois de um 'ba'.

4
Funcionamento da Aplicação

Neste capítulo é descrito em detalhes o funcionamento da aplicação. Ao longo deste capítulo são formalizados, a medida que é explicado, dois exemplos de frases. Para estes exemplos será usado o F-EXT-WS.

Na seção primeira seção é tratado o enriquecimento de dados usando a ferramenta léxica. Na segunda é explcitado o algoritmo de diferenciação entre proposições atômicas e moleculares. Na terceira seção é retratado o método de substituição de conectivos usado. Na quarta seção é explicado como se dá a recursão em uma proposição. Finalmente, na ultima seção, é descrito o processo de criação da forma lógica.

4.1
Enriquecimento de Dados

Este item retrata detalhadamente o processo de criação de uma expressão lógica. Este processo é repetido para todas as frases de um texto inserido pelo usuário. Para fins de ilustração, faremos um exemplo de formalização ao longo da explicação.

Após a entrada de um texto pelo usuário, o código inícia o processo de enriquecimento dessa informação. Na primeira etapa deste processo são usados os métodos obtidos pelo servidor que provém o F-EXT-WS. Estes foram recebidos pelo código Java através de uma tecnologia chamada RMI (Remote Method Invocation). Ao submetermos o código recebemos como retorno uma string com o texto original misturado com etiquetas dadas pelo serviço. Vejamos um exemplo de entrada e saída, ao submetermos a string:

"Maria gosta de dormir mas Maria gosta de festas. Se Maria e Jorge forem na festa logo Maria usará seu perfume preferido."

Dada a string acima temos como resposta:

"Maria NPROP gosta V de PREP dormir V mas KC Maria NPROP gosta V de PREP festas N. Se KS Maria NPROP e KC Jorge NPROP forem V em PREP a ART festa N logo ADV Maria NPROP usará N seu PROADJ perfume N preferido PCP."

Devemos separar esta string em sentenças e descartar as que não podem ser proposições. Fazemos isso checando a pontuação da sentença. Uma proposição é uma expressão declarativa. Isto para nós será traduzido como: "A sentença terá que terminar com '.' (ponto final). Caso termine com outra pontuação, não será considerada uma proposição.". O que realmente estamos fazendo é a transformação de uma string em uma lista de frases que podem ser proposições.

A partir disto, transformamos estas strings em listas de classes do tipo DuplaTextoProcessado. Esta classe relaciona uma palavra a sua tag, além de conter métodos que retornam ou esperam como entrada uma classe DuplaTextoProcessado, isto facilitará sua manipulação no futuro.

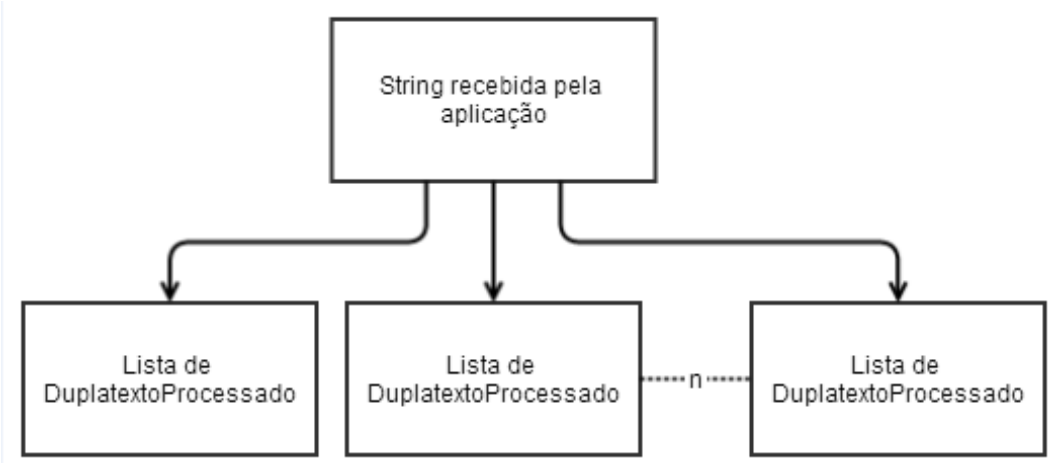

Figura 4.1: Estrutura de decomposição da String Entrada após ser enriquecida.

4.2 Diferenciando Proposições Atômicas de Moleculares

O próximo passo deste algoritmo é, a partir de uma lista de classes DuplaTextoProcessado, dar origem às classes ProposicaoAtomica e ProposicaoMolecular. Estas duas classes herdam da classe abstrata chamada Proposição. Elas possuem cada uma sua respectiva lista de DuplaTextoProcessado que representa o texto que as gerou.

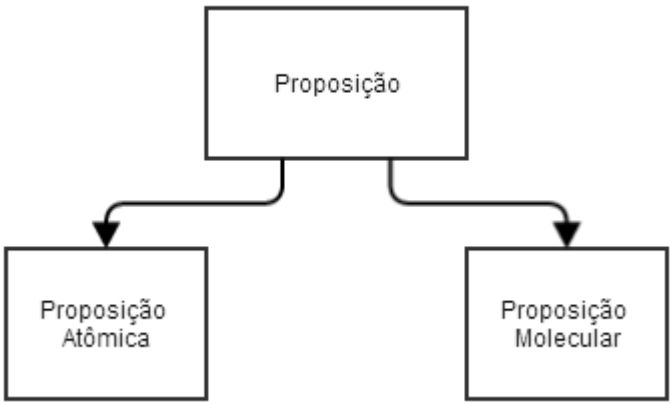

Figura 4.2: Diagrama de classes de proposição.

Note que para um texto grande teremos uma lista de classes do tipo ProposicaoAtomica e uma outra lista de classes do tipo ProposiaoMolecular. A ideia de diferenciarmos as proposições existe, pois para proposições moleculares podemos formalizá-las de forma direta, atribuindo uma variável a ela. Já as moleculares devemos tratar com mais detalhes.

O processo para identificar se uma proposição é molecular ou atômica é bem simples. É percorrida cada palavra (DuplaTextoProcessado) da frase original e é checada a sua etiqueta POS. Se esta etiqueta for 'KC' (conectivo) podemos afirmar que esta frase não é uma proposicão atômica.

Mesmo assim podem existir casos em que não há conectivos na frase e mesmo assim esta não é uma proposição atômica. Um exemplo disso é a implicação:

"Se Maria for na festa logo Maria usará seu perfume preferido."

Esta frase enriquecida ficaria:

"Se KS Maria NPROP for V em PREP a ART festa N logo ADV Maria NPROP usará N seu PROADJ perfume N preferido PCP."

Não existem conectivos ('KC') e mesmo assim temos uma proposição molecular. Para resolver este caso tratamos de outra forma. Verificamos a existência de uma etiqueta 'KS' (Conjunção subordinativa) antes de uma etiqueta 'ADV' (Advérbio). Caso isto aconteça temos uma implicação, que obviamente não é uma proposição atômica.

Então, retornando ao nosso exemplo, ficamos com duas proposições moleculares que são dadas por duas listas de estruturas do tipo DuplaTextoProcessado:

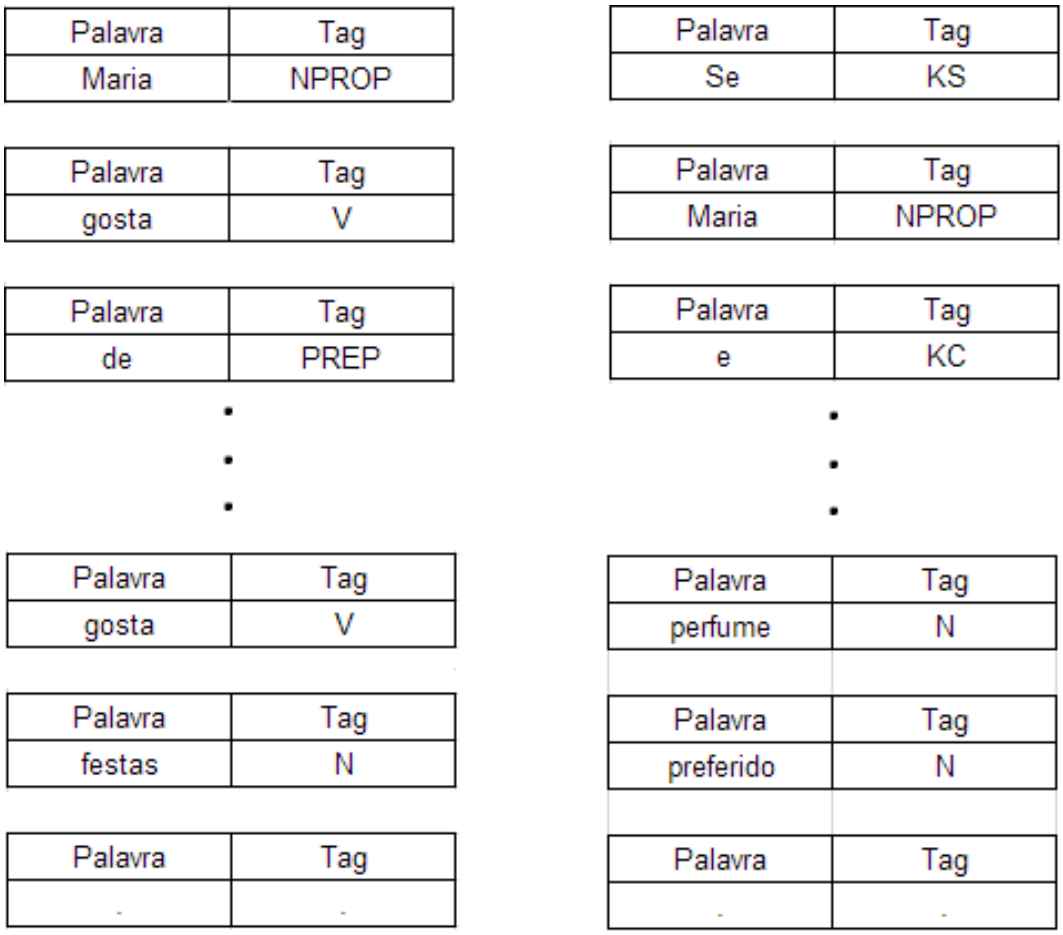

Figura 4.3: Lista de DuplaTextoProcessado para o exemplo

Agora temos que separar esta proposição molecular em outras proposições menores.

4.3 Substituição de Conectivos - Formas Correspondentes

A estrutura de uma proposição molecular que tem como operador o 'e' ou o 'ou' tem uma composição gramatical idêntica.

"Maria e Julia gostam de bonecas."
"Maria NPROP e KC Julia NPROP gostam V de PREP bonecas N ."

Assim como:

"Maria ou Julia gostam de bonecas."

"Maria NPROP ou KC Julia NPROP gostam V de PREP bonecas N ."

Porém estas expressões têm sentenças lógicas diferentes. Outro problema encontrado é a existência de vários conectivos diferentes, porém com o mesmo significado, que podem ser usados na construção de uma conjunção ou disjunção. Por exemplo, para a conjunção abaixo:

"Maria gosta de dormir tarde mas Maria sempre acorda cedo."
"Maria NPROP gosta V de PREP dormir V tarde N mas KC Maria NPROP sempre ADV acorda V cedo ADV."

O conectivo desta conjunção não é a palavra 'e'. Por isso, para sabermos qual símbolo devemos usar para formalizar, vamos ter que primeiro criar um **dicionário de conectivos** onde conseguimos dizer se o conectivo KC é um 'e' ou um 'ou'.

Logo, o próximo passo tomado pelo algoritmo é encontrar o conectivo na sentença e substituir sua palavra por uma equivalente, 'e' ou 'ou'. Neste exemplo podemos observar isto acontecer na primeira sentença, onde o conectivo 'mas' será substituido pelo conectivo 'e'.

4.4 Recursão em uma Proposição

Nesta etapa do processo todas as proposições estão devidamente diferenciadas (moleculares ou atômicas) e com sua estrutura definida em uma classe Proposição, que contém a sua lista de classes DuplaTextoProcessado. Neste momento começa o processo de encontrar proposições atômicas dentro de cada proposição molecular. Para cada proposição será usado o mesmo procedimento. Trata-se de uma recursão. O caso base é definido como a identificação da proposição atômica. Caso não seja atômica o algoritmo tenta quebrar-la em duas. Caso contrário, esta proposição é retornada pois é atômica. Segue um fluxograma ilustrando o processo:

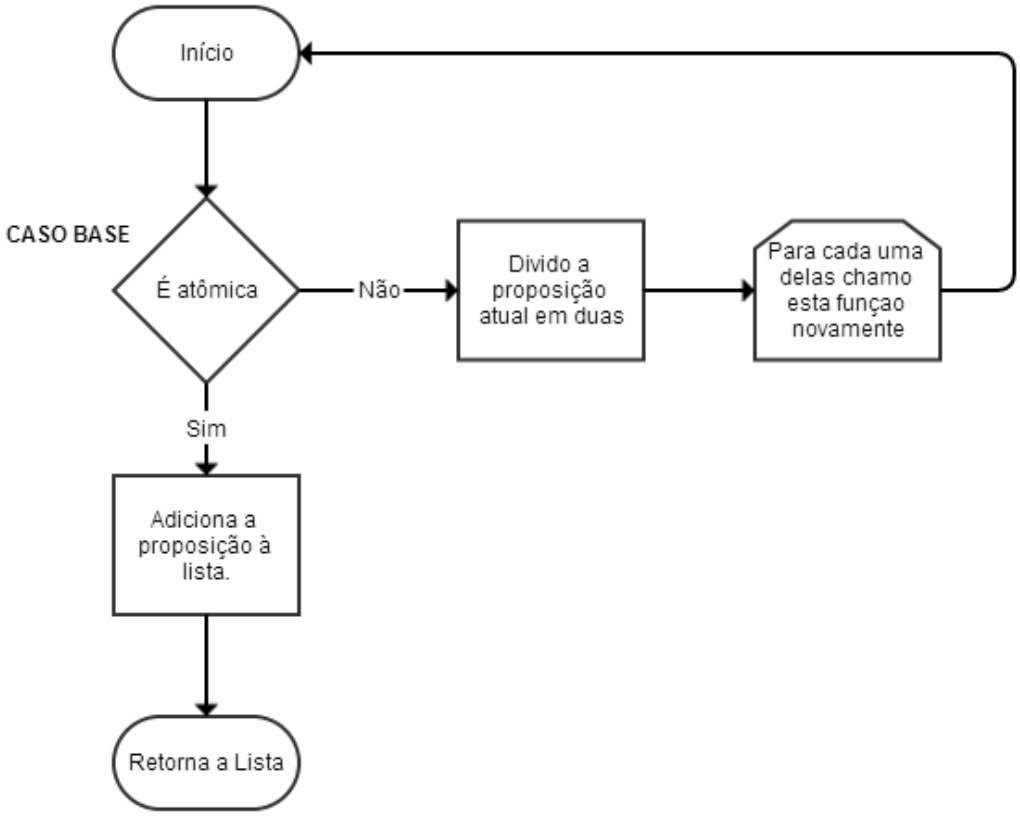

Figura 4.4: Fluxograma ilustrando o processo de recursão

No primeiro passo deste processo é usada uma string com a proposição corrente. Nela, cada palavra que pussui uma tag considerada útil é substituido(No próximo capitulo existe uma lista que mostra as tags que usamos para esta classificação). Assim, para a primeira proposição do exemplo, é obtido:

"Maria gosta de dormir tarde e Maria sempre acorda cedo."

Somente considerando as etiquetas da sentença acima pode-se escrever:

"NPROP V de V N KC NPROP ADV V ADV."

Onde 'Maria' é relacionada à 'NPROP', 'gosta' a 'V', 'dormir' a 'V', 'tarde' a 'N', 'e' a 'KC' e assim por diante.

Nesta etapa é introduzida a estrutura ProposicaoTag. Esta estrutura é identificada por uma expressão regular e possui uma lista de expressões regulares (**?**). Estas expressões conseguem obter as proposições internas da proposição a que ela se refere. Esta estrutura também guarda o operador lógico principal dela.

Continuando o processo, é usado agora um dicionário para obter a estrutura escolhida para processar a proposição. Esta estrutura (dicionário) liga

Propoposição TAG	
String	IdRegExp
List<RegExp>	ListaRegExp
String	OperadorLogico

Tabela 4.1: Estrutura simplificada ProposiçãoTag

uma chave a uma outra estrutura capaz de prover as informações necessárias para capturar as proposições atômicas de uma proposição molecular.

Note que cada uma destas chaves mencionadas acima é uma expressão regular. O dicionário é capaz de listar estas expressões (as chaves do dicionário). É percorrida esta lista para tentar encontrar uma expressão regular que case com a forma da proposição que o algoritmo está processando. Novamente, a primeira frase deste exemplo foi re-escrita para:

"NPROP V de V N KC NPROP ADV V ADV."

A expressão regular que casa com esta forma pode ser a abaixo:

".*?V.*?KC.*?$"

A estrutura encontrada com esta expressão regular possui duas expressões regulares.Lembre-se que o objetivo é sempre dividir uma proposição em outras duas proposições menores então, basta apenas duas expressões regulares. Elas são:

".*?(?=KC)"

E

"(?<=KC).*?$"

Para a primeira expressão regular é possivel obter a expressão:

"NPROP V de V N"

Para a segunda expressão:

"NPROP ADV V ADV."

Substituindo os tags pelas suas respectivas palavras, é obtido:

"Maria gosta de dormir tarde" "Maria sempre acorda cedo."

Neste ponto, é possível dividir a proposição em duas menores. O algoritmo agora repete o processo para cada uma, porém ambas satisfazem o caso base, pois ambas são atômicas.

Para a segunda frase do exemplo é necessário realizar recursão para chegar às proposições atômicas. Este caso será resolvido para ilustrar o processo. A expressão:

“Se Maria e Jorge forem na festa logo Maria usará seu perfume preferido.”

Temos:

“KS NPROP KC NPROP V em ART N ADV NPROP N seu N preferido .”

A expressão regular encontrada para este caso é:

“KS .*? V .*? ADV .*?$”

As seguintes expressões regulares são as associadas a esta estrutura:

“(?<=KS).*? (?=ADV)”

E

“(?<=ADV).*?$”

Como resultado delas foi obtido, respectivamente:

“NPROP KC NPROP V em ART N”

E

“NPROP N seu N preferido .”

Realizando a substituição por palavras obtemos, respectivamente:

“Maria e Jorge forem em a festa” “Maria usará seu perfume preferido .”

Note que a primeira proposição não é atômica, logo o algorítmo continuará usando ela como base agora. É obtido então:

“Maria e Jorge forem em a festa”

Logo

“NPROP KC NPROP V em ART N”

A expressão regular encontrada para esta é:

“NPROP KC NPROP .*?V .*?.”

Com esta são obtida as seguintes expressões:

“KC0.*(?=V)”

E

“.*?(?<=KC) ”

Que por fim leva as seguintes proposições:

“Jorge forem em a festa .” “Maria forem em a festa .”

Com isso é possível encontrar as cinco proposições atômicas contidas na string dada como exemplo, são elas:

“Maria gosta de dormir tarde” “Maria sempre acorda cedo.” “Maria usará seu perfume preferido .” “Jorge forem em a festa .” “Maria forem em a festa .”

É possível notar que a ferramenta não se preocupa com o tempo verbal de nenhuma proposição e que tanto o LX Suite quanto o F-EXT-WS quebraram palavras que ele considera composta em suas componentes. No caso, ele quebrou a palavra ‘forem’ em ‘for’ e ‘em’, e quando substituimos de volta o verbo V, sobrou o resíduo (’em’) no texto da proposição.

4.5 Criação da Forma Lógica

Desde que temos todas as proposições atômicas da nossa string, é possível encontrar os operadores que as relacionam, formando assim a forma lógica. O processo para isso é bem similar ao anterior. Novamente é usado de recursão para resolver o problema.

Na seção anterior, ao chegar no caso base, é retornada a proposição atômica que foi encontrada. Desta vez, é retornado o símbolo que representa aquela proposição. Após o retorno, é concatenado o operador lógico contido na ProposiçaoTag para aquele caso, e então é chamada a recursão para a próxima proposição molecular, concatenando o resultado da recursão com o que já existia antes, chegando à expressão final. A figura abaixo ilustra o processo:

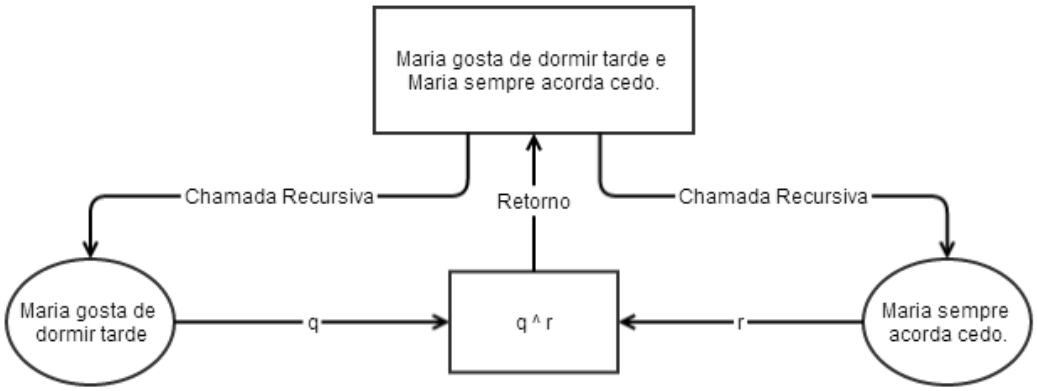

Figura 4.5: Ilustração da construção da forma lógica para a primeira sentença do exemplo

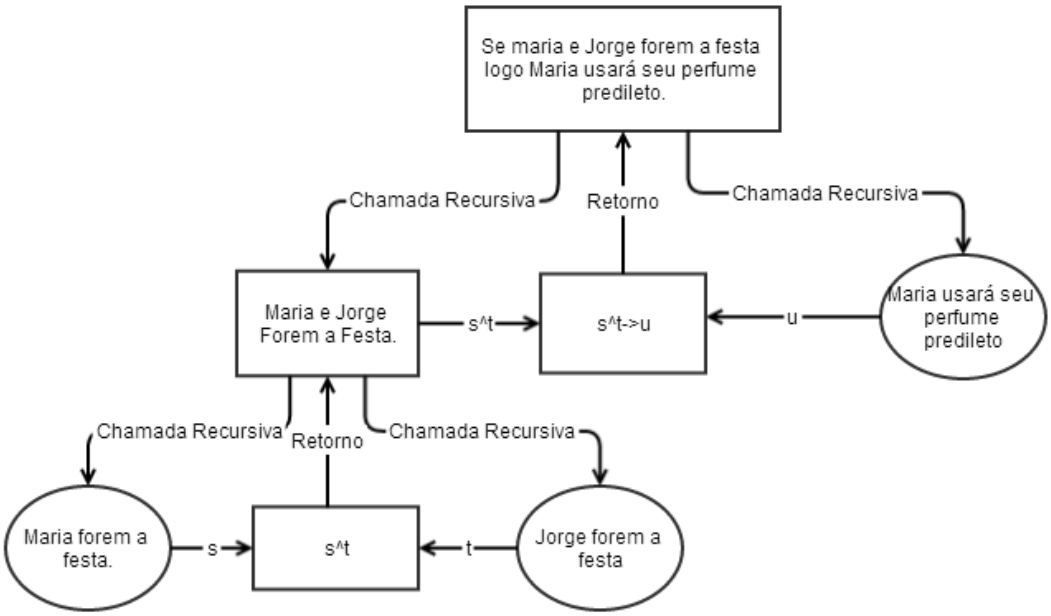

Figura 4.6: Ilustração da construção da forma lógica para a segunda sentença do exemplo

5
Limites e Expansão

Neste capítulo serão definidas as entradas e saídas da que devem ser fornecidas e o que é esperado da aplicação, suas limitações, e como é possível superá-las expandindo a ferramenta. Este capítulo é essencial e condiz com a proposta do projeto de servir como exemplo base para outros alunos que quiserem continuar este trabalho.

Na primeira seção são descritos os dicionários usados pela aplicação. Na segunda seção são descritos os tags que são relevantes para a aplicação. A seguir na seção três é exibido o diagrama de classes da aplicação. Por fim na seção quatro e cinco são definidas as entradas e saídas do programa.

5.1
Dicionários da Aplicação

No capítulo anterior foram introduzidos alguns dos dicionários que a ferramenta utiliza para funcionar. Nesta seção eles serão definidos e será explicado como eles funcionam, por que existem e como incrementá-los.

5.1.1
Dicionário de Conectivos

O dicionário de conectivos existe para ser possível relacionar conectivos a seu significado lógico. Como foi dito anteriormente, um operador lógico (por exemplo 'e') pode ser relacionado a vários conectivos diferentes.

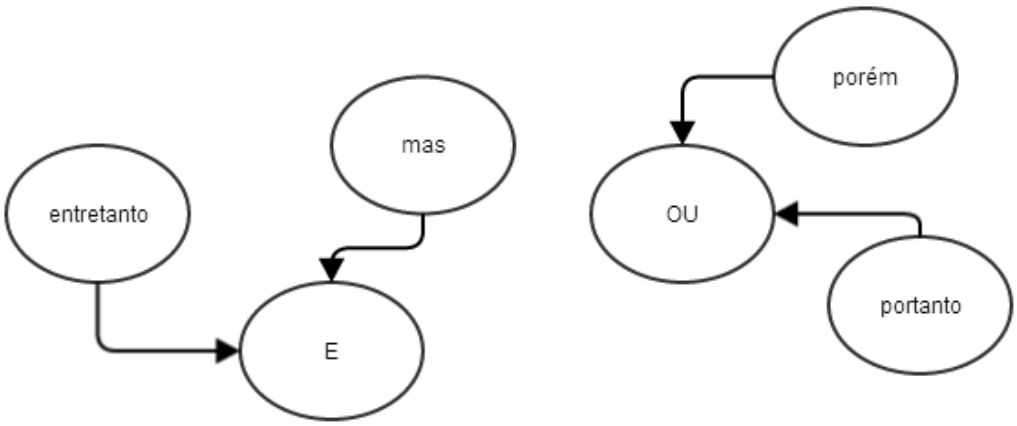

Figura 5.1: Exemplo de palavras com o mesmo significado que os conectivos 'E' e 'OU'

Como todos os dicionários desta aplicação, ele implementa o design pattern (Singleton). Ele possui duas funções. Uma chamada **carregaDicionário** que o inicializa com as relações pré-definidas e outra chamada **obtemConectivo**, onde, a partir de uma palavra ele retorna se esta é equivalente a um 'e', ou um 'ou'.

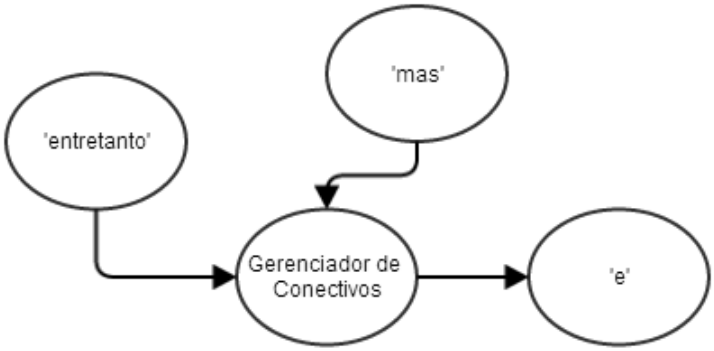

Figura 5.2: O dicionário de conectivos funciona como tradutor.

É possível expandir este dicionário, modificando a função carregaDicionário, adicionando a linha de código:

$$map.put([Palavra], [operadorEquivalente])$$

5.1.2 Dicionário de Padrões

O Dicionario de Padrões é essencial para ser possível reconhecer a estrutura de uma proposição. Ele relaciona um inteiro a uma proposiçãoTag. Ele consegue devolver a expressão regular que identifica uma ProposicaoTag a partir de um inteiro identificador, assim como obter uma proposiçãoTag baseado em uma expressão regular.

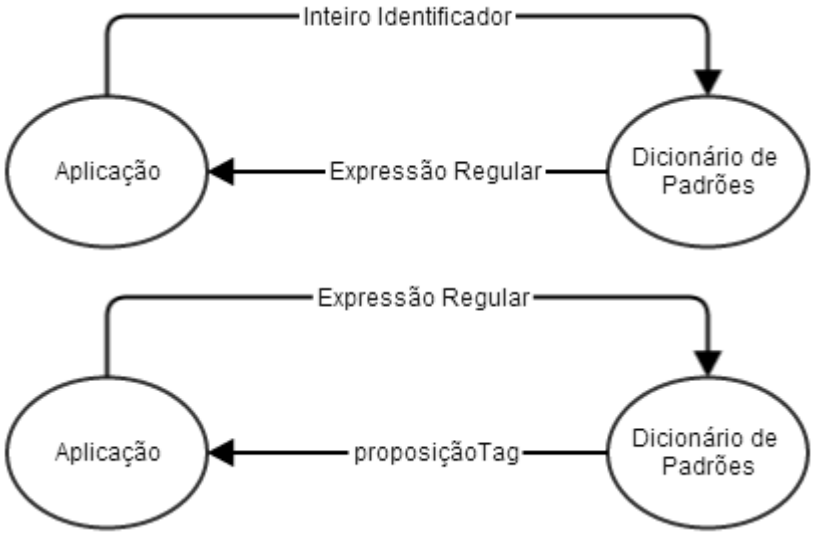

Figura 5.3: Exemplo de operações realizadas pelo dicionário de Padrões.

No exemplo do capítulo anterior, quando foi mencionado que era procurado uma proposição em uma lista de ProposicaoTag, se tratava deste dicionário. A lista de ProposicaoTag é o conjunto de elementos deste dicionário.

Para adicionar um novo elemento a este dicionário deveremos inserir as linhas como descrito no exemplo da figura abaixo:

```
// Caso #2: Maria e Jorge gostam de novela.    ->  NPROP KC NPROP V de N.
lista = new ArrayList<Pattern>();
lista.add(Pattern.compile("KC0.*(?=V0)")); // Match "KC0 Jorge "
lista.add(Pattern.compile(".*?(?<=KC0) ")); // Match "Maria KC0 "
ProposicaoTag a1 = new ProposicaoTag("NPROP KC NPROP .*?V .*?\\.", lista, "EOU", -1);
_map.put(_numeroProposicaoTagNoMap,a1);
_listTags.add(a1);
_numeroProposicaoTagNoMap++;
```

Figura 5.4: Acrescentando um padrão ao dicionário por código.

Este é um exemplo para proposições do tipo "Maria e Jorge gostam de novela", ou seja, algo similar a "NPROP KC NPROP V de N". Para este caso serão definidas primeiro as duas expressões regulares que têm a capacidade de decompor esta proposição. Seguido disto é definida a expressão regular capaz de identificar esse caso, por fim ela é adicionada ao HashMap deste dicionário

e adicionada à lista de elementos do dicionário assim como é incrementado o número de elementos do mesmo.

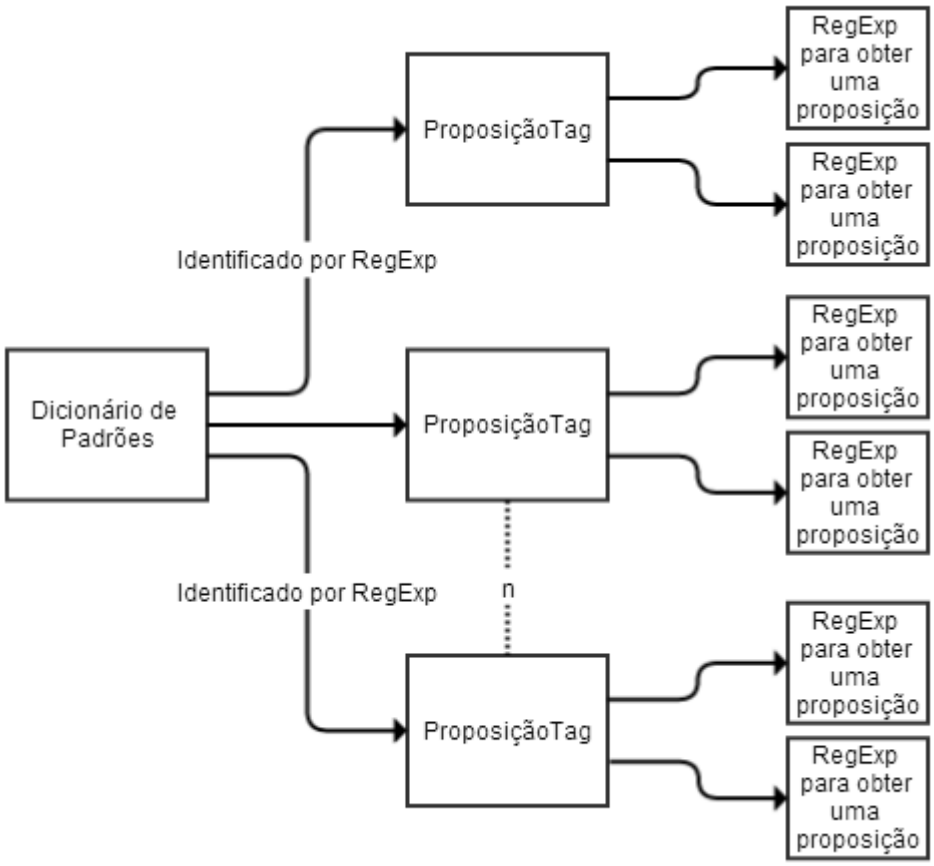

Figura 5.5: Estrutura simplificada do Dicionário de Padrões

Deve-se notar que a ordem de adição dos elementos nesse dicionário é relevante! É obrigatório ordená-los do caso mais abrangente ao mais restrito assim assegurando que obtemos a ProposicaoTag correta para o tipo de proposição que queremos formalizar.

É preciso organizar desta forma a lista pois é possível existir uma interseção entre os conjuntos que duas expressões regulares distintas podem capturar. Sendo assim se começada a lista pelo caso mais restrito é menor a chance de a expressão capturar algo que ela não deveria.

Outro detalhe importante é o da numeração dos tags que está sendo usado. Para ser possível encontrar qual tag está ligado à palavra correta na momento de restaurar a sentença original, é preciso numerar os tags que usamos nas expressões regulares da lista. Na imagem acima é possível ver que os tags vem acompanhados de um número (zero no caso). Este tag numerado a partir de sua posição na proposição original, ou seja, o primeiro KC da proposição deve ser númerado como KC0, o segundo, KC1 e assim por diante.

5.2 Tags Relevantes

Tanto a ferramenta F-EXT-WS quanto o LX Suite nos fornece uma quantidade grande de tags que podemos usar (como mostrado no capítulo 2). Porém esta ferramenta não precisou utilizar todos os tags disponiveis. Os que foram usados nessa versão final da aplicação para o F-EXT-WS são os abaixo (seus equivalentes foram utilizados ao trabalhar com o LX Suite):

Podemos expandir esses tags para aumentar a expressividade (número de proposições que coneseguimos formalizar) através da possibilidade de criar expressões regulares mais específicas. Para expandir o número de tags, devemos inserir novos tags na classe GerenciadorDeTags, dentro do método inicializaTags() da seguinte forma:

```
private void inicializaTags()
{
    _tags.add(new Tag("KC"));
    _tags.add(new Tag("V"));
    _tags.add(new Tag("VAUX"));
    _tags.add(new Tag("KS"));
    _tags.add(new Tag("ADV"));
    _tags.add(new Tag("NPROP"));
    _tags.add(new Tag("PROPESS"));
    _tags.add(new Tag("PDEN"));
    _tags.add(new Tag("PROP"));
    _tags.add(new Tag("ART"));
    _tags.add(new Tag("N"));
}
```

Figura 5.6: Exemplo de tags inseridos no sistema.

5.3 Diagrama de Classes

Para melhor entendimento da aplicação, segue abaixo uma imagem do diagrama de classes da aplicação, no modo em que ela se encontra em sua versão final.

O Diagrama é simplificado, não contendo os métodos nem as propriedades de cada classe. Toda esta documentação pode ser encontrada no JavaDoc em anexo. Logo, a figura abaixo também não contempla as cinco classes relacionadas à conexão com o web service do F-EXT-WS nem à classe relacionada com a implementação interface. Somente as classes relacionadas ao algoritmo usado para implementar a minha metodologia.

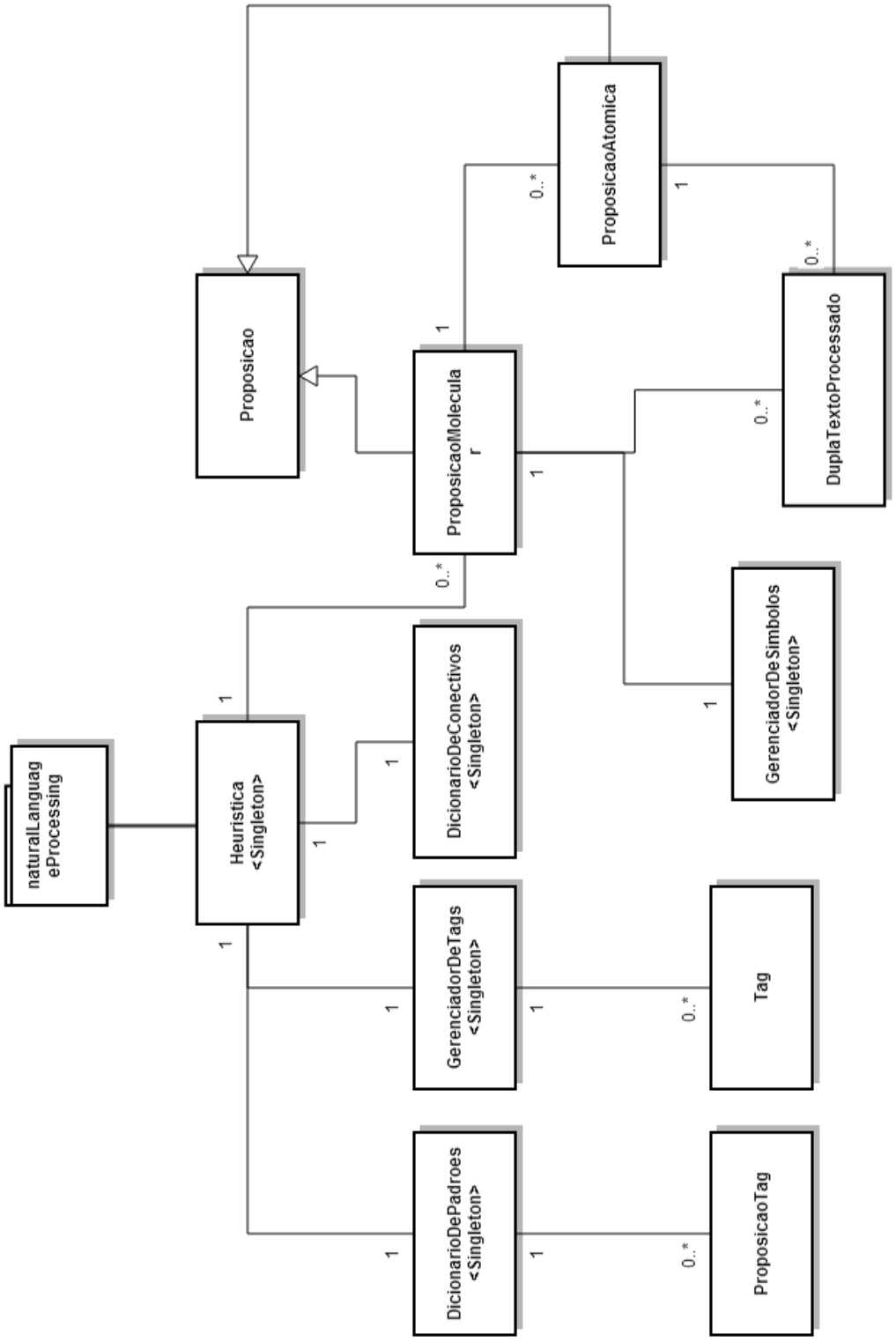

Figura 5.7: Diagrama de classes simplificado do algoritmo usado.

5.4 Definição da Entrada da Aplicação

Como foi dito no capítulo três, para a formalização acontecer a proposição deve satisfazer algumas das expressões regulares definidas no programa (no DicionarioDePadrões) para ser possível localizar a estrutura ProposiçãoTag relacionada e dela obter as expressões regulares para capturar duas proposições menores.

5.4.1 Conjunções e Disjunções Caso I

A expressão regular que captura este caso é:

“.*?V.*?KC.*?$”

Exemplo de caso capturado com esta expressão:

“Maria gosta de novela ou Jorge gosta de novela.”

5.4.2 Conjunções e Disjunções Caso II

A expressão regular que captura este caso é:

“NPROP KC NPROP .*?V .*?
.”

Exemplo de caso capturado com esta expressão:

“Maria e Jorge gostam de novela.”

5.4.3 Conjunções e Disjunções Caso III

A expressão regular que captura este caso é:

“NPROP KC NPROP .*? VAUX V .*?.”

Exemplo de caso capturado com esta expressão:

“Lucas e Matheus foram jogar bola.”

5.4.4
Conjunções e Disjunções Caso IV

A expressão regular que captura este caso é:

“VAUX V .*?,.*?KC.*?.”

Exemplo de caso capturado com esta expressão:

“Foram jogar bola, Lucas e Matheus.”

5.4.5
Implicação Caso I

A expressão regular que captura este caso é:

“KS .*? V .*? ADV .*?.”

Exemplo de caso capturado com esta expressão:

“Se Lucas foi jogar bola logo Matheus foi jogar bola.”

5.4.6
Implicação Caso II

A expressão regular que captura este caso é:

“KS\s*?KS\s*?KS\s*?ART .*? V .*? ADV .*?.”

Exemplo de caso capturado com esta expressão:

“A menos que o carro seja novo o motor nao precisa de revisao.”

5.4.7
Bi-implicação Caso I

A expressão regular que captura este caso é:

“NPROP\s*?V([a-z]|\s)*?PROPESS\s*?KC\s*?PDEN\s*?PROPESS.*?$”

Exemplo de caso capturado com esta expressão:

“A menos que o carro seja novo o motor nao precisa de revisao.”

5.5 Definição da Saida da Aplicação

Nesta seção será definida a saída da aplicação. A saída consiste em uma lista com frases consideradas proposições extraídas do texto dado como entrada, onde para cada frase são explicitadas todas as proposições atômicas que foram possíveis extrair dela. Assim como a forma lógica final que ela representa.

5.5.1 Associação de Proposições a Letras

As proposições atômicas são associadas a letras, para assim podermos repesentá-las no contexto geral do problema. Se duas proposições são ideênticas (palavras iguais), elas então possuem o mesmo sentido e, logo elas são atribuídas a um único símbolo. Um exemplo para o texto seria:

"Maria e Jorge foi ao parque. Maria foi ao parque."

Teriamos dois conjuntos na resposta:

"q $\wedge$ r"

Onde:

"q: Maria foi a o parque ; r: Jorge foi a o parque . "

E o conjunto:

"q"

Onde:

"q: Maria foi a o parque ; "

Note que ao invés da ferramenta associar esta última proposição a outra letra (no caso 's'), ela identifica que esta proposição já foi descobera anteriormente para este texto e usa a sua váriavel ao invés de criar uma nova (o que não faria sentido para a formalização deste texto).

A ordem da criação das variáveis para atribuição das proposições atômicas pode ser vista na tabela abaixo:

Cardinalidade	**Variável**
1	q
2	r
3	s
.	.
.	.
10	w
11	qq
12	qr
.	.
.	.

Tabela 5.1: Tabela mostrando ordenação na atribuição de váriaveis.

5.5.2 Diagramação dos Resultados

Os resultados são mostrados na área de texto da parte inferior do formulário inicial. Nele estão todas as frases consideradas proposições, na ordem em que aparecem na string de entrada, e para cada uma, é mostrado abaixo sua representação lógica seguida das suas proposições atômicas associadas a uma váriavel única. Para o exemplo da seção anterior, ficaríamos com:

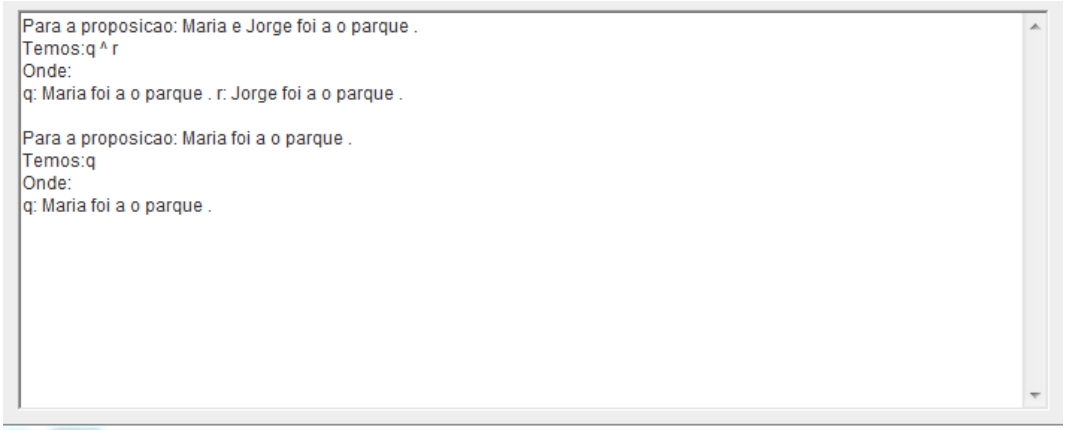

Figura 5.8: Exemplo da saída no console da aplicação.

6
Como usar a Aplicação

Este capítulo é um pequeno guia indicando como usar a ferramenta e o que esperar como resposta. Na primeira seção é encontrada a diagramação da aplicação. Na segunda seção um exemplo de formalização de uma sentença. E por fim na terceira seção é comentado sobre o problema de equivalência de palavras e como contornar alguns problemas relacionados ao processamento de linguagem natural.

6.1
Diagramação

A ferramenta possui somente um formulário e este é composto de um botão e de duas caixas de texto que são, respectivamente, de cima para baixo, a entrada e a saída da aplicação. O botão serve para executá-la.

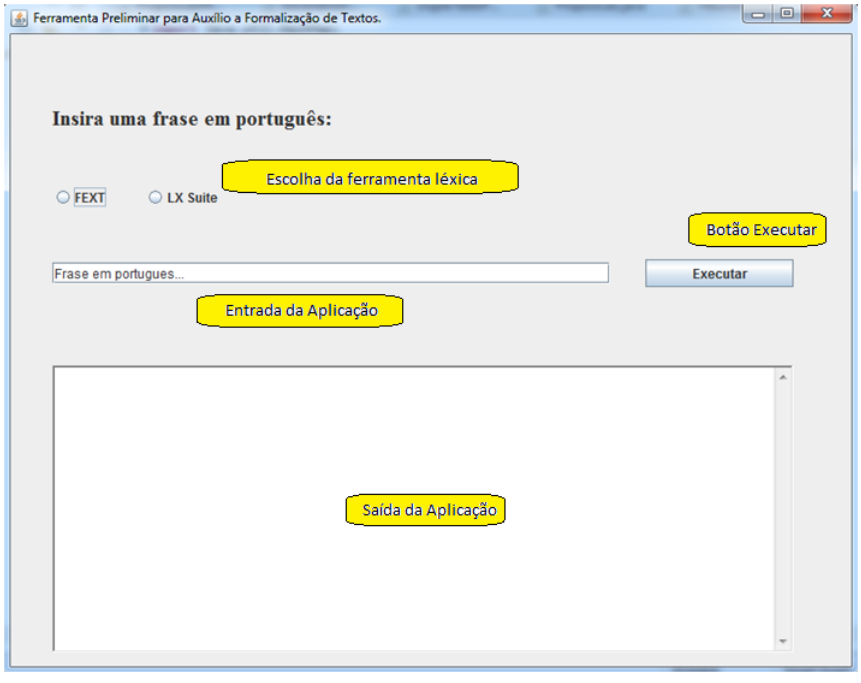

Figura 6.1: Tela completa da aplicação.

6.2 Formalização de uma sentença

A aplicação só funciona se o computador estiver conectado à internet. Para formalizar uma sentença devemos então digitá-la no campo marcado na figura anterior como Entrada da Aplicação. Em seguida basta clicar no botão Executar e esperar o resultado ser mostrado no campo inferior da imagem. O resultado é mostrado como definido na última seção do capítulo anterior.

A entrada desta aplicação deve seguir as normas do português culto e todas as sentenças devem ser finalizadas com '.'. O uso de vírgulas é desnecessário.

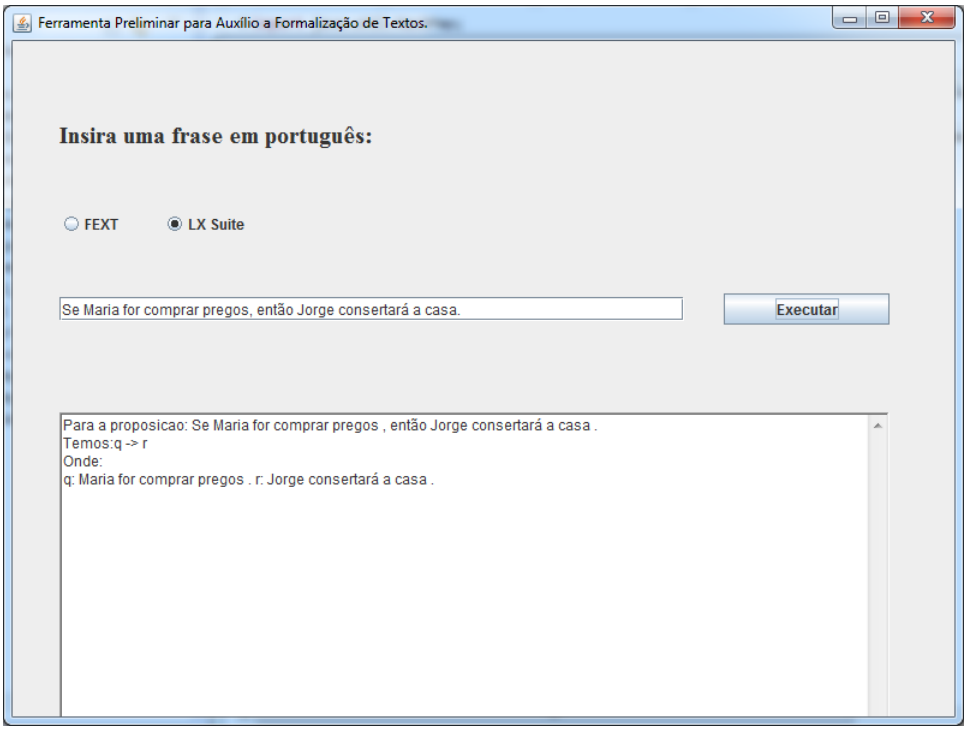

Figura 6.2: Exemplo de execução do programa.

6.3 Equivalência de palavras e Troubleshooting

Como foi descrito anteriormente, quando é usado processamento de linguagem natural, não existe a certeza absoluta de que, de fato, o texto que estamos processando foi totalmente compreendido pelo programa. No caso desta ferramenta, é comum obtermos tags errados.

Esta seção tem o objetivo de enunciar formas fáceis de tentar contornar o problema, caso o resultado da ferramenta não seja o esperado pelo usuário.

Podemos usar substituição de palavras para resolver o problema. A aplicação trabalha em cima dos tags obtidos por uma ferramenta classificadora POS, sendo assim podemos substituir uma palavra que estamos usando por

um sinônimo mais comum, facilitando o processo de associação a tags e não mudando o sentido semântico da frase. Por exemplo para a sentença:

“Se A maior que B então B é menor que A. ”

Pode ser substituído por:

“Se A maior que B logo B é menor que A. ”

O significado semântico é mantido neste exemplo, logo o mesmo resultado, com palavras diferentes. Outra forma de contornar o problema de classificação é substituindo verbos. Para nós o tempo verbal não faz diferença. Então podemos trocar todos os tempos verbais para um que tenhamos certeza que nos dará o resultado esperado. Um exemplo disso é a seguinte sentença:

“Maria é bonita. ”

Esta sentença é associada a tags de forma errada pelo F-EXT-WS. Ele não identifica que o ’é’ é o verbo ’ser’, então podemos trocar esta palavra por ’ser’ para resolver o problema. Desta forma obteremos a sentença lógica correta.

7
Conclusão e Trabalhos Futuros

Este capítulo descreve possíveis caminhos para a expansão deste trabalho por outros alunos. Onde as três primeiras seções deste capítulo são referentes a possíveis abordagens para trabalhos futuros e a última trata somente da conclusão.

1 Melhoria na PLN.

2 Interface de Assistência a Formalização.

3 Tratamento de casos compostos mais complexos.

4 Conlcusão

7.1
Melhoria na PLN

O projeto atualmente utiliza somente a classificação POS como forma de análise do texto entrado. Há outras formas de classificação que, se utilizadas em conjunto com esta, podem oferecer mais informação para realizar esta análise. Como exemplo, a própria ferramenta F-EXT-WS dispõe de mais funcionalidades. Dentre elas podemos destacar:

- Phrase chunking
- Clause identification
- Semantic role labeling.

A ferramenta F-EXT-WS possui 88% de taxa de acerto segundo o site de seus criadores, existem outras ferramentas disponíveis que tem uma porcentagem maior de acerto. Já o LX Suite desenvolvido pela universidade de Lisboa, usado neste projeto também, que possui 96% de acerto em seu POS Tagger.

7.2 Interface de Assistência

Outra forma de abordar o problema é através de diálogos com o usuário, que poderá assim colaborar com o processo de formalização, e até inserir, de forma assistida, expressões regulares para casos mais específicos, de forma que a aplicação seja capaz de "aprender"casos mais complexos. A janela abaixo pode ser um exemplo de janela de diálogo:

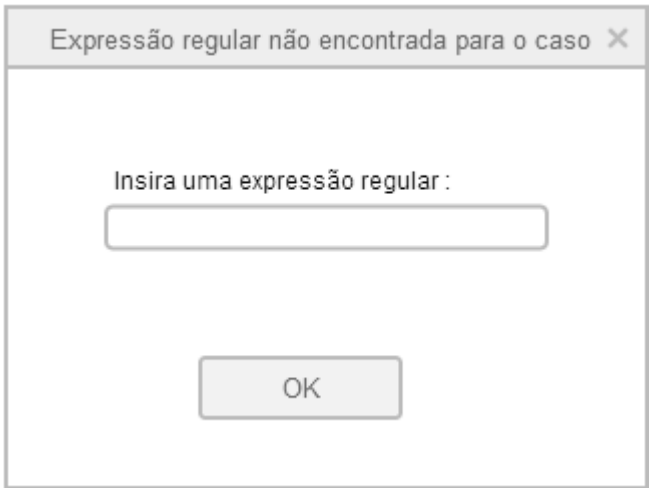

Figura 7.1: Exemplo janela de diálogo hipotética

7.3 Tratamento de casos compostos mais complexos

Há alguns casos que a ferramenta não é capaz de tratar corretamente, como quando é usado um pronome para fazer referência a um ator em outra sentença, por exemplo:

"Maria é inteligente. Por isso ela conseguiu o emprego. "

Nesta frase a ferramenta não é capaz de tratar corretamente no estado que se encontra, pois não consegue identificar e relacionar o pronome pessoal 'ela' à Maria. Nesses casos o usuário deve re-escrever a frase como:

"Maria é inteligente logo Maria conseguiu o emprego. "

Um projeto futuro poderia abordar focar o tratamento deste tipo de situação, criando uma ferramenta mais inteligente e funcional.

7.4 Conclusão

Devido ao trabalho com as incertezas geradas pelo processamento de linguagem natural e pela complexidade de unir a lingua portuguesa à linguagem lógica. Esta a solução proposta, embora simples, foi um primeiro passo adotado para compreeender e propor uma solução para este problema.

Além de servir como proposta de solução, este projeto tinha como objetivo criar uma plataforma documentada que pudesse ser usada como degrau para outros projetos. Por este motivo segue em anexo ao trabalho a documentação em JavaDoc do código do trabalho assim como o endereço para o seu repositório online onde outros desenvolvedores podem aproveitar o trabalho feito como base.

Referências Bibliográficas

[Correia] CORRÊIA, M. DA S.; TRALES, P. R. B. C. R. C. M.. **Lógica para Computação**. Thomson Learning, 2006. 1.2.2

[Deitel] DEITEL, P. J.; DEITEL, H. M.. **Ajax, Rich Internet Applications and Web development for programmers**. Pearson, 2008. 1.3.1

[EtiquetasPOSLXSuite] **LX Suite, etiquetas do lx suite**. http://lxcenter.di.fc.ul.pt/services/en/LXServicesSuite.html#tags. Acessado: 3-07-2014. 1.4.1

[FEXTWSDoc] **F—EXT–WSDoc, site da ferramenta**. http://www.learn.inf.puc-rio.br/index.php/F-EXT-WSHelp. Acessado: 4-07-2014. 1.3

[Flowcharts] **Flowcharts, referência sobre flowcharts**. http://en.wikipedia.org/wiki/Flowchart. Acessado: 4-07-2014. 2.3

[Gliffy] **Gliffy, site usado para desenhar fluxogramas**. http://www.gliffy.com/. Acessado: 4-07-2014. 2.3

[Google Code] **Google Code, repositório do projeto**. https://code.google.com/p/ferramenta-auxiliar-para-formalizacao-de-sentencas/. Acessado: 4-07-2014. 2.1, 2.4.3

[Grupo NLX] **Grupo NLX, site do grupo nlx**. http://nlxgroup.di.fc.ul.pt/. Acessado: 4-07-2014. 1.4

[Javadoc] **Javadoc, referência sobre javadoc**. http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html. Acessado: 4-07-2014. 2.4.3

[LXSuite] **LX Suite, projeto lx suite**. http://lxcenter.di.fc.ul.pt/services/en/LXServicesSuite.html. Acessado: 4-07-2014. 2.1

[LacioWeb] **Lácio Web, website do grupo**. http://www.nilc.icmc.usp.br/lacioweb/descricao.htm. Acessado: 4-07-2014. 1.3.1

[LoLITA] **LoLITA, projeto lolita**. `http://lolita.dimap.ufrn.br/llt/`. Acessado: 4-07-2014. 2.2

[Metacaractere] **Metacaractere, artigo sobre metacaracteres**. `http://en.wikipedia.org/wiki/Metacharacter`. Acessado: 4-07-2014. 3.1

[NFA] **NFA, artigo sobre automatos finitos**. `http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton`. Acessado: 4-07-2014. 3.1

[Remote Method Invocation] **Java RMI, referência sobre rmi**. `http://en.wikipedia.org/wiki/Java_remote_method_invocation`. Acessado: 4-07-2014. 4.1

[Singleton] **Singleton, referência sobre singleton**. `http://en.wikipedia.org/wiki/Singleton_pattern`. Acessado: 4-07-2014. 5.1.1

[Van Dalen] DALEN, D. V.. **Logic and Structure**. Universitext, Springer, 2003. Informacoes sobre logica proposicional. 1.2