INSTITUTO DE MATEMÁTICA E ESTATÍSTICA - USP

MAC0499 - Trabalho de Formatura Supervisionado

CATA: Collaborative Academic Text Advisor

Um sistema de verificação de estilo para textos acadêmicos de Computação

Ana Luiza Domingues Fernandez Basalo

Orientação: Professor Marco Aurélio Gerosa

1 de dezembro de 2011

Resumo

Linguagens formais - como as linguagens de programação - são desenvolvidas artificialmente e possuem uma sintaxe bem definida. Línguas humanas, por sua vez, surgem naturalmente como instrumento de comunicação e estão sujeitas a ambiguidades e múltiplas interpretações. O ramo da Computação que lida com esse último tipo de linguagem é chamado de Processamento de Linguagens Naturais (PLN). Entre os problemas comuns de PLN estão extração e recuperação de informação, assim como correção, resumo e tradução de textos. Há, atualmente, muitos programas que usam algoritmos de PLN para efetuar diversos tipos de correção de textos (ortográfica, gramatical e de estilo), mas nenhum que realize correção de estilo para a área de Computação especificamente.

Assim sendo, neste trabalho, foi desenvolvido um sistema de verificação de estilo para textos acadêmicos de Computação. Tal sistema - denominado "CATA" - analisa textos levando em conta aspectos linguísticos e estéticos, para, por exemplo, detectar ocorrências de traduções incorretas e estrangeirismos: como o uso de "testes unitários" como tradução para "unit tests", em vez da forma correta "testes de unidade"; ou ainda, "a função retorna um determinado valor" (tradução de "return"), quando mais elegante seria "a função devolve um determinado valor".

Ademais, o sistema implementado é colaborativo, fazendo uso da Inteligência Coletiva, cujo propósito é combinar o conhecimento de várias pessoas para criar soluções e ferramentas mais poderosas. Mais especificamente, a partir de informações fornecidas voluntariamente pelos usuários, o software desenvolvido neste trabalho aperfeiçoa sua avaliação. Com esse objetivo, foram estudados técnicas probabilísticas e algoritmos de Aprendizagem Computacional.

Sumário

1 Introdução

A ideia de que um dia computadores irão "entender" textos em linguagens naturais ainda está muito longe da realidade. É fato que foram realizados consideráveis avanços na área de Processamento de Linguagens Naturais: já existem soluções bastante razoáveis para resumo e tradução, por exemplo, mas, em muitos casos, tais soluções não estão num nível comparável ao desempenho humano. Com correção de textos não é diferente: em particular, correção de estilo de textos em línguas naturais é um processo subjetivo e complexo até para seres humanos. Mesmo assim, restringindo um pouco a definição de *estilo* e introduzindo técnicas de uso da Inteligência Coletiva, foi possível desenvolver um *software* para semiautomatizar esse processo.

1.1 Motivação

Escrever textos acadêmicos é uma tarefa árdua para muitos dos alunos de Computação: além dos desafios da escrita em si - com os quais os alunos não foram habituados a lidar durante sua formação - há questões de terminologia, anglicanismo e tradução inerentes à área. Disso decorrem diversos problemas de escrita, que levam orientadores de trabalhos de conclusão de graduação, mestrado e doutorado e professores a despender um longo tempo com correção de textos. Muitos dos problemas de escrita não são detectados e a disseminação dos resultados das pesquisas desenvolvidas é prejudicada. Editores de textos oferecem corretores ortográficos e, em alguns casos, gramaticais e de estilo. Porém, não são encontrados corretores específicos para estilos e termos da área de Computação. Com esta motivação, foi desenvolvido o Sistema *CATA: Collaborative Academic Text Advisor*, cujo principal objetivo é ser capaz de detectar problemas de estilo em textos acadêmicos de Computação (tanto em português como em inglês), bem como sugerir possíveis correções para tais problemas.

1.2 Questões pesquisadas

Seguindo o princípio de "divisão e conquista"[1], para responder à pergunta fundamental deste trabalho - *"Como realizar verificação de estilo de textos acadêmicos de Computação?"* - foram definidos e pesquisados problemas "menores", esboçados a seguir.

1. *O que é corrigir o estilo de textos em linguagens naturais? O que isso significa para textos acadêmicos da área de Computação?*

   Naturalmente, o ponto de partida do trabalho foi buscar o que entende-se por *estilo*, tanto na questão artística e estética, como no que se refere a expressividade e clareza. A seguir, entender como isso se aplica a textos técnicos de Computação - quais são os problemas de escrita mais comuns desses tipos de trabalhos ou quais são mais relevantes de serem resolvidos.

2. *Como definir o problema da verificação de estilo para que possa ser resolvido algoritmicamente?*

   A questão da verificação de estilo é bastante ampla e tem certo caráter subjetivo e artístico, de modo que não pode ser solucionada totalmente por sistemas computacionais. Portanto, era necessário definir com mais objetividade o problema, de maneira que pudesse ser resolvido por um *software*.

[1] **Divisão e conquista** (do inglês *Divide and conquer*) é um paradigma de projeto de algoritmos no qual um determinado problema é dividido recursivamente em problemas menores, suficientemente simples para que possam ser resolvidos diretamente. A seguir, as soluções desses subproblemas são então combinadas para gerar a resposta do problema original.

3. *A partir da definição da questão anterior, como processar os textos para detectar os problemas de estilo?*

   Definido então o escopo do trabalho, o próximo passo era formalizar soluções para analisar textos e encontrar problemas de estilo.

4. *Como o sistema poderia melhorar sua avaliação ao longo do tempo a partir do comportamento dos usuários?*

   Como já mencionado, verificação de estilo é algo subjetivo e complexo. Assim sendo, o sistema desenvolvido poderia ter melhor eficácia e ser melhor aproveitado se os usuários pudessem contribuir de maneira relevante - por exemplo, usando seu próprio conceito de estilo e reportando erros ou inconsistências na análise dos textos realizada pelo sistema.

1.3 Objetivos

A partir dos questionamentos levantados na seção anterior, podemos sintetizar os objetivos deste trabalho:

Produzir um *software* que seja capaz de verificar o estilo de textos acadêmicos da área de Computação, aperfeiçoando a qualidade de suas análises a partir de informações fornecidas voluntariamente pelos usuários.

1.4 Estrutura do presente trabalho

Esta monografia está organizada segundo a seguinte estrutura:

- *Seção 1*, **Introdução**
  Esta seção tem por objetivo apresentar o trabalho ao leitor: provê uma visão geral dos objetivos e temas pesquisados.

- *Seção 2*, **Contextualização**
  Descreve, sucintamente, as áreas de conhecimento que forneceram a base teórica sobre a qual este trabalho se sustenta.

- *Seção 3*, **Trabalhos relacionados**
  Apresenta, brevemente, alguns trabalhos relacionados - de corretores de estilo existentes a trabalhos em Processamento de Linguagens Naturais em geral.

- *Seção 4*, **Fundamentação teórica**
  Neste trecho são apresentadas definições formais dos problemas resolvidos (e possíveis discussões filosóficas), bem como as soluções e algoritmos estudados para resolvê-los, incluindo análises sobre suas complexidades.

- *Seção 5*, **Arquitetura e implementação**
  Apresenta, com detalhes, a arquitetura, as tecnologias e as ferramentas utilizadas na implementação do CATA.

- *Seção 6*, **Resultados**
  Expõe os métodos e testes usados para avaliar, em vários aspectos, o sistema desenvolvido.

- *Seção 7*, **Conclusão**
  Relaciona os resultados obtidos com as questões e motivações do trabalho, fornecendo um "encerramento" a este texto.

- *Seção 8*, **Anexos**
  Foram incluídos, nesta seção, mais detalhes sobre a implementação, os testes e seus resultados.

- *Seção 9*, **Parte subjetiva**
  Aqui é feito um paralelo entre o Trabalho de Formatura Supervisionado e o Curso de Bacharelado em Ciência da Computação (quais disciplinas foram mais relevantes no desenvolvimento do trabalho), entre outras observações mais subjetivas sobre desafios e frustrações encontrados na elaboração do trabalho.

- **Referências**
  É a "Bibliografia" - lista todas as referências usadas na produção deste texto, com breves comentários sobre o que foi lido.

2 Contextualização

Esta monografia e o *software* no qual se baseia, embora sejam um trabalho primeiramente de Computação, têm um caráter multidisciplinar. Nas seções subsequentes são apresentadas, em linhas gerais, as áreas de conhecimento nas quais este trabalho se insere.

2.1 Processamento de Linguagens Naturais (PLN)

Processamento de Linguagens Naturais (PLN, do inglês *Natural Language Processing*) é um campo da Inteligência Artificial que ocupa-se de línguas humanas naturais.

> Em Filosofia da Linguagem e Linguística, uma **linguagem natural** é qualquer língua que surge de uma forma não premeditada, como resultado da capacidade inata que o intelecto humano possui para linguagens, visando sobretudo a comunicação. Desta forma, linguagens naturais distinguem-se daquelas chamadas de artificiais (como as linguagens de programação e aquelas usadas em lógica formal e matemática), as quais são construídas e definidas formalmente.[WK2]

O objetivo do PLN é desenvolver teorias e técnicas para analisar, entender e gerar textos em línguas que humanos usam naturalmente, de modo que, finalmente, um usuário possa dirigir-se ao seu computador como se estivesse dirigindo-se a outra pessoa. Esta meta não é simples de ser alcançada. "Entender" uma linguagem significa, entre outras coisas, saber o significado das palavras e frases dentro de um contexto (lidando com ambiguidades e polissemia), compreender referências a conceitos e fatos que não estão expressos nos textos (contexto externo), assim como discernir variações e dialetos - como regionalismos. Assim sendo, há muitos desafios na manipulação de linguagens naturais por computadores.

2.1.1 Histórico

Historicamente, aceita-se que o Processamento de Linguagens Naturais surgiu no início da década de 1950, embora possam ser encontrados trabalhos anteriores a essa data. Em 1950, Alan Turing (1912-1954) publicou seu artigo *"Computing Machinery and Intelligence"*, no qual propõe o "Teste de Turing" como critério de inteligência. Tal critério decide a inteligência de um programa de computador avaliando sua capacidade de "imitar" um humano numa conversa escrita em tempo real. Nesses primórdios do PLN, o principal foco de pesquisa era tradução de textos: no contexto da Guerra Fria, tanto os Estados Unidos como a União Soviética desejavam traduzir rapidamente os textos científicos de seu oponente.

Já nos anos 1960, ocorreu o desenvolvimento do famoso "ELIZA"[2], um *software* que simulava um psicólogo que "conversava" com o usuário, respondendo e reagindo a perguntas e frases digitadas. O programa baseava-se num algoritmo relativamente simples de casamentos de padrões, mas acabou levando a discussões sobre a relevância de decisões baseadas em resultados fornecidos por computadores[3]. Os anos 1980, por sua vez, são considerados um marco para a área: nesta época, foram introduzidas soluções de Aprendizagem de Máquina para processar línguas naturais, substituindo a antiga técnica - regras explícitas de antemão no código. Recentemente (mais

[2] Uma implementação do ELIZA pode ser encontrada no popular editor de textos Emacs - o "Emacs Psychotherapist".

[3] Joseph Weizenbaum, escritor e acadêmico da área de Computação, desenvolveu o ELIZA e, motivado pelas reações e repercussão do *software*, publicou o *"Computer Power and Human Reason: From Judgement To Calculation"*, livro no qual diferencia "decidir" de "escolher", afirmando que o primeiro seria computacionalmente possível, mas o segundo não, por envolver, mais do que cálculos, julgamento.

precisamente, neste ano de 2011), veio à mídia o "Watson" - sistema desenvolvido pela IBM -, que foi capaz de vencer humanos numa competição de perguntas e respostas (em inglês).

Atualmente, o interesse em Processamento de Linguagens Naturais é grande, devido não apenas ao fato de muitos dos problemas da área ainda não terem soluções satisfatórias, mas também à grande quantidade de conteúdo codificado em linguagens naturais disponível na Internet.

2.1.2 Aplicações

Os resultados e algoritmos de PLN encontram aplicação em diversas tarefas, como tradução, resumo e correção de textos, extração e recuperação de informação e interpretação de conteúdo.

2.2 Inteligência Coletiva

O que os sistemas de recomendação do *Last.fm* e da *Amazon*, o motor de buscas *Google* e a enciclopédia *Wikipedia* têm em comum?

Last.fm - uma mistura de rede social com sistema de recomendação musical - sugere músicas e artistas a seus usuários de acordo com o histórico de músicas ouvidas. *Amazon.com* é uma empresa de comércio eletrônico: quando alguém efetua uma compra ou mesmo visita a página de um produto, são exibidas recomendações sobre outros produtos que poderiam ser de interesse para o usuário. *Google* dispensa apresentações: é o mais usado motor de buscas da atualidade. Seu funcionamento baseia-se no algoritmo *PageRank*, que, grosso modo, classifica os resultados das buscas levando em conta a quantidade e a relevância das páginas que possuem *links* para eles. *Wikipedia*, por sua vez, é uma enciclopédia na Internet e seu conteúdo é construído, voluntariamente, por usuários do mundo inteiro.

O que os sistemas citados têm em comum é que todos fazem uso da chamada Inteligência Coletiva. Basicamente, Inteligência Coletiva é "uma combinação de comportamentos, preferências ou concepções de um grupo de pessoas que podem ser usadas para criar novas ideias"[IC]. Em outras palavras, significa usar o conhecimento e a iniciativa de várias pessoas para produzir ferramentas mais poderosas. Esse é o caso dos sistemas de recomendação da *Amazon* - dados de compras realizadas anteriormente alimentam o mecanismo de sugestões de produtos - e do *Last.fm* - informações providas pelos usuários bem como o comportamento dos mesmos possibilitam que o sistema deduza que artistas e músicas pertencem a um mesmo gênero musical. Também é o caso da *Wikipedia* - exemplo no qual o conceito de Inteligência Coletiva fica mais evidente - e do *Google* - pensando abstratamente, o algoritmo *PageRank* usa a opinião de várias pessoas acerca de uma página para classificá-la dentre os resultados de uma busca.

É preciso ressaltar, entretanto, que, embora os exemplos aqui expostos sejam de sistemas computacionais, a Inteligência Coletiva já era possível mesmo antes do surgimento dos computadores: a Era Digital simplesmente tornou mais fácil coletar informações de milhões de pessoas e, por isso, motivou a discussão acerca do conceito. O termo Inteligência Coletiva foi introduzido em 1994 pelo filósofo da informação Pierre Lévy (1956).

> *"O que é Inteligência Coletiva? É uma inteligência distribuída por toda a parte, incessantemente valorizada, coordenada em tempo real, que resulta em mobilização efetiva das competências. Acrescentemos à nossa definição este complemento indispensável: a base e o objetivo da inteligência coletiva são o reconhecimento e o enriquecimento mútuo das pessoas, senão o culto de comunidades fetichizadas ou hipostasiadas. Uma inteligência distribuída por toda parte: tal é o nosso axioma inicial. Ninguém sabe tudo, todos sabem alguma coisa, todo o saber está na humanidade."[WP3]*
>
> Pierre Lévy in
> *A inteligência coletiva: por uma antropologia do ciberespaço*

2.3 Aprendizagem de Máquina

Aprendizagem de Máquina (ou Aprendizagem Computacional) - também uma subárea da Inteligência Artificial - dedica-se a algoritmos que possibilitam que computadores "aprendam". O termo "aprender" aqui tem um sentido bastante específico: para um computador, "aprender" significa inferir padrões e generalizações a partir de um conjunto de dados e, com base nessas generalizações, fazer previsões sobre dados futuros e desconhecidos[IC]. Trata-se, portanto, de um processo indutivo. Proceder desta maneira leva a conclusões razoáveis para dados que não sejam aleatórios, pois esses tipos de dados possuem algum tipo de padrão que pode ser detectado.

Existem diversas técnicas de Aprendizagem de Máquina, classificadas nos chamados paradigmas de aprendizado. Dentre esses modelos de aprendizado distinguem-se o Aprendizado Supervisionado (ao algoritmo são providos conjuntos de dados e a resposta esperada para cada um) e o Aprendizado Não-Supervisionado (não é fornecida a saída esperada para o conjunto de dados de entrada), entre outros.

Alguns usos para as técnicas de Aprendizagem de Máquina são detecção de fraudes financeiras, análise de mercado de ações, tarefas em Bioinformática e Processamento de Linguagens Naturais ou qualquer campo onde haja uma tal quantidade de informação impossível de ser gerenciada totalmente por um ser humano.

3 Trabalhos relacionados

O interesse principal deste trabalho é a verificação de estilo de textos - mais especificamente, de textos acadêmicos de Computação. Existem vários programas que realizam correção ortográfica, gramatical e de estilo para textos, mas nenhum que efetue correção de estilo para a área de Computação em particular.

- **WhiteSmoke:** Trata-se de um *software* cujo propósito é melhorar a qualidade da escrita de textos em Inglês, oferecendo vários tipos de correção (incluindo de estilo).
- **Grammarly:** Também com foco na língua inglesa, provê vários mecanismos para correção ortográfica, gramatical e de estilo dos textos e, além disso, ainda consegue detectar plágios.
- **Microsoft Word:** Trata-se de um processador de textos, mas que possui ferramentas para correção deles (em várias línguas), oferecendo também sugestões de estilo.

Os sistemas de *software* listados são todos proprietários e não é possível obter acesso aos seus códigos-fonte. Assim, outras ferramentas de correção foram buscadas: existem diversas ferramentas livres para correção de textos, como o LanguageTool (que provê correção gramatical e de estilo para vários idiomas - Inglês, Espanhol e Francês, entre outros). Em particular, para a língua portuguesa, existe o CoGrOO - um corretor gramatical.

3.1 CoGrOO

O CoGrOO é um corretor gramatical livre para o Português, acoplável ao LibreOffice. Eis alguns dos erros gramaticais que o CoGrOO é capaz de detectar[CG]:

- colocação pronominal
- concordância nominal
- concordância entre sujeito e verbo
- concordância verbal
- uso de crase
- regência nominal
- regência verbal

O CoGrOO foi implementado segundo uma arquitetura modular - existem três partes principais nas quais o sistema foi divido: a *interface* - responsável por interagir com o LibreOffice, o *núcleo* - que efetua o processamento dos textos, e a camada de *dados* - que engloba os modelos construídos a partir do *corpus*[4] CETENFolha[CF].

O *núcleo* do CoGrOO, por sua vez, também é formado por uma série de módulos, que irão realizar uma análise do texto para detectar os erros gramaticais. Entre esses módulos estão o Separador de Sentenças e o "Toquenizador", responsáveis por separar o texto em sentenças e em *tokens*[5]. Em seguida, é realizada a etiquetagem morfológica[6] dos itens lexicais obtidos na fase anterior. Após passar pelo Etiquetador Morfológico, o texto é submetido ao Detector de Sintagmas - que agrupa sintagmas nominais e verbais, e ao Detector de Relações Gramaticais - que determina a função

[4] O termo *corpus* está definido em 4.3.1.1.
[5] O processo de Segmentação - que produz os *tokens* - é descrito em 4.2.
[6] A Etiquetagem Morfológica é apresentada com mais detalhes em 4.3.

sintática dos elementos do texto (sujeito, predicado, etc.). Finalmente, o texto é então sujeito às regras gramaticais disponíveis no sistema. É importante salientar que esta estrutura modular da análise/correção dos textos permite que os componentes sejam substituídos: podemos usar uma ferramenta diferente da existente para realizar Etiquetagem Morfológica, por exemplo.

Embora o CoGrOO não realize correção de estilo, seu estudo foi relevante para este trabalho, pois, como apresentado, o CoGrOO possui diversas ferramentas úteis de processamento de linguagens naturais para analisar textos em língua portuguesa.

3.2 Outros trabalhos

Como o CATA lida com várias áreas da Computação, vários sistemas computacionais poderiam ser citados aqui. Aqueles que foram usados diretamente na implementação do Sistema CATA serão apresentados na Seção de *Arquitetura e implementação* (5).

4 Fundamentação teórica

4.1 Estilo

Afinal o que é *estilo* de um texto?

Para os linguistas, *estilo* é o conjunto de características que definem a "personalidade" de um texto. Tais características expressas na escrita revelam tanto os atributos individuais e idiossincrasias do autor como a forma como ele vê e interpreta sua audiência. Assim, os *estilos* de vários textos de um mesmo autor podem ser diferentes (de acordo com o propósito e a situação), mas, ao mesmo tempo, é possível falar em um estilo individual desse autor - uma marca própria que ele deixa em todos os seus textos.

> "*Agora Fabiano conseguia arranjar as idéias. O que o segurava era a família. Vivia preso como um novilho amarrado ao mourão, suportando ferro quente. Se não fosse isso, um soldado amarelo não lhe pisava o pé não. O que lhe amolecia o corpo era a lembrança da mulher e dos filhos. Sem aqueles cambões pesados, não envergaria o espinhaço não, sairia dali como onça e faria uma asneira. Carregaria a espingarda e daria um tiro de pé de pau no soldado amarelo. Não. O soldado amarelo era um infeliz que nem merecia um tabefe com as costas da mão. Mataria os donos dele. Entraria num bando de cangaceiros e faria estrago nos homens que dirigiam o soldado amarelo. Não ficaria um para semente. Era a idéia que lhe fervia na cabeça. Mas havia a mulher, havia os meninos, havia a cachorrinha.*" [VS]
>
> Graciliano Ramos in
> *Vidas Secas*

O trecho acima, de uma obra capital do escritor brasileiro Graciliano Ramos (1892-1953), apresenta um estilo seco e objetivo, dominado por períodos curtos e poucos adjetivos. Tal estilo áspero concorda com o tema da narrativa - a desumanização provocada pela miséria e pela seca.

O *estilo* de um texto é, portanto, resultado das várias escolhas linguísticas feitas pelo autor: escolha lexical, do tom, da organização da linguagem para transmitir uma informação ou traduzir uma visão. Em particular, para textos científicos e acadêmicos, essas escolhas devem ser feitas de modo a produzir clareza e objetividade. Assim, termos ou figuras de linguagem que poderiam trazer mais expressividade a outros tipos de textos, para textos acadêmicos são considerados impróprios e problemáticos. Alguns desses problemas são: alteração da ordem sintática natural, repetição excessiva das mesmas palavras ao longo do texto, uso de linguagem coloquial e informal, entre outros. Vejamos mais alguns exemplos[DU]:

- **Uso indevido de substantivos para representar ações**

  ⚠ *Nós realizamos a **análise** dos dados.*

  ✔ *Nós **analisamos** os dados.*

  O uso de substantivos em vez de verbos para representar ações não pode ser considerado, em geral, um problema. Contudo, certas vezes, tal procedimento pode resultar em períodos obscuros e deselegantes: embora a primeira frase não possa ser considerada errada, a segunda é mais direta e concisa.

- **Escolha inadequada de sujeito**

  ⚠ *The **movement in the liquid medium** of the bacteria was accomplished by microflagella.*

  ✔ *The **bacteria** move themselves in the liquid medium with microflagella.*

  Na primeira sentença, o sujeito é um substantivo abstrato ("*movement*"), enquanto na segunda, um substantivo concreto ("*bacteria*"). Esta última é bem mais clara, pois coloca em evidência o principal agente da oração.

4.1.1 Estilo em textos acadêmicos de Computação

Além dos problemas de estilo citados anteriormente, existe um que é particularmente recorrente em textos acadêmicos de Computação em língua portuguesa: o emprego inadequado da terminologia da área. É fato que escrever textos de Computação em uma língua diferente do Inglês não é fácil: frequentemente, os alunos não sabem como traduzir determinadas expressões ou acreditam que é mais correto usar um termo em Inglês do que traduzi-lo para o Português. Assim, é comum encontrarmos nesses textos construções como "testes unitários" ("testes de unidade" é mais adequado como tradução para "*unit tests*"), "debugar" (a tradução apropriada para "*to debug*" é "depurar"), "rodar o programa" (quando melhor seria "executar o programa") ou "*browser*" (no lugar de "navegador").

Quando o assunto é textos científicos de Computação em Inglês, embora a questão da terminologia não seja um problema, existem outros pontos delicados, sobretudo quando os textos são escritos por alunos cuja língua materna não é o Inglês. Estes alunos muitas vezes não conseguem realizar as escolhas linguísticas adequadas para comunicar exatamente o que pretendem, ou acabam fazendo uso de uma linguagem pouco formal. Alguns exemplos de erros de estilo para esses textos são escrever "*On one hand, [...]. On the other hand, [...]*" (quando o correto é "*On **the** one hand, [...]. On the other hand, [...]*") ou usar contrações (e.g. "*it's*" em vez de "*it is*").

4.1.2 Estilo em textos acadêmicos de Computação para este trabalho

Por tudo o que foi abordado nesta seção, é possível concluir que a questão do *estilo* é bastante abrangente e complexa. Para que fosse viável construir um sistema computacional para auxiliar professores e alunos com o estilo de textos acadêmicos de Computação, foi preciso restringir o que é considerado problema.

Para este trabalho então foi adotada a seguinte definição:

> **Problema de estilo:** Uso de palavras, termos ou expressões considerados incorretos ou deselegantes. Do ponto de vista computacional, esta definição engloba os problemas de estilo que podem ser detectados por uma busca de padrões.

Esta definição abrange, portanto, estrangeirismos, traduções inadequadas, clichês, pleonasmos, contrações (no caso de textos em Inglês), entre outros.

No *software* implementado, uma sugestão de estilo é chamada de "regra". Por exemplo, o par "core" (estrangeirismo, termo inadequado)/ "núcleo" (alternativa para substituir o termo inadequado) é uma regra do sistema. Existem dois tipos de regra no CATA: as regras padrões - que já estão cadastradas no sistema inicialmente - e as demais regras - as quais são cadastradas pelos usuários. A lista das regras padrões do Sistema CATA foi incluída no *Anexo A* (8.1) - elas foram retiradas de [FK1], [FK2] e [DU], as principais referências que motivaram e guiaram este trabalho.

4.2 Segmentação (*Tokenization*)

Em geral, quando processa-se textos em linguagens naturais, segue-se um determinado fluxo de processamento. Uma das primeiras etapas desse fluxo é a chamada Segmentação (ou *Tokenization*), que consiste em "quebrar" o texto em suas unidades fundamentais, os chamados *tokens*. O conceito do que é um *token* depende do domínio de aplicação: podem ser conjuntos de palavras, palavras ou mesmo caracteres. Para este trabalho, *tokens* são palavras ou sinais de pontuação.

Ademais, para certas situações, a Segmentação também remove alguns *tokens*. É o caso da determinação de palavras-chave de textos: são removidos os *tokens* que não são candidatos a palavras-chave, como pontuação, por exemplo. Para este trabalho, no entanto, isto não ocorre: a Segmentação é realizada para obter a lista de todos os *tokens* do texto original, com o propósito de que a busca de problemas de estilo seja realmente uma busca de termos inadequados, i.e., para que não haja casamentos com partes dos termos do texto. Além disso, a etapa de Segmentação é fundamental para o processamento posterior [7], o qual recebe uma lista de *tokens* como entrada.

A Segmentação pode parecer, a princípio, extremamente simples, mas, na verdade, exige alguns cuidados. É preciso diferenciar, por exemplo, um ponto (".") que termina uma oração de um ponto de uma abreviatura (como em "Sr."). Para resolver este problema, usa-se um dicionário de abreviaturas.

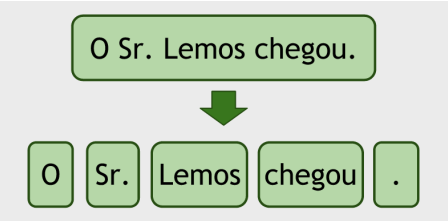


Figura 4.1: Segmentação - Exemplo

4.3 Etiquetagem Morfológica e Lematização

Ao processarmos um texto em linguagem natural, podemos estar interessados em saber a classificação de cada termo e de qual palavra ele derivou. Por exemplo, considere a palavra "meninas": sabemos que trata-se de um substantivo feminino e plural. Sabemos também que "meninas" origina-se de "menino". A primeira informação diz respeito à Etiquetagem Morfológica e a segunda, à Lematização ("menino" é o lema de "meninas").

Etiquetagem Morfológica: Em Inglês *POS-Tagging* (*Part-Of-Speech Tagging*), é um processo que consiste em designar uma etiqueta morfológica a cada palavra de um texto, de acordo com seu significado e com o contexto no qual está inserida. As etiquetas que podem ser conferidas às palavras variam de conjuntos bem simples ("substantivo", "adjetivo", "verbo", etc.) a conjuntos mais sofisticados, que incluem subclassificações como gênero (no caso de substantivos, adjetivos, etc.), tempo (no caso de verbos), etc.

Lematização: É a determinação do lema de cada palavra de um texto. O lema de uma palavra é sua forma não flexionada: informalmente, é a palavra na forma como aparece no dicionário. Por exemplo, em Português, o lema de "ando" é "andar". A palavra em Inglês "better" tem como lema "good".

[7] Este processamento posterior é a etapa de Etiquetagem Morfológica e Lematização. Mais detalhes sobre a análise dos textos realizada pelo Sistema CATA na Seção de *Arquitetura e implementação* (5).

Esses dois processos (Etiquetagem Morfológica e Lematização) estão intimamente ligados: é necessário saber a classificação de uma determinada palavra para podermos obter seu lema, e, mais ainda, é preciso que a palavra esteja inserida num contexto para que sua Lematização seja efetiva. Considere, por exemplo, a palavra "como". Alguém mais apressado poderia dizer que seu lema é "comer", mas isso não é sempre verdade. Na seguinte frase, "O aluno fez como o professor pediu.", a palavra "como" não é flexão do verbo "comer".

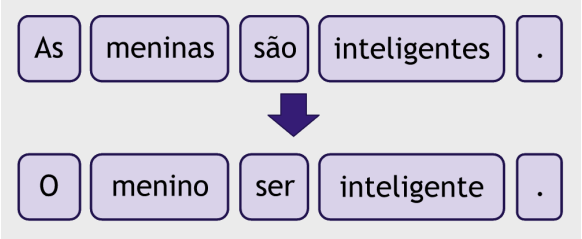


Figura 4.2: Lematização - Exemplo (Português)

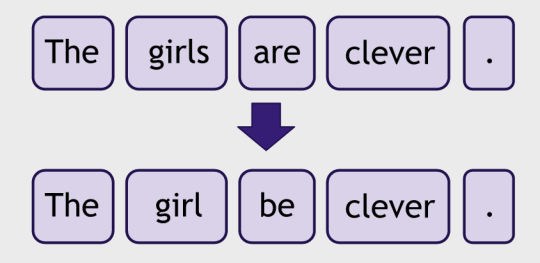


Figura 4.3: Lematização - Exemplo (Inglês)

Para este trabalho, é necessário aplicar os procedimentos de Etiquetagem Morfológica e Lematização aos textos fornecidos como entrada ao sistema. Vamos explicar o porquê por meio de um exemplo. Considere o verbo - "inexistente" na língua portuguesa - "debugar". Se tal verbo ocorresse num texto acadêmico de Computação, o sistema deveria apontá-lo como problema de estilo (a sugestão seria trocar por "depurar"). Mas essa regra também deveria funcionar para as seguintes flexões do verbo: "debugo", "debuguei", "debugamos", etc. Seria cansativo para um usuário ter que inserir todas as flexões de uma palavra ao cadastrar uma nova regra e até mesmo ineficiente para o sistema ter que armazenar e buscar todas estas flexões, sendo que, o que se quer, na verdade, é saber se uma determinada palavra é ou não flexão de uma outra.

Portanto, dado um texto acadêmico de Computação, o sistema realiza a Segmentação e, em seguida, aplica técnicas de Lematização e Etiquetagem Morfológica (conforme o idioma), obtendo, assim, um texto "lematizado". Problemas de estilo em suas formas não flexionadas poderão então ser buscados nesse texto "lematizado" (mais detalhes sobre o fluxo de processamento do Sistema CATA na Seção de *Arquitetura e implementação* (5)).

Por tudo que foi exposto, fica claro que tanto a Etiquetagem Morfológica como a Lematização são métodos que dependem do idioma. Assim, foram usadas soluções distintas para Português e Inglês.

4.3.1 Português

A princípio, a intenção era usar o CoGrOO (3.1) para realizar Lematização e Etiquetagem Morfológica para o Sistema CATA. Como descrito na Seção de *Trabalhos relacionados* (3), o CoGrOO é um corretor gramatical livre para a língua portuguesa e possui várias ferramentas de processamento de textos (incluindo um Etiquetador Morfológico e um Lematizador) bastante simples de usar. No entanto, o CoGrOO mostrou-se ineficaz para determinar o lema de palavras "desconhecidas", como neologismos e estrangeirismos - que estão entre os principais alvos do verificador de

estilo produzido neste trabalho. Vejamos o resultado de alguns dos testes que levaram a tal conclusão (os testes completos foram incluídos no *Anexo B* (8.2)):

Foram escolhidas frases com estruturas simples como entrada para as ferramentas de Etiquetagem Morfológica e Lematização do CoGrOO.

1. **"debugar"**

   Entrada 1.1:
   ```
   Vamos debugar o sistema.
   ```
   Saída 1.1:
   ```
   Vamos        lemma[ir]        tag[V_PR_1P_IND_VFIN_]
   debugar      lemma[]          tag[KS_]
   o            lemma[o]         tag[DET_M_S_]
   sistema      lemma[sistema]   tag[N_M_S_]
   .            lemma[.]         tag[-PNT_ABS_]
   ```

   Entrada 1.2:
   ```
   Eu debuguei o sistema.
   ```
   Saída 1.2:
   ```
   Eu           lemma[eu]        tag[PERS_M/F_1S_NOM_]
   debuguei     lemma[]          tag[ADV_]
   o            lemma[o]         tag[DET_M_S_]
   sistema      lemma[sistema]   tag[N_M_S_]
   .            lemma[.]         tag[-PNT_ABS_]
   ```

2. **"randômico"**

   Entrada 2.1:
   ```
   O dado é randômico.
   ```
   Saída 2.1:
   ```
   O            lemma[o]         tag[DET_M_S_]
   dado         lemma[dado]      tag[N_M_S_]
   é            lemma[ser]       tag[V_PR_3S_IND_VFIN_]
   randômico    lemma[]          tag[N_M_S_]
   .            lemma[.]         tag[-PNT_ABS_]
   ```

   Entrada 2.2:
   ```
   Os dados são randômicos.
   ```
   Saída 2.2:
   ```
   Os           lemma[o]         tag[DET_M_P_]
   dados        lemma[dado]      tag[N_M_P_]
   são          lemma[ser]       tag[V_PR_3P_IND_VFIN_]
   randômicos   lemma[]          tag[V_M_P_PCP_]
   .            lemma[.]         tag[-PNT_ABS_]
   ```

Para nenhuma das entradas escolhidas, o CoGrOO conseguiu obter os lemas "debugar" e "randômico". Isso ocorreu devido ao fato de serem palavras que não existem no léxico da língua portuguesa. Mas a obtenção de tais lemas é fundamental para o funcionamento adequado do verificador de estilo. Assim, outra solução, baseada em [FM], foi utilizada.

4.3.1.1 FORMA

A estratégia utilizada para realizar a Lematização e Etiquetagem Morfológica para os textos em Português, chamada por seus autores de FORMA[FM], é probabilística e usa conjuntos de dados para embasar suas decisões (estes conjuntos de dados foram criados com base num *corpus* anotado com lemas e etiquetas morfológicas).

> Um *corpus* é um conjunto de dados que contém textos e informações linguísticas (chamadas de anotações), usado para análise estatística, construção de modelos e provas de conceito (*Proof of Concept*). Para desenvolver a ferramenta descrita nesta seção, foi usado o *corpus* Folha94[F94], uma compilação de artigos de 229 edições do jornal Folha de São Paulo, publicadas em 1994.

A partir dos conjuntos de dados supracitados, são construídos três autômatos e duas matrizes probabilísticas. O primeiro autômato (que chamaremos de $Autômato$ 1) serve para classificar formas compostas (e.g. "à primeira vista"). O segundo ($Autômato$ 2), por sua vez, serve para etiquetar palavras nas quais os acentos são uma característica diferenciadora (e.g. "forca" e "força" ou "e" e "é"). O último dos autômatos ($Autômato$ 3) é usado para determinar lemas e etiquetas com base em sufixos. As matrizes probabilísticas - que vamos chamar de BP e AP - são usadas para estimar a probabilidade de uma determinada etiqueta estar correta de acordo com o contexto. A matriz BP é uma matriz de dimensão 21×21 (o conjunto de etiquetas nas quais uma palavra pode ser classificada possui 21 elementos - vide *Anexo C* (8.3.1) para a lista completa) na qual cada elemento $bp_{mj} = Pr(\ j \mid m\ )$, ou seja, é a probabilidade da etiqueta j estar correta dado que a etiqueta m é a etiqueta do termo anterior no texto. Analogamente, a matriz AP é uma matriz 21×21 em que cada elemento $ap_{nj} = Pr(\ j \mid n\ )$, isto é, a probabilidade de que a etiqueta j esteja correta dado que a etiqueta n é a etiqueta do próximo termo do texto.

De posse das estruturas descritas, uma lista de *tokens* (obtida após a Segmentação) é recebida como entrada e, após seguir as etapas esquematizadas[FM] a seguir, produz como saída o texto lematizado[8].

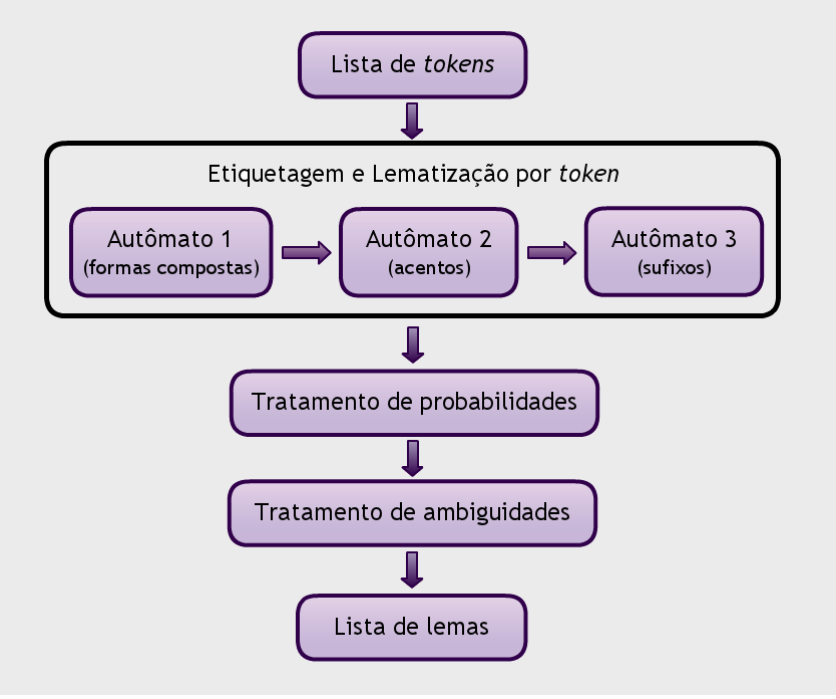


Figura 4.4: Lematização para textos em Português

[8] Na verdade, o algoritmo produz as etiquetas e os lemas para cada *token*, mas as etiquetas não serão utilizadas em etapas posteriores pelo Sistema CATA e, portanto, podem ser descartadas.

- **Etiquetagem e Lematização por *token*:** Para cada *token* t_i, esta etapa pode produzir dois tipos de saída. O primeiro tipo é

$$t_i \; L_i e_i \tag{1}$$

onde L_i é o lema definitivo e e_i é a etiqueta definitiva de t_i, que passa então a ser considerado um *token* etiquetado.
O outro tipo de saída é

$$t_i \; L_{0i} pe_{0i} \; L_{1i} pe_{1i} \; L_{2i} pe_{2i} \; ... \; L_{19i} pe_{19i} \; L_{20i} pe_{20i} \tag{2}$$

onde pe_{ji} é a probabilidade da etiqueta e_j para t_i, L_{ji} é o lema relacionado à etiqueta e_j para t_i (se $pe_{ji} = 0$, então L_{ji} é vazio). Em outras palavras, nesse segundo tipo de saída, para cada uma das 21 etiquetas possíveis, é atribuída uma probabilidade e um lema para o *token* t_i, que continua sendo considerado um *token* não etiquetado.
Esta etapa inicia-se então por um processamento no autômato para formas compostas, que lê sequências de quatro *tokens* por vez. Se houver um casamento entre um ou mais *tokens* e as formas compostas do $Autômato$ 1, então uma saída do primeiro tipo é produzida e essa fase está encerrada. Senão, o processamento continua com os próximos autômatos. O $Autômato$ 2 lê um *token* por vez e, se houver um casamento, esta etapa termina com uma saída do tipo (1). Finalmente, o $Autômato$ 3 (sufixos) - que também lê apenas um *token* por vez - produz, para cada *token*, uma saída do segundo tipo.

- **Tratamento de probabilidades:** Nesta etapa, *tokens* que ainda não foram etiquetados terão sua "vizinhança" analisada. Dada uma sequência de 3 *tokens* $t_{i-1} t_i t_{i+1}$ que inclui um *token* não etiquetado t_i, a nova probabilidade pe_{ji} para a etiqueta e_j de t_i é:

$$pe_{ji} = (b + 3pe_{ji} + a)/5$$

onde $b = bp_{mj} \in BP$ se m é a etiqueta definitiva para t_{i-1}, caso contrário, $b = 0$; e $a = ap_{nj} \in AP$ se n é a etiqueta definitiva de t_{i+1}, caso contrário, $a = 0$. Ao final desta etapa, a etiqueta com maior probabilidade e seu correspondente lema serão aplicados a t_i.

- **Tratamendo de ambiguidades:** Palavras ambíguas serão etiquetadas corretamente por meio da verificação de *tokens* vizinhos.

Em suma, o que este algoritmo aqui proposto[9] faz é tentar etiquetar *tokens* através de casamentos com termos cujas etiquetas são previamente conhecidas (autômatos 1 e 2). Se não for possível realizar a etiquetagem nesta primeira fase, a classificação do *token* será realizada inspecionando seu sufixo (terceiro autômato) e as etiquetas atribuídas a seus vizinhos no texto (etapas de probabilidade e desambiguação).

Vejamos agora como o algoritmo descrito lematiza as frases usadas para testar os componentes do CoGrOO (as frases são as mesmas apresentadas anteriormente, porém, esta solução espera como entrada uma lista de *tokens*):

1. **"debugar"**

   Entrada 3.1:
   ```
   {"vamos", "debugar", "o", "sistema", "."}
   ```
   Saída 3.1:

[9] O algoritmo FORMA é descrito em [FM]. Aqui é apresentada uma pequena variação.

```
{"ir", "debugar", "o", "sistema", "."}
```

Entrada 3.2:
```
{"eu", "debuguei", "o", "sistema", "."}
```
Saída 3.2:
```
{"eu", "debugar", "o", "sistema", "."}
```

2. **“randômico”**

Entrada 4.1:
```
{"o", "dado", "é", "randômico", "."}
```
Saída 4.1:
```
{"o", "dado", "ser", "randômico", "."}
```

Entrada 4.2:
```
{"os", "dados", "são", "randômicos", "."}
```
Saída 4.2:
```
{"o", "dado", "ser", "randômico", "."}
```

Note que os termos problemáticos foram lematizados corretamente.

4.3.2 Inglês

O Lematizador e o Etiquetador Morfológico do CoreNLP[SCN] da Stanford University foram utilizados no Sistema CATA para analisar textos em língua inglesa. A ideia do CoreNLP é que seus componentes possam ser treinados para qualquer língua, bastando, para isso, fornecer um *corpus* anotado adequado. Para este trabalho, foram usados componentes já treinados para o Inglês. As soluções do Stanford CoreNLP para Lematização e Etiquetagem Morfológica adotadas aqui baseiam-se em algoritmos log-lineares (descritos em [ME] e [CD]) e usam como etiquetas um conjunto composto por 45 elementos (listados no *Anexo C* (8.3.2)).

4.4 Busca de padrões em textos

Os métodos descritos anteriormente (Segmentação, Etiquetagem Morfológica e Lematização) são usados para pré-processar os textos submetidos como entrada para o Sistema CATA. Uma vez realizado este pré-processamento, é possível partir para a “busca” dos problemas de estilo. Como tais problemas foram restringidos apenas a termos inadequados, esta etapa reduz-se a uma busca de padrões. Mais formalmente, trata-se de uma busca de um conjunto finito de padrões - a saber, o conjunto dos termos considerados problemáticos. Para esta tarefa, foi adotado o algoritmo Aho-Corasick, que, basicamente, constrói uma máquina de estados (autômato finito) que assemelha-se a uma *trie*[10] com ponteiros entre os nós internos. O algoritmo será descrito com detalhes nas próxima seção (escrita com base em [AC] e [WK7]).

4.4.1 Aho-Corasick

Antes de descrevermos o algoritmo, vamos definir com mais cuidado o problema que estamos resolvendo:

[10] Uma *trie* é uma árvore na qual a posição dos nós definem suas chaves.

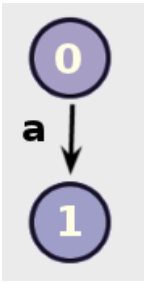
Vamos chamar de *cadeia* uma sequência finita de caracteres.
Seja $K = \{y_1, y_2, ..., y_k\}$ um conjunto finito de cadeias, que serão chamadas de *chaves*. K será chamado de *dicionário*.
Seja x uma cadeia arbitrária qualquer, que será chamada de *texto*.
O problema consiste em localizar e identificar todas as *subcadeias* de x que estão em K (tais subcadeias podem se sobrepor). Assim, o algoritmo recebe como entrada o conjunto K e o *texto* x e produz como saída todos os índices de x nos quais *cadeias* de K aparecem como *subcadeias*.

O algoritmo começa pela construção do autômato, que ocorre através do cálculo de três funções: *goto*, *output* e *failure*.

O cálculo da função *goto* consiste na construção de um grafo dirigido. Inicialmente, tal digrafo possui apenas um vértice, que chamaremos de *estado inicial* (ou vértice 0). Para cada *chave* y do *dicionário*, é acrescentado um caminho que começa no *estado inicial*. Para isso, novos vértices e arcos são acrescentados ao digrafo, de modo que haja um caminho que começa no vértice 0 e vai até um outro vértice e que "soletra" a *chave* y. Por exemplo, suponha que o *dicionário* é o conjunto $K = \{a, ab, bc, bca, c, caa\}$.

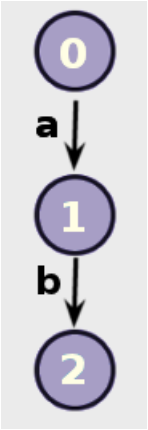
1. No início, o digrafo possui apenas o *estado inicial*.

2. Após a inserção da primeira *chave* ("a") no digrafo, obtemos:



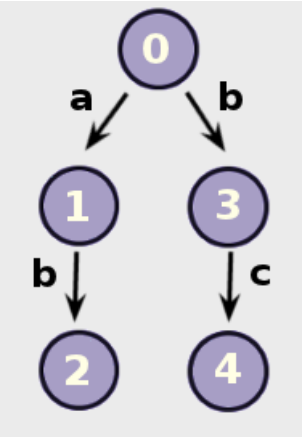
   O caminho do *estado inicial* ao vértice 1 soletra "a" (indicamos $g(0,\text{'a'}) = 1$ e $g(0,\sigma) = \textit{fail}$ $\forall\ \sigma \neq$ 'a').

3. A próxima chave é *"ab"*:



Note que não foi acrescentado um novo vértice para o caractere 'a', pois já havia um caminho com etiqueta "a" do *estado inicial* para o vértice 1.

4. A próxima inserção é a de "bc":



Note como há um caminho do *estado inicial* até o vértice *4* que "soletra bc" (indicamos $g(0,\text{'b'}) = 3$ e $g(3,\text{'c'}) = 4$).

5. Repetimos este processo para todas as demais *chaves* do *dicionário*. No fim, acrescentamos um laço no *estado inicial* para todo σ tal que $g(0,\sigma) = fail$. O resultado do cálculo da função *goto* seria:

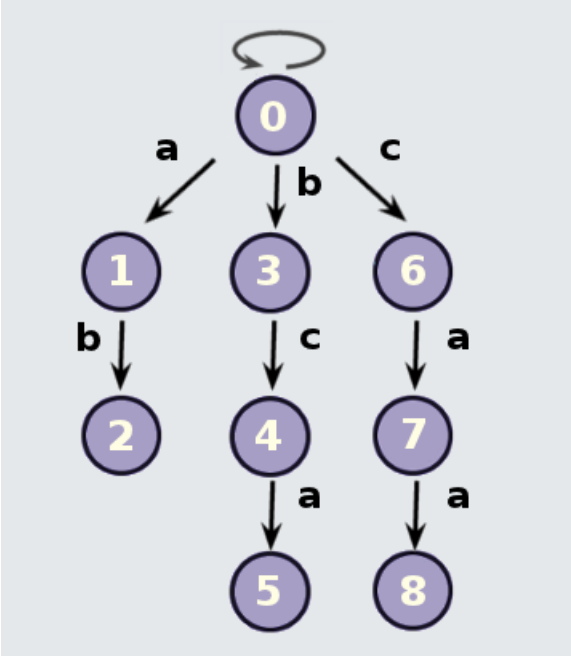


Figura 4.5: Função *goto*.

A função *output* é uma função de vértices (ou estados) em subconjuntos de K. Informalmente, ao realizarmos um passeio no digrafo (ou autômato) e "passarmos" pelos vértices, a função *output* dirá, para cada um deles, que *cadeias* do *dicionário* foram encontradas.

A função *output* é parcialmente calculada durante a construção do digrafo da função *goto*: para cada *chave* y que foi inserida, temos um vértice s cujo caminho de 0 a s possui como etiqueta a *chave* y e, portanto, $y \in output(s)$. Para o nosso exemplo, após a construção do digrafo da função *goto*, teríamos calculada parcialmente a função *output*, conforme mostra a tabela:

i	$output(i)$
0	∅
1	{a}
2	{ab}
3	∅
4	{bc}
5	{bca}
6	{c}
7	∅
8	{caa}

Tabela 4.1: Função *output* (parcialmente calculada).

A seguir, uma apresentação mais formal[AC] dos algoritmos para calcular as funções *goto* e *output*.

Algoritmo 4.1: Construção da função *goto*

Entrada: Um *dicionário* $K = \{y_1, y_2, ..., y_k\}$.
Saída: Função *goto* g e função *output* (parcialmente calculada).
Dados: Estamos supondo que $output(s)$ é o conjunto vazio quando um estado s é criado e $g(s,\sigma) = fail$ se σ é indefinido ou se $g(s,\sigma)$ ainda não foi definido. *novoEstado* é uma variável global.

```
1  início
2      novoEstado ← 0
3      para i ← 1 até k faça
4          insira(y_i)
5      para todo σ tal que g(0,σ) = fail faça
6          g(0,σ) ← 0
7  fim
```

Algoritmo 4.2: Função $insira(a_1 a_2 ... a_m)$

Entrada: Uma *cadeia*.

```
1  início
2      estado ← 0
3      j ← 1
4      enquanto g(estado, a_j) ≠ fail faça
5          estado ← g(estado, a_j)
6          j ← j + 1
7      para p ← j até m faça
8          novoEstado ← novoEstado +1
9          g(estado, a_p) ← novoEstado
10         estado ← novoEstado
11     output(estado) ← {(a_1 a_2 ... a_m)}
12 fim
```

Resta agora terminar o cálculo da função *output* e calcular a função *failure*. O cálculo da função *failure* [11] é necessário para "transformar" o digrafo obtido em um autômato.

Algoritmo 4.3: Construção da função *failure*

Entrada: Função *goto* e função *output* obtidas no Algoritmo 4.1.
Saída: Funções *failure* f e *output* (completa).

```
1  início
2      fila ← ∅
3      para todo σ tal que g(0,σ) = s ≠ 0 faça
4          fila ← fila ∪ {s}
5          f(s) ← 0
6      enquanto fila ≠ ∅ faça
7          seja r o próximo estado da fila
8          fila ← fila − {r}
9          para todo σ tal que g(r,σ) = s ≠ fail faça
10             fila = fila ∪ {s}
11             estado ← f(r)
12             enquanto g(estado,σ) = fail faça
13                 estado ← f(estado)
14             f(s) ← g(estado,σ)
15             output(s) ← output(s) ∪ output(f(s))
16 fim
```

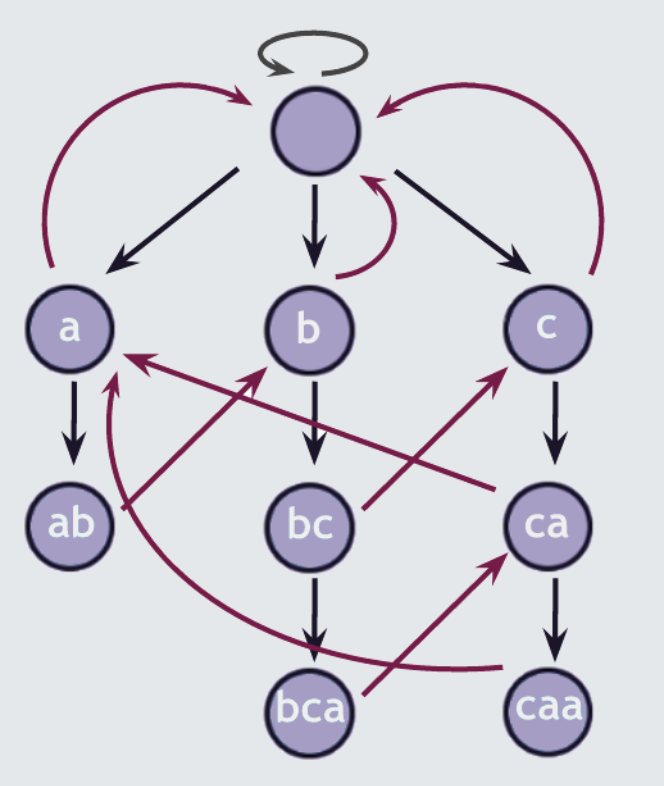


Figura 4.6: Autômato construído para o *dicionário* $K = \{a, ab, bc, bca, c, caa\}$ (os arcos em rosa representam transições *failure*).

[11] A função *failure* não é definida para o *estado inicial*.

A função *output* para o $K = \{a,\ ab,\ bc,\ bca,\ c,\ caa\}$ ficaria assim, após completamente calculada:

i	$output(i)$
0	∅
1	{a}
2	{ab}
3	∅
4	{bc, c}
5	{bca, a}
6	{c}
7	∅
8	{caa, a}

Tabela 4.2: Função *output* (completa).

Uma vez construído o autômato, podemos usá-lo para efetuar a busca pelos padrões do *dicionário*. A máquina de estados processa o *texto x* lendo sucessivamente seus caracteres, efetuando transições e, ocasionalmente, emitindo uma saída. O comportamento do autômato é definido pelas três funções calculadas anteriormente (*goto*, *output* e *failure*).

Algoritmo 4.4: Busca de padrões

Entrada: Um *texto* $x = a_1 a_2 ... a_n$ e uma máquina de estados M construída a partir das funções *goto*, *output* e *failure*.

Saída: Índices nos quais ocorrem *chaves* do *dicionário*.

```
1  início
2      estado ← 0
3      para i ← 1 até n faça
4          enquanto g(estado, a_i) = fail faça
5              estado ← f(estado)
6          estado ← g(estado, a_i)
7          se output(estado) ≠ ∅ então
8              imprima i
9              imprima output(estado)
10 fim
```

4.4.1.1 Complexidade

De fato, o Algoritmo Aho-Corasick é bastante eficiente: dado que as funções *goto*, *output* e *failure* já estão calculadas, a busca de padrões pode ser realizada em tempo linear. Para provarmos isso, precisamos de uma definição simples: chamaremos de *profundidade* de um vértice s ($p(s)$, no digrafo produzido no cálculo da função *goto*) o comprimento do menor caminho que vai do vértice 0 a s.

Afirmamos que, dadas as saídas criadas pelas funções goto, failure e output, o Algoritmo 4.4 realiza menos de $2n$ *transições para processar um texto de comprimento* n.

Demonstração. Em cada iteração, o *Algoritmo 4.4* faz zero ou mais transições *failure* seguidas por exatamente uma transição *goto*. De um estado s de *profundidade* d, o *Algoritmo 4.4* não pode fazer mais que d transições *failure* numa iteração (isto decorre da própria construção da função *failure*:

para todo vértice $s \neq 0$ vale que $p(s) > p(f(s)))$. Portanto, o número total de transições *failure* (levando em conta todas as iterações) é, no máximo, o número total de transições *goto* menos um. Ao processar uma entrada de comprimento n, o algoritmo realiza exatamente n *goto* transições (o laço é executado n vezes). Portanto, o número total de transições de estado é menor que $2n$. □

4.5 Melhorando a eficácia ao longo do tempo

Apesar do problema da verificação de estilo ter sido simplificado para poder ser tratado por um sistema computacional, ainda assim as técnicas descritas até aqui não são suficientes para que a análise dos textos seja sempre adequada. Consideremos este exemplo: o termo "cores" pode assumir vários significados: pode ser a expressão em Inglês para "núcleos" (de processadores) ou o plural do substantivo "cor". O primeiro caso é considerado um estrangeirismo inadequado e o sistema deveria marcá-lo como problema. Contudo, pode ser que o sistema aponte como erro de estilo o termo "cores" num contexto em que a palavra assume o segundo significado. Para lidar com este tipo de situação, foram aplicadas técnicas para usar a Inteligência Coletiva. Assim, quando um problema de estilo é indicado, o usuário pode reportar ao sistema se concorda ou não com aquela indicação . Em ambos os casos, o *software* infere qual a semântica do contexto (de que assunto trata o texto no trecho em que foi detectado o problema de estilo) e armazena esta informação para, futuramente, aplicar ou não a sugestão de estilo num termo que está inserido num contexto similar.

4.5.1 Extração de palavras-chave e similaridade semântica entre textos

Para inferir qual a semântica dos textos, a estratégia utilizada foi a extração de palavras-chave.

> Uma **palavra-chave** é uma palavra (ou sequência de palavras) que sintetiza os assuntos principais de um texto. Em particular, para este trabalho, para que um termo seja palavra-chave de um texto, ele tem que necessariamente estar contido no texto.

As pesquisas por estratégias para realizar este tipo de tarefa levaram a vários resultados e técnicas, como o [KEA]. Contudo, a adaptação dessas técnicas para a língua portuguesa é demasiado complexa, pois envolve a seleção de um *corpus* adequado e devidamente etiquetado. Assim, uma solução mais simples, descrita em [KE], foi adotada.

4.5.1.1 Extração de palavras-chave sem *corpus*

O algoritmo brevemente esboçado aqui serve para extrair as palavras-chave de um texto qualquer (não é necessário que o texto esteja em um determinado idioma ou seja de um domínio específico). Além disso, não é necessário prover um *corpus* para que o algoritmo "aprenda" a extrair as palavras-chave.

Antes de extrair as palavras-chave, o texto é pré-processado: é realizada a detecção de sentenças (o texto é dividido em frases) e a Segmentação (divisão das frases em *tokens*). Em seguida, é efetuada a Radicalização (*Stemming*) dos *tokens*, que consiste em reduzi-los aos seus *radicais*.

> O Radical é o elemento portador de significado, comum a um grupo de palavras da mesma família[RA]: **sonh-** é radical de **sonh**o e **sonh**ador.
> A etapa de Radicalização é importante para a tarefa de extração de palavras-chave porque um texto que contém o termo "durável" pode ter o mesmo assunto que textos que contenham as palavras "durabilidade" e "duradouro".

Durante este pré-processamento do texto ainda são eliminados elementos que não são candidatos a palavras-chave, como pontuação ou palavras como “as”, “e”, “com”, etc.

Em linhas gerais, para determinar as palavras-chave do texto, são determinados os termos mais frequentes (vamos chamar de F o conjunto de termos mais frequentes) dentre os “termos *radicalizados*” obtidos no processamento anterior. A seguir, é calculada uma matriz de co-ocorrência M, na qual cada elemento m_{ab} representa o número de frases em que um termo qualquer a do texto ocorreu com um termo frequente $b \in F$. Serão admitidos como palavras-chaves do texto os termos mais frequentes e aqueles cuja distribuição de probabilidade da co-ocorrência é viesada para um determinado subconjunto de F.

Uma vez que extraímos as palavras-chave dos textos, temos informações sobre suas semânticas e podemos, portanto, compará-los e, eventualmente, não aplicar uma determinada sugestão de estilo num determinado contexto. Para medir o quanto dois textos são parecidos com relação ao assunto, foi usada uma **escala de similaridade**. Dois textos (ou trechos de textos) serão mais similares quanto mais palavras-chave em comum tiverem.

5 Arquitetura e implementação

O Sistema CATA é uma aplicação Web e foi implementado em Java, uma linguagem de programação orientada a objetos.

5.1 Ambiente de desenvolvimento

O Sistema CATA foi desenvolvido usando o Eclipse, um ambiente de desenvolvimento integrado livre.

5.2 Modelo-Visão-Controle (MVC)

O Modelo-Visão-Controle (em Inglês *Model-View-Controller*) é um padrão de projeto arquitetural baseado no desacoplamento entre *interface* e lógica de domínio.

> O *modelo* está relacionado ao *estado* da aplicação: é a representação das informações sobre as quais o sistema opera, provendo meios de acesso e alteração de dados.
> A *visão* é a apresentação do conteúdo e da lógica de processamento de uma forma adequada para permitir interação, fornecendo toda a funcionalidade de processamento requerida pelo usuário final.
> O *controlador* responde a eventos, gerencia o acesso ao *modelo* e à *visão* e coordena o fluxo de dados entre eles.[ES]

As vantagens do MVC residem no fato dele permitir independência entre desenvolvimento, teste e manutenção de cada aspecto da aplicação (lógica de entrada, lógica de negócio e lógica de *interface*).

O Sistema CATA segue o padrão arquitetural MVC e foi desenvolvido sobre o VRaptor, um arcabouço implementado em Java.

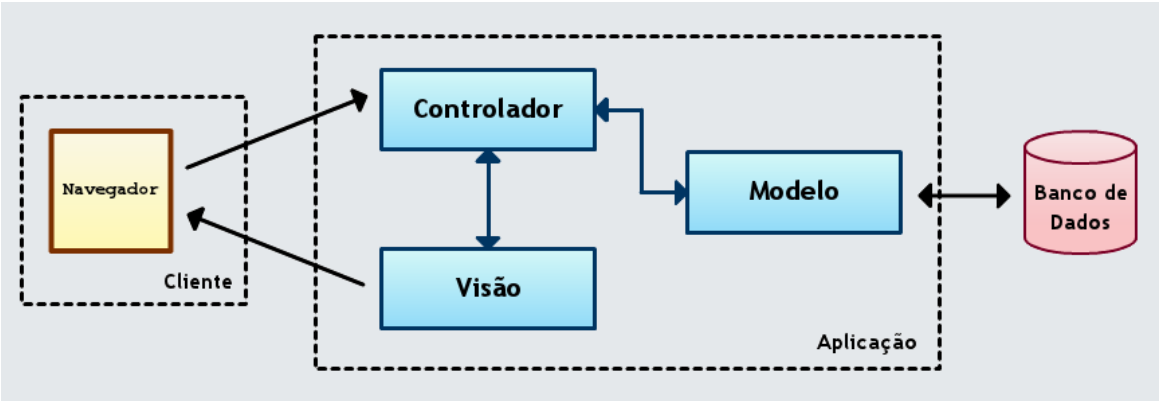


Figura 5.1: Arquitetura MVC do Sistema CATA

5.3 Fluxo de trabalho

O Sistema CATA espera, como entrada, um texto acadêmico de Computação e produz, como saída, o mesmo texto, mas com os problemas de estilo (acompanhados das respectivas sugestões) marcados. A partir do momento em que um arquivo é submetido ao CATA, uma série de processamentos acontecem: primeiro ocorre a "extração" do texto - precisamos obter um texto sem formatação a partir do arquivo enviado. Em seguida, são realizados os processos de Segmentação,

Etiquetagem Morfológica e Lematização. Finalmente, os problemas de estilo são detectados e o resultado é então apresentado ao usuário.

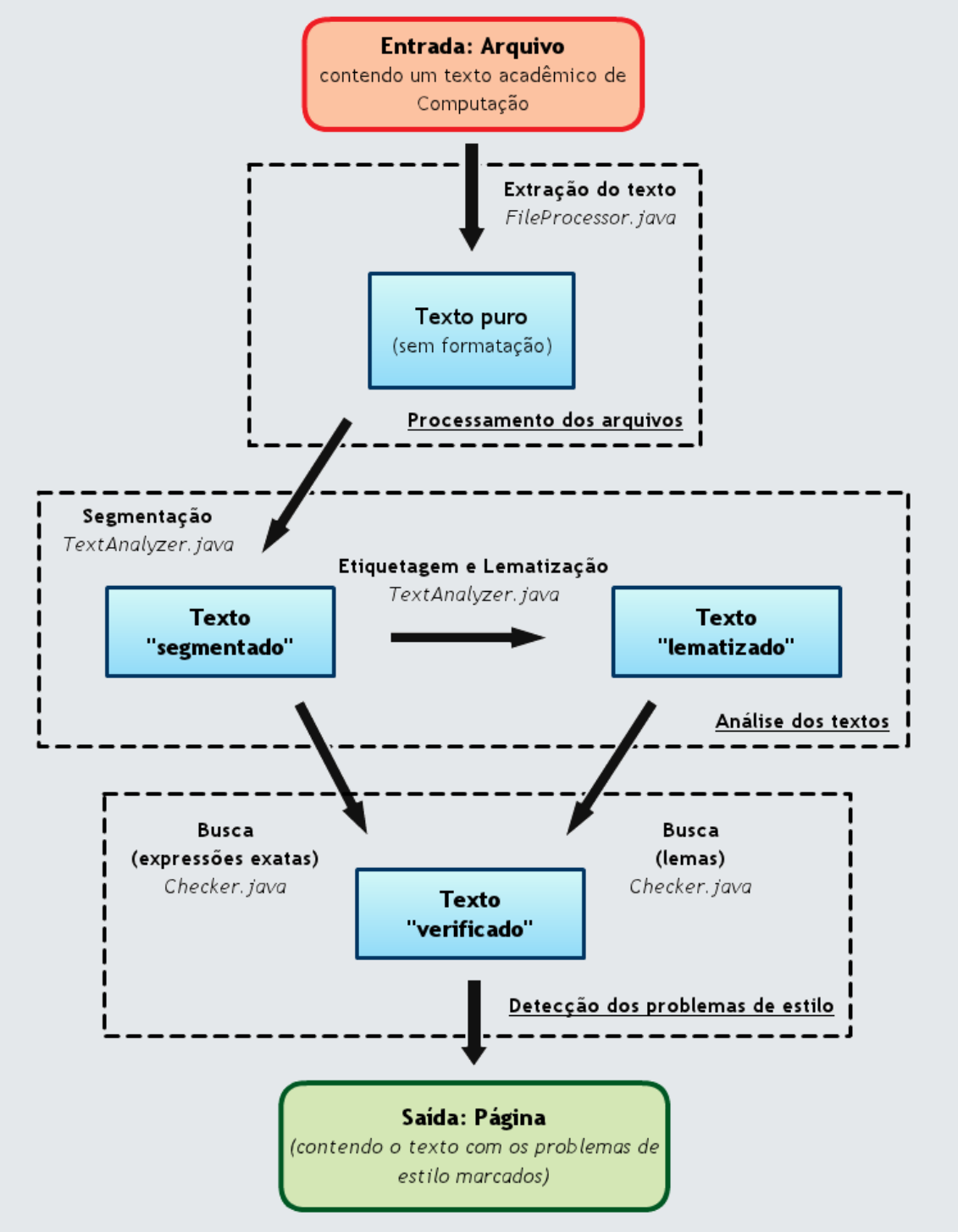


Figura 5.2: Fluxo de processamento no Sistema CATA

5.3.1 Processamento dos arquivos

Dado um arquivo que contém um texto acadêmico de Computação, a primeira tarefa do Sistema CATA é "extrair" o texto desse arquivo. Para arquivos de textos sem formatação (*.txt*), isso corresponde a "ler" o arquivo com a codificação correta. Se forem arquivos *Portable Document Format* (*.pdf*), a extração do texto é realizada com o auxílio de uma ferramenta - também escrita em Java - o PDFBox.

5.3.2 Análise dos textos

Uma vez que o texto foi extraído, a próxima etapa a ser executada é a Segmentação: o texto é "quebrado" em suas unidades fundamentais, os *tokens*. Para o caso de textos em Português, a ferramenta escolhida para realizar esta etapa é a que está presente no CoGrOO (o *CoGrOO SentenceDetector* e o *CoGrOO Tokenizer*); já para textos em língua inglesa, a Segmentação é efetuada por um componente do CoreNLP.

A Segmentação produz o texto "segmentado" e, além disso, também constrói um mapeamento entre este texto obtido e o texto original. Dessa forma, quando um problema de estilo for encontrado entre os *tokens* do texto "segmentado", será possível saber exatamente a qual posição do texto original tal problema corresponde.

O texto "segmentado" é então submetido aos processos de Etiquetagem Morfológica e Lematização, para obtermos o "texto lematizado". Se o texto estiver em Português, será etiquetado e lematizado por uma implementação em Java do algoritmo FORMA (4.3.1.1). Para textos em Inglês, são usados componentes do CoreNLP. Aqui também será necessário mapear os lemas obtidos para o texto original. Assim, se um problema de estilo for encontrado entre os lemas do texto "lematizado", será possível recuperar exatamente as palavras do texto original às quais os lemas problemáticos correspondem.

5.3.3 Detecção dos problemas de estilo

Para a busca dos problemas de estilo, são usados dois autômatos (construídos conforme o Algoritmo Aho-Corasick (4.4.1)): um para expressões exatas e outro para lemas. O primeiro deles é usado para detectar problemas de estilo no texto "segmentado", o segundo, para a detecção no texto "lematizado".

No Sistema CATA, uma "regra" pode ser usada para detectar lemas problemáticos ou expressões exatas inadequadas. Por exemplo, uma das regras padrões do CATA é o par "randômico" (termo inadequado) / "aleatório" (sugestão para substituir o termo inadequado). Com esta regra, espera-se que as flexões "randômicos", "randômica" e "randômicas" sejam detectadas. Assim, a busca por "randômico" deve ocorrer no texto "lematizado". Por outro lado, considere outra regra do CATA: "softwares" (termo inadequado) / "sistemas de software" (sugestão para substituir o termo inadequado). Neste caso, não gostaríamos que o sistema apontasse como erro a palavra "software", pois só existe um problema de estilo se o termo estiver no plural (é deselegante usar o termo "software" no plural). Assim, a "regra dos softwares" deverá ser aplicada ao texto "segmentado". Se aplicássemos as regras somente ao texto "lematizado", o padrão "softwares" nunca seria encontrado: pois o lema de "softwares" é "software", e assim, não ocorre o termo "softwares" no texto "lematizado".

Após as buscas nos autômatos, os erros encontrados são "revisados": o Sistema infere a semântica do texto (de que assunto tratam os textos) e remove as marcações de problemas de estilo que não estejam adequadas ao contexto.

5.4 *Interface* gráfica

A *interface* do Sistema foi projetada com vistas a produzir uma interação fácil e intuitiva e um produto bonito esteticamente. Para isso, vários fatores foram considerados, como tamanho da fonte, por exemplo. Outro detalhe que mereceu atenção foi vincular cores e outros elementos gráficos a informações relevantes. Por exemplo: se no preenchimento de um formulário por um usuário houver problema como um determinado campo, tal campo será destacado em vermelho, para que o usuário possa descobrir rapidamente porque seu preenchimento não foi aceito.

Como o Sistema CATA é uma aplicação Web, a *interface* foi construída usando HTML - *HyperText Markup Language*, uma linguagem de marcação para a Web, CSS - *Cascading Style Sheets*,

uma linguagem de estilo, JavaScript - uma linguagem de *script*, jQuery - uma biblioteca em JavaScript de fácil uso, e Ajax - *Asynchronous Javascript and XML*, usado para trazer mais dinamismo às páginas.

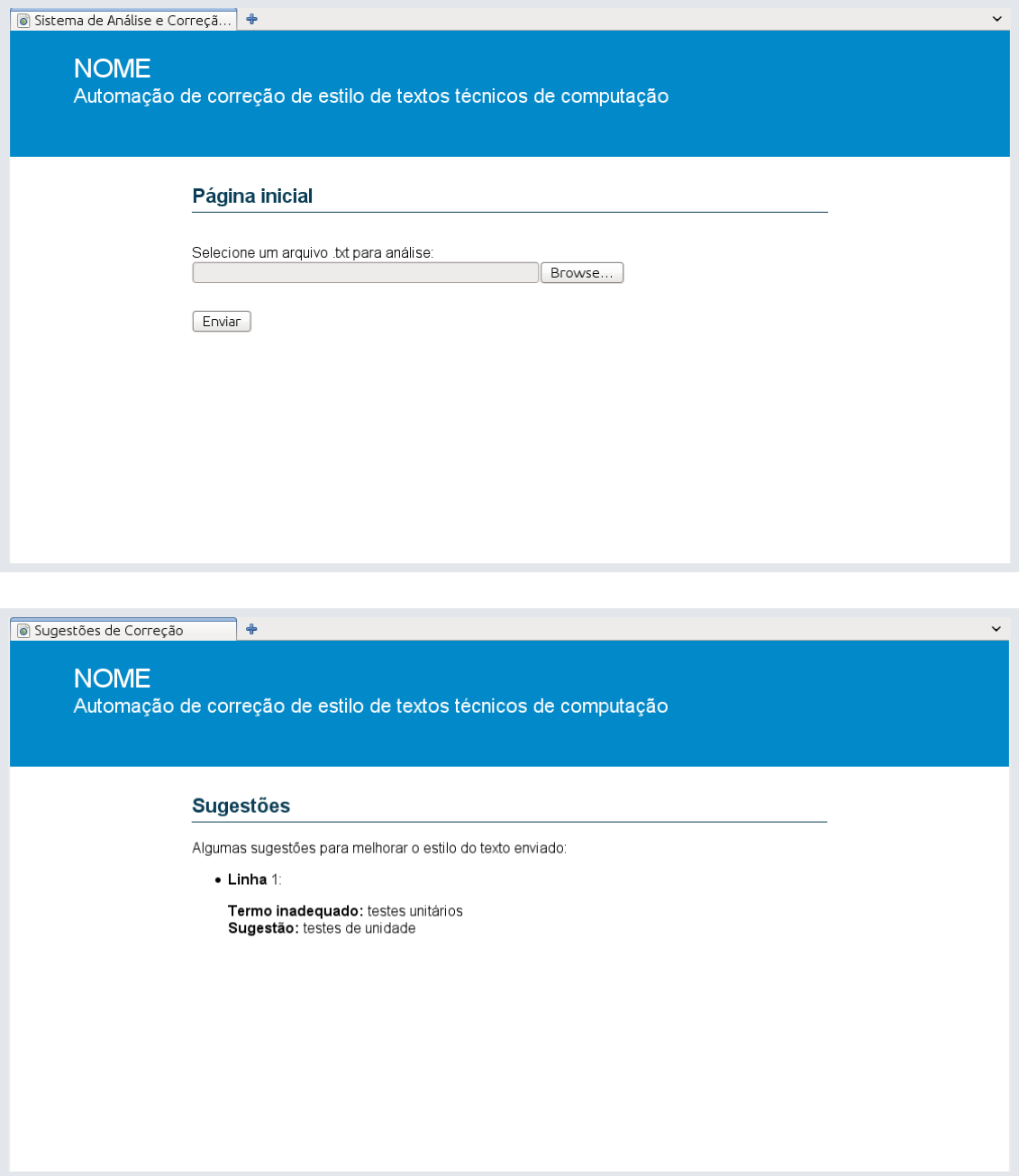


Figura 5.3: Imagens do Sistema CATA em abril de 2011. A primeira imagem apresenta a página inicial. A imagem mais abaixo mostra como eram dadas as sugestões de estilo. A evolução da *interface* pode ser observada ao compararmos estas imagens com as incluídas na Seção de *Resultados* (6.1).

5.5 Outros aspectos

Como já mencionado, os usuários podem se registrar no Sistema CATA e cadastrar suas próprias regras e sugestões de estilo. Para isso, foi necessário implementar um mecanismo de persistência: com esse propósito foram usados o Hibernate e o MySQL. Além disso, o servidor usado para executar o Sistema CATA foi o Apache Tomcat.

6 Resultados

6.1 Produto obtido

O produto obtido foi o Sistema CATA, cujo código fonte está **totalmente** disponível em um repositório no GitHub ([GH]).

O sistema vai ao encontro dos objetivos deste trabalho: ele é capaz de detectar problemas de estilo em textos acadêmicos de Computação (tanto em Português como em Inglês) e melhora sua avaliação ao longo do tempo. Além disso, os usuários podem se registrar no CATA, e assim, podem cadastrar suas próprias regras de estilo, as quais ficarão disponíveis a todos os demais usuários.

Figura 6: Página inicial do Sistema CATA e sugestões de estilo.

6.2 Testes

Para avaliar a eficácia do Sistema CATA, foi utilizado um conjunto de doze textos acadêmicos de Computação. Tratam-se de textos em Português, todos monografias de conclusão do curso de Ciência da Computação (retiradas de [CEF]).

Os resultados da Tabela 6.1 foram obtidos ao submeter como entrada os textos do conjunto de testes. Em **todos** foram encontrados problemas de estilo:

Total de problemas de estilo encontrados	Verdadeiros positivos	Falsos positivos
271	236	35
100%	87.08%	12.92%

Tabela 6.1: Total de problemas de estilo encontrados pelo Sistema CATA.

A Tabela 6.2 mostra os erros mais frequentes[12] (contando apenas os verdadeiros positivos):

Problema de estilo	Frequência absoluta	Frequência (%)
thread	26	11.02%
site	22	9.32%
design	19	8.05%
suportar	19	8.05%
parser	18	7.63%
retornar	18	7.63%
framework	17	7.21%
rodar	15	6.36%
softwares	11	4.66%
checar	8	3.39%
log	8	3.39%
bug	6	2.54%
corretude	6	2.54%
patch	6	2.54%
template	6	2.54%
overhead	5	2.12%
performance	5	2.12%
layout	4	1.70%
assumir	3	1.27%
deadlock	3	1.27%
string	3	1.27%
grid	2	0.85%
kernel	2	0.85%
I/O	1	0.42%
cluster	1	0.42%
peer-to-peer	1	0.42%
round-robin	1	0.42%

Tabela 6.2: Distribuição dos problemas encontrados.

Para mais informações sobre os resultados dos testes, consulte o *Anexo D* (8.4).

[12] É importante salientar que todas as flexões de um determinado problema foram incluídas na contagem de tal problema: e.g. “suportam” e “suportado” são contados como “suportar”.

7 Conclusão

Observando os resultados dos testes, a conclusão mais importante é com relação à relevância deste trabalho: em todos os textos do conjunto de testes foram encontrados problemas de estilo. Isso comprova a dificuldade que os alunos de Computação encontram ao redigir textos acadêmicos.

Com relação à eficácia do sistema implementado, podemos afirmar que trata-se de um sistema eficaz: para os testes, a taxa de erros ficou em torno de 13%. Se desconsiderarmos os falsos positivos que ocorreram apenas por casamentos com títulos de obras citadas nas referências bibliográficas, essa taxa cai para 5.2% (seria um total de 249 problemas, sendo 13 falsos positivos).

Portanto, o Sistema CATA prova que é viável e relevante a implementação de um sistema computacional para analisar o estilo de textos de Computação, mesmo simplificando o que é considerado problema de estilo.

Sendo a escrita uma das principais formas de comunicação científica, melhorar o estilo dos textos acadêmicos significa melhorar a comunicação e, consequentemente, a colaboração entre os acadêmicos. Assim, concluímos que a adoção de um sistema como o CATA poderia beneficiar toda a comunidade científica/acadêmica de Computação.

8 Anexos

8.1 Anexo A - Regras padrões do Sistema CATA

As regras padrões do Sistema CATA podem ser “lemas” ou “expressões exatas”: no primeiro caso, a intenção é que todas as flexões do problema de estilo sejam encontradas; no segundo, o sistema só marcará como problemas de estilo os termos que forem iguais ao padrão cadastrado na regra.

8.1.1 Português

As regras de estilo para a Língua Portuguesa foram retiradas de [FK1] e [MF].

Problema de estilo	**Sugestões**
assembler	montador
assembly language	linguagem de montagem
assumir (com o sentido de admitir por hipótese, supor - e.g. “Assuma que *x* é maior que zero”)	considerar, supor
audio	áudio
binding	enlace, associação
browser	navegador
bug	erro
checar	verificar
checkpoint	ponto de salvaguarda
cluster	aglomerado, agrupamento
comitar	consolidar, efetivar
commit	consolidação
container	contêiner
core	núcleo
corretude	correção
customizar	configurar, particularizar, personalizar
deadlock	impasse, travamento
debugar	depurar
debugger	depurador
debugging	depuração
default	padrão, valor padrão
delay	atraso
deployment	implantação
design	desenho, projeto
dynamic binding	enlace dinâmico, associação
elicitar	capturar, coletar, descobrir, eliciar (fig.), extrair, obter, prospectar
embebido (quando tradução do inglês “embedded”)	embutido
failure	defeito, avaria, (falha pode ser usada se o contexto permitir)
fault	falha, falta
framework	arcabouço

Tabela A.1: Regras padrões: lemas.

Problema de estilo	**Sugestões**
grid	grade
hack	gambiarra
hash function	função de espalhamento, função de resumo criptográfico ou, simplesmente, resumo criptográfico
hash table	tabela de dispersão, tabela de espalhamento
host	máquina, hospedeiro, computador
job	tarefa
kernel	núcleo
layout	leiaute, formato
linker	ligador
lock	bloqueio
log	registro
mandatório	obrigatório
nested	aninhado, encaixado
offset	deslocamento
overhead	carga extra, custo adicional, sobrecarga
paper	documento, texto acadêmico
parser	analisador, analisador sintático
patch	componente de atualização, remendo
peer	par
performance	desempenho
pipe	duto
polling	consulta periódica, varredura
prefecthing	transferência antecipada, busca antecipada
procedural	procedimental
proxy	procurador, representante
randômico	aleatório
requerimento (quando tradução do inglês “requirement”)	requisito
request	pedido, requisição
retornar	devolver
rodar	executar
router	roteador
site	sítio
starvation	inanição
string	cadeia de caracteres
suportar	comportar, contemplar, dar suporte, disponibilizar, implementar, incluir, oferecer, prover
switch	comutador (de pacotes)
template	gabarito, modelo, molde
testbed	ambiente de/para testes
teste unitário	teste de unidade
thread	fluxo de execução, linha de execução, processo leve
throughput	vazão
trade-off	dicotomia (é melhor explicar a relação por extenso)

Tabela A.1 (continuação): Regras padrões: lemas.

Problema de estilo	**Sugestões**
a grosso modo	grosso modo
Autonomic Computing	Computação Autônoma
Computação Autonômica	Computação Autônoma
Computação Pervasiva	Computação Ubíqua
Cloud Computing	Computação na Nuvem, Computação em Nuvem
data mining	mineração de dados
eXtreme Programming	Programação eXtrema
I/O	E/S
middlewares	sistemas de middleware
Pervasive Computing	Computação Ubíqua
round-robin	rodízio
softwares	sistemas de software
stakeholders	interessados, partes interessadas
test-driven development	programação dirigida por testes

Tabela A.2: Regras padrões: expressões exatas.

8.1.2 Inglês

As regras de estilo para o Inglês foram retiradas de [FK2] e [DU].

Problema de estilo	**Sugestões**
(a) new initiative(s)	(an) initiative(s)
both cultures are (to be) equally affected	the cultures are (to be) equally affected
elucidate	show
etiology	cause
has (to have) the capacity to	can
methodology	method
to be in agreement	agree
to play a key role in	to be essential to
used for fuel purposes	used for fuel
utilize	use

Tabela A.3: Regras padrões: lemas.

Problema de estilo	**Sugestões**
a large majority of	most
aren't, isn't (e outras contrações)	are not, is not
at this point in time	now
due to the fact that	because
in a careful manner	carefully
in the event that	if
nearly unique	unique/rare
prior to	before
putative	(nada)
subsequent to	after
the question as to whether	whether
there is no doubt but that	doubtless
this is a subject that	this subject
whether or not	whether

Tabela A.4: Regras padrões: expressões exatas.

8.2 Anexo B - Testes realizados com o CoGrOO

Os testes foram realizados com a versão 3.1.2 do CoGrOO, usando métodos contidos entre os arquivos de exemplo do CoGrOO (`CommandLineOwnPipelineAnalyzer.java`).

1. **"bug"**

Entrada 1.1:

```
Um bug foi encontrado.
```

Saída 1.1:

```
Um          lemma[um]         tag[DET_M_S_]
bug         lemma[]           tag[N_M_S_]
foi         lemma[foi]        tag[V_PS_3S_IND_VFIN_]
encontrado  lemma[encontrar]  tag[V_M_S_PCP_]
.           lemma[.]          tag[-PNT_ABS_]
```

Entrada 1.2:

```
Vários bugs foram encontrados.
```

Saída 1.2:

```
Vários       lemma[vários]     tag[DET_M_P_]
bugs         lemma[]           tag[N_M_P_]
foram        lemma[foram]      tag[V_PS/MQP_3P_IND_VFIN_]
encontrados  lemma[encontrar]  tag[V_M_P_PCP_]
.            lemma[.]          tag[-PNT_ABS_]
```

2. **"corretude"**

Entrada 2.1:

```
A corretude do algoritmo foi provada.
```

Saída 2.1:

```
A           lemma[o]          tag[DET_F_S_]
corretude   lemma[]           tag[N_F_S_]
do          lemma[de]         tag[PRP_]
do          lemma[o]          tag[DET_M_S_]
algoritmo   lemma[algoritmo]  tag[N_M_S_]
foi         lemma[foi]        tag[V_PS_3S_IND_VFIN_]
provada     lemma[provar]     tag[V_F_S_PCP_]
.           lemma[.]          tag[-PNT_ABS_]
```

3. **"customizar"**

Entrada 3.1:

```
Vamos customizar a barra de ferramentas.
```

Saída 3.1:

```
Vamos       lemma[ir]         tag[V_PR_1P_IND_VFIN_]
customizar  lemma[]           tag[KS_]
a           lemma[a]          tag[DET_F_S_]
barra       lemma[barra]      tag[N_F_S_]
```

```
de           lemma[de]          tag[PRP_]
ferramentas  lemma[ferramenta]  tag[N_F_P_]
.            lemma[.]           tag[-PNT_ABS_]
```

Entrada 3.2:

```
Eu customizei a barra de ferramentas.
```

Saída 3.2:

```
Eu           lemma[eu]          tag[PERS_M/F_1S_NOM_]
customizei   lemma[]            tag[V_PR_1S_IND_VFIN_]
a            lemma[a]           tag[DET_F_S_]
barra        lemma[barra]       tag[N_F_S_]
de           lemma[de]          tag[PRP_]
ferramentas  lemma[ferramenta]  tag[N_F_P_]
.            lemma[.]           tag[-PNT_ABS_]
```

Entrada 3.3:

```
Ele customizará a barra de ferramentas.
```

Saída 3.3:

```
Ele          lemma[ele]         tag[PERS_M_3S_NOM_]
customizará  lemma[]            tag[V_PR_3S_IND_VFIN_]
a            lemma[a]           tag[DET_F_S_]
barra        lemma[barra]       tag[N_F_S_]
de           lemma[de]          tag[PRP_]
ferramentas  lemma[ferramenta]  tag[N_F_P_]
.            lemma[.]           tag[-PNT_ABS_]
```

4. **“debugar”**

Entrada 4.1:

```
Vamos debugar o sistema.
```

Saída 4.1:

```
Vamos        lemma[ir]          tag[V_PR_1P_IND_VFIN_]
debugar      lemma[]            tag[KS_]
o            lemma[o]           tag[DET_M_S_]
sistema      lemma[sistema]     tag[N_M_S_]
.            lemma[.]           tag[-PNT_ABS_]
```

Entrada 4.2:

```
Eu debuguei o sistema.
```

Saída 4.2:

```
Eu           lemma[eu]          tag[PERS_M/F_1S_NOM_]
debuguei     lemma[]            tag[ADV_]
o            lemma[o]           tag[DET_M_S_]
sistema      lemma[sistema]     tag[N_M_S_]
.            lemma[.]           tag[-PNT_ABS_]
```

Entrada 4.3:

```
Ele debugará o sistema.
```

Saída 4.3:

```
Ele          lemma[ele]        tag[PERS_M_3S_NOM_]
debugará     lemma[]           tag[V_PR_3S_IND_VFIN_]
o            lemma[o]          tag[DET_M_S_]
sistema      lemma[sistema]    tag[N_M_S_]
.            lemma[.]          tag[-PNT_ABS_]
```

5. **“mandatório”**

Entrada 5.1:

```
Este campo é mandatório.
```

Saída 5.1:

```
Este         lemma[este]       tag[DET_M_S_]
campo        lemma[campo]      tag[N_M_S_]
é            lemma[ser]        tag[V_PR_3S_IND_VFIN_]
mandatório   lemma[mandatório] tag[ADJ_M_S_]
.            lemma[.]          tag[-PNT_ABS_]
```

Entrada 5.2:

```
Estes campos são mandatórios.
```

Saída 5.2:

```
Estes        lemma[este]       tag[DET_M_P_]
campos       lemma[campo]      tag[N_M_P_]
são          lemma[ser]        tag[V_PR_3P_IND_VFIN_]
mandatórios  lemma[]           tag[N_M_P_]
.            lemma[.]          tag[-PNT_ABS_]
```

6. **“randômico”**

Entrada 6.1:

```
O dado é randômico.
```

Saída 6.1:

```
O            lemma[o]          tag[DET_M_S_]
dado         lemma[dado]       tag[N_M_S_]
é            lemma[ser]        tag[V_PR_3S_IND_VFIN_]
randômico    lemma[]           tag[N_M_S_]
.            lemma[.]          tag[-PNT_ABS_]
```

Entrada 6.2:

```
Os dados são randômicos.
```

Saída 6.2:

```
Os           lemma[o]          tag[DET_M_P_]
dados        lemma[dado]       tag[N_M_P_]
são          lemma[ser]        tag[V_PR_3P_IND_VFIN_]
randômicos   lemma[]           tag[V_M_P_PCP_]
.            lemma[.]          tag[-PNT_ABS_]
```

Entrada 6.3:

```
São variáveis randômicas.
```

Saída 6.3:

```
São          lemma[ser]        tag[V_PR_3P_IND_VFIN_]
variáveis    lemma[variável]   tag[ADJ_F_P_]
randômicas   lemma[]           tag[N_F_P_]
.            lemma[.]          tag[-PNT_ABS_]
```

8.3 Anexo C - Etiquetas

8.3.1 Etiquetador morfológico - Português

Etiqueta	Significado
_AD	Artigo definido
_AI	Artigo indefinido
_AJ	Adjetivo
_AP	Particípio
_AV	Advérbio
_CC	Conjunção coordenativa
_CS	Conjunção subordinativa
_IN	Interjeição
_NC	Numeral (cardinal)
_NO	Numeral (ordinal)
_PS	Pronome possessivo
_PD	Pronome demonstrativo
_PI	Pronome indefinido
_PL	Pronome relativo
_PP	Pronome pessoal
_PN	Pontuação
_PR	Preposição
_SU	Substantivo
_VA	Verbo auxiliar
_VB	Verbo
_VG	Vírgula, parênteses, barra

Tabela C.1: Etiquetas usadas pela ferramenta FORMA[FM] para etiquetagem de textos em Português.

8.3.2 Etiquetador morfológico - Inglês

Etiqueta	Significado
CC	Coordinating conjunction (e.g. *but, and*)
CD	Number (cardinal)
DT	Determiner (e.g. *every, much*)
EX	Existencial *there*
FW	Foreign word
IN	Preposition or Subordinating Conjunction
JJ	Adjective
JJR	Adjective (comparative)
JJS	Adjective (superlative)
LS	List item marker (e.g. *A A. B B. C C.*)
MD	Modal (e.g. *can, may*)
NN	Noun (singular or mass)
NNP	Noun (proper, singular)
NNPS	Noun (proper, plural)
NNS	Noun (plural)
PDT	Pre-determiner (e.g. *all, both* ... when they precede an article)
POS	Possessive Ending (*'s*)
PRP	Personal Pronoun
PRP$	Possessive Pronoun
RB	Adverb
RBR	Adverb (comparative)
RBS	Adverb (superlative)
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb (base form)
VBD	Verb, past tense
VBG	Verb, gerund or persent participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner (e.g. *which*)
WP	Wh-pronoun (e.g. *what, who*)
WP$	Possessive wh-pronoun (e.g. *whose*)
WRB	Wh-adverb (e.g. *where, why*
#	#
$	$
”	”
(	(
)	)
,	,
.	.
:	:
“	“

Tabela C.2: Etiquetas usadas pela ferramenta CoreNLP[SCN].

8.4 Anexo D - Testes realizados com o CATA

Para avaliar a eficácia do CATA, foram realizados testes sobre um conjunto de doze textos acadêmicos de Computação em Português. Tratam-se de 12 monografias de conclusão do curso de Ciência da Computação (retiradas de [CEF]).

Total de problemas de estilo encontrados	Verdadeiros positivos	Falsos positivos
271	236	35
100%	87.08%	12.92%

Tabela D.1: Total de problemas de estilo encontrados pelo Sistema CATA.

A tabela a seguir fornece algumas medidas de dispersão para o número de problemas encontrados por texto: o número mínimo, o número máximo, a média de problemas de estilo encontrados por texto e o desvio padrão.

Mínimo	Máximo	Média	Desvio padrão	Total
2	52	22.58	13.26	271

Tabela D.2: Medidas de dispersão: problemas por texto (considerando verdadeiros e falsos positivos). Interpretação dos dados: no texto com menor número de problemas, foram encontrados dois termos problemáticos, a média é o total de problemas por texto: 271 (total) / 12 (número de textos), etc.

A Tabela D.3 apresenta as mesmas medidas de dispersão, mas, dessa vez, são considerados apenas os verdadeiros postivos: o número mínimo, o número máximo, a média de verdadeiros positivos encontrados por texto e o desvio padrão.

Mínimo	Máximo	Média	Desvio padrão	Total
1	44	19.67	11.58	236

Tabela D.3: Medidas de dispersão: problemas por texto (considerando apenas verdadeiros positivos).

E para o caso dos falsos positivos:

Mínimo	Máximo	Média	Desvio padrão	Total
0	15	2.92	4.46	35

Tabela D.4: Medidas de dispersão: falsos positivos por texto.

Problema de estilo	Total	Falsos positivos	Verdadeiros positivos
design	29	10	19
thread	26	0	26
site	23	1	22
parser	20	2	18
suportar	19	0	19
retornar	19	1	18
framework	18	1	17
rodar	15	0	15
softwares	11	0	11
checar	8	0	8
log	8	0	8
bug	6	0	6
corretude	6	0	6
patch	6	0	6
template	6	0	6
performance	6	1	5
layout	6	2	4
assumir	6	3	3
overhead	5	0	5
paper	5	5	0
grid	4	2	2
deadlock	3	0	3
string	3	0	3
kernel	3	1	2
peer-to-peer	2	1	1
cluster	1	0	1
I/O	1	0	1
round-robin	1	0	1
audio	1	1	0
eXtreme Programming	1	1	0
failure	1	1	0
Pervasive Computing	1	1	0
request	1	1	0

Tabela D.5: Distribuição dos problemas encontrados.

A maioria dos falsos positivos ocorreu porque o Sistema detectou termos problemáticos em títulos de obras (nas seções de Bibliografia): dos 35 falsos positivos, 22 foram casados com partes de títulos de obras citadas como referências nas monografias.

Problema de estilo	Número de textos em que ocorreu
retornar	7
softwares	6
framework	5
rodar	5
suportar	5
thread	5
corretude	4
site	4
checar	3
deadlock	3
design	3
overhead	3
parser	3
performance	3
template	3
assumir	2
string	2
cluster	1
bug	1
grid	1
I/O	1
kernel	1
layout	1
log	1
patch	1
peer-to-peer	1
round-robin	1

Tabela D.6: Distribuição dos problemas por texto (considerando apenas os verdadeiros positivos): se houve apenas falsos positivos para um termo num texto, então este texto não entrou na contagem de textos em que tal problema ocorre.

9 Parte subjetiva

Nesta seção da monografia, os alunos têm a oportunidade de tecer comentários subjetivos acerca da experiência do desenvolvimento do trabalho de conclusão de curso.

9.1 Desafios encontrados na elaboração do trabalho

Certamente não estou sozinha quando afirmo que o maior dos desafios é conciliar o Bacharelado em Ciência da Computação com outras atividades e com a vida pessoal: tive que dividir meu tempo entre o trabalho de conclusão de curso, as demais disciplinas, uma monitoria e o estágio.

Com relação à implementação do software em particular, foi bastante desafiador trabalhar em todos os aspectos do desenvolvimento de um sistema, sobretudo considerando a falta de conhecimento e experiência com as tecnologias utilizadas. Certamente foi um processo enriquecedor, mas penso que talvez teria sido mais agradável e mais produtivo fazer o trabalho em grupo (em dupla ou até mesmo em trio).

9.2 Funcionalidades, funcionalidades, funcionalidades...

Fiquei satisfeita com o resultado obtido neste trabalho: o sistema produzido funciona bem e ficou bastante bonito e agradável de usar. Contudo, quando trata-se de um sistema "de verdade" - não apenas um EP[13] a ser entregue - é difícil dizer "está pronto": sempre é possível melhorar e acrescentar novas funcionalidades interessantes. O caso do CATA não é diferente - há muito para ser feito.

O Sistema CATA aceita, como entrada, textos acadêmicos de Computação em arquivos de texto sem formatação (.txt) ou em arquivos "Portable Document Format" (*.pdf*). Um passo natural seria possibilitar outros tipos de arquivo como entrada: arquivos do *Microsoft Word* (*.doc* e *.docx*), arquivos de formato aberto (*.odt*) e sobretudo arquivos LaTeX (*.tex*) - pois além de serem muito usados por alunos de Computação também são arquivos cujo texto é "difícil de extrair manualmente": é preciso remover marcações referentes a formatação (as "*tags*").

Outra melhoria que poderia ser realizada é a análise dos textos em si: melhorar a eficácia e a eficiência do processo de Lematização, por exemplo. Além disso, possibilitar a detecção de outros tipos de problemas de estilo, como repetição excessiva de termos e inversão da ordem sintática natural.

Outras funcionalidades, digamos, menos críticas, que poderiam ser implementadas tem a ver com a "personalização" do sistema. No momento, o Sistema aplica as chamadas "regras padrões" aos textos e, se o usuário deseja aplicar todas as regras cadastradas ou usar apenas as regras de determinadas referências bibliográficas, ele deve usar um mecanismo que chamei de "Verificação Avançada". Contudo, as opções selecionadas para a tal verificação avançada não são persistidas. Essas opções poderiam ficar salvas para o usuário.

Enfim, há muitas possibilidades para o Sistema. No repositório[GH] do trabalho, disponibilizei um arquivo "TODO.txt" com essas e outras sugestões para expandir o sistema.

9.3 Paralelo entre o trabalho de formatura e o curso de Ciência da Computação

O curso de Bacharelado em Ciência da Computação do Instituto de Matemática e Estatística da USP é um curso de caráter bastante teórico. As matérias matemáticas e teóricas, embora bastante

[13] **EP** é a sigla usada (no curso de Ciência da Computação do IME) para **Exercício Programa** - um trabalho que exige a entrega de um programa para resolver um determinado problema.

desprezadas e execradas pelos alunos em geral, foram bastante importantes para o desenvolvimento deste trabalho, assim como as outras de enfoque mais prático. Portanto, acredito que todas as disciplinas da grade do Curso tiveram seu papel na minha formação acadêmica e profissional e no desenvolvimento deste trabalho. Mesmo assim, listo abaixo as matérias que foram aplicadas mais diretamente na implementação do Sistema CATA e na elaboração desta Monografia.

- **MAT011 - Cálculo Diferencial e Integral I, MAT0221 - Cálculo Diferencial e Integral IV**
  Foram fundamentais para a análise dos algoritmos.

- **MAE0121 - Introdução à Probabilidade e à Estatística I, MAE0228 - Noções de Probabilidade e Processos Estocásticos**
  Estas disciplinas de Estatística foram essenciais ao entendimento das soluções em Processamento de Linguagens Naturais e Aprendizagem de Máquina.

- **MAC0110 - Introdução à Computação, MAC0122 - Princípios de Desenvolvimento de Algoritmos**
  São cursos introdutórios de Computação e serviram de base para todo o conhecimento que adquiri posteriormente e que foi amplamente empregado neste trabalho.

- **MAC323 - Estruturas de Dados, MAC0328 - Algoritmos em Grafos, MAC0338 - Análise de Algoritmos e MAC414 - Linguagens Formais e Autômatos**
  Mostraram-se úteis para a compreensão, análise e implementação dos algoritmos e soluções pesquisadas.

- **MAC0316 - Conceitos Fundamentais de Linguagens de Programação, MAC434 - Programação Funcional Contemporânea**
  Estas disciplinas envolveram Programação Funcional e, embora não estejam diretamente relacionadas a este trabalho, foram úteis, pois acredito que, após cursá-las, tornei-me uma programadora melhor.

- **MAC0438 - Programação Concorrente, MAC0211 - Laboratório de Programação I, MAC0242 - Laboratório de Programação II**
  Cursos com viés mais prático, bastante aplicados aqui. Vale mencionar que "Laboratório de Programação II" foi a disciplina em que criei meu primeiro programa com interação através de *interface* gráfica.

- **MAC0426 - Sistemas de Bancos De Dados**
  O Sistema CATA precisa persistir dados sobre as sugestões de estilo e sobre seus usuários. Naturalmente, conceitos de Bancos de Dados foram diretamente aplicados.

- **MAC0441 - Programação Orientada a Objetos**
  Quando comecei a implementação do Sistema CATA, no início do ano de 2011, não tinha muitos conhecimentos de Padrões de Projeto (*Design Patterns*). Quando tomei contato com essas práticas, na disciplina de MAC0441, no segundo semestre de 2011, tentei aplicá-las neste trabalho, refatorando e remodelando o Sistema.

- **CTR0716 - Introdução à Iconomia**[14]
  Esta disciplina, cursada na Escola de Comunicação e Artes, mostrou-se bastante útil para este trabalho: entre os temas abordados em "CTR0716", estavam Pierre Levy e a Inteligência Coletiva. Além disso, a "Introdução à Iconomia" foi uma matéria que estimulou minha criatividade na elaboração da apresentação e do pôster.

[14] Está certo, é **I**conomia, com **I** mesmo!

- **MAC0342 - Laboratório de Programação Extrema**
  Este foi o curso que mais contribuiu para o desenvolvimento do meu trabalho de Conclusão de Curso, não só pelo fato de que o projeto de que participei usava as mesmas tecnologias escolhidas para desenvolver o Sistema CATA, mas sobretudo pelo contato com a metodologia ágil em si. Depois de cursar esta disciplina, tornei-me muito mais consciente com relação a aspectos importantíssimos do desenvolvimento de *software* em geral: planejamento, trabalho em equipe e testes, e, sobretudo, o foco no ser humano, pautando atitudes em valores que devem nortear qualquer tipo de trabalho, como respeito, comunicação e coragem.

9.4 Comentários finais

Gostaria de agradecer ao Professor Marco Aurélio Gerosa pela orientação e criação do nome *CATA: Collaborative Academic Text Advisor*. Também gostaria de agradecer àqueles professores do IME que fizeram valer a pena acordar cedo e enfrentar o transporte coletivo lotado para assistir suas aulas. Finalmente, agradeço aos meus familiares e amigos que tanto ouviram falar sobre este trabalho.

Pelos complexos problemas a serem resolvidos (em Processamento de Linguagens Naturais e em Aprendizagem Computacional), por ter exigido tanto da minha criatividade e pelo fato de eu ter desenvolvido sozinha, este Trabalho de Conclusão de Curso representou um grande desafio. Assim como o BCC (como é mais conhecido o Bacharelado em Ciência da Computação) também representou um imenso desafio. Lembro-me de uma palestra que assisti, logo que entrei na USP, sobre o BCC, ministrada por um ex-aluno do IME - a apresentação terminava com uma citação de Nietzsche, de *Crepúsculo dos ídolos*:

> "Aquilo que não nos mata, torna-nos mais fortes."

Referências

[AC] AHO, A. V.; CORASICK, M. J.;
Efficient String Matching: an Aid to Bibliographic Search,
In Communications of the ACM 18, págs. 333–340, 1975.
Esta referência é o artigo publicado em 1975 por Aho e Corasick, descrevendo o Algoritmo Aho-Corasick, o qual foi usado neste trabalho para a busca dos problemas de estilo. Os algoritmos, definições e demonstrações incluídos na Seção 4.4.1 foram retirados, com poucas alterações, desta referência.

[CD] TOUTANOVA, Kristina; KLEIN, Dan; MANNING, Christopher; SINGER, Yoram;
Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network,
In Proceedings of HLT-NAACL, págs. 252-259, 2003.

[CEF] MAC499-2010,
Trabalho de Formatura Supervisionado 2010,
http://www.ime.usp.br/~cef/mac499-10/monografias/
Neste sítio estão as monografias de conclusão de curso produzidas pelos alunos de Ciência da Computação do IME EM 2010. Dentre eLas, doze foram usadas para realizar testes com o Sistema CATA.

[CEM] BATISTA, Thiago M.,
Corretor Gramatical para o Emacs,
Universidade de São Paulo, 2010.
http://www.ime.usp.br/~cef/mac499-10/monografias/thiago/monografia/monografia.pdf
Esta referência é uma monografia do Trabalho de Formatura Supervisionado do Curso de Bacharelado em Ciência da Computação do IME-USP. Foi uma das primeiras leituras realizadas para elaborar o Sistema CATA e serviu muito bem como texto introdutório para a área de "Processamento de Linguagens Naturais" e para a arquitetura do CoGrOO.

[CF] Linguateca,
CETENFolha,
http://www.linguateca.pt/CETENFolha/

[CG] SourceForge,
CoGrOO - Corretor Gramatical acoplável ao LibreOffice,
http://cogroo.sourceforge.net/

[CGO] KINOSHITA J.; SALVADOR L.N. Salvador; MENEZES, C.E.D.;
CoGrOO – Um Corretor Gramatical para a língua portuguesa, acoplável ao OpenOffice
Universidade de São Paulo, 2005.

[ES] PRESSMAN, Roger S.
Engenharia de Software,
AMGH, 6 ed., pág. 443, 2010.

[F94] **Folha94**
http://www.inf.pucrs.br/~gonzalez/tr+/folha94.htm

[FM] GONZALEZ, M.; LIMA, V. L. S. de; LIMA, J. V. de.;
Tools for Nominalization: an Alternative for Lexical Normalization
In Workshop on Comp. Proc. of the Portuguese Lang. - Written and Spoken 7; PROPOR, 2006.
A implementação do Lematizador para a língua portuguesa baseou-se fortemente neste trabalho. Algumas definições, o diagrama e os passos do algoritmo, foram retirados, com alterações, dessa referência.

[GE] Guia do Estudante,
Vidas Secas - Trechos comentados
http://guiadoestudante.abril.com.br/estude/literatura/materia_417671.shtml

[IC] SEGARAN, Toby,
Programando a Inteligência Coletiva,
Alta Books, 2008.

[KE] MATSUO, Yutaka; ISHIZUKA, Mitsuru;
Keyword Extraction from a Single Document using Word Co-ocurrence Statistical Information,
In International Journal on Artificial Intelligence Tools,
Vol.13, Nº 1, págs. 157-169, 2004
Este texto descreve um algoritmo para extrair palavras-chave de textos sem a necessidade de um corpus para treinamento.

[KEA] KEA,
Keyword Extraction Algorithm,
http://www.nzdl.org/Kea/

[MA] OLENA, Medelyan,
Human-competitive automatic topic indexing,
The University Of Waikato, 2009.
Esta referência, embora não esteja tão diretamente relacionada ao tema principal desta monografia, foi utilizada como inspiração na "Introdução" e para definir a estrutura do trabalho.

[ME] TOUTANOVA, Kristina; MANNING, Christopher D.;
Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger
In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, págs. 63-70, 2000.

[MF] Folha de São Paulo,
Manual de Redação,
Publifolha, 16 ed., 2010.

[MR] Microsoft Research,
Natural Language Processing,
http://research.microsoft.com/en-us/groups/nlp/
Texto utilizado na contextualização do trabalho, na seção de "Processamento de Linguagens Naturais". Esta referência é o sítio sobre pesquisas em PLN da "Microsoft Research" - contém descrições sobre projetos que estão sendo desenvolvidos assim como os já concluídos.

[PACA] PACA,
MAC0499-11 - Projetos do Prof. Marco Gerosa,
http://paca.ime.usp.br/mod/forum/discuss.php?d=15893
A motivação do trabalho, apresentada na "Introdução" e proposta pelo orientador Prof. Dr. Marco Aurélio Gerosa, foi retirada, com pequenas modificações, desta referência.

[PM] BASALO, Ana L. D. F.,
MAC0499 - Proposta da Monografia,
http://www.linux.ime.usp.br/~albasalo/mac499/proposta.html
Foi usado, com algumas alterações, o resumo que consta nesta referência, que é a proposta desta monografia e tem a mesma autoria.

[RA] VIERA, Angel F. G.; VIRGIL, Johnny;
Uma revisão dos algoritmos de radicalização em língua portuguesa,
http://informationr.net/ir/12-3/paper315.html

[SCN] The Stanford Natural Language Processing Group,
CoreNLP
http://nlp.stanford.edu/software/corenlp.shtml

[SEP] Stanford Encyclopedia Of Philosophy,
The Turing Test,
http://plato.stanford.edu/entries/turing-test/
Referência usada na composição do trecho sobre história do Processamento de Linguagens Naturais.

[SSE] Stanford School Of Engineering,
artificial intelligence | natural language processing,
http://see.stanford.edu/see/courseinfo.aspx?coll=63480b48-8819-4efd-8412-263f1a47
http://see.stanford.edu/materials/ainlpcs224n/transcripts/
NaturalLanguageProcessing-Lecture01.html
Esta referência é o curso de "Natural language processing" da Stanford School Of Engineering: disponibiliza o conteúdo das aulas e foi usado, sobretudo, na contextualização histórica da seção de "Processamento de Linguagens Naturais".

[VS] RAMOS, Graciliano,
Vidas Secas
Record, pág. 37, 2007.

[WK1] Wikipedia,
Natural language processing,
http://en.wikipedia.org/wiki/Natural_language_processing

[WK2] Wikipedia,
Natural language,
http://en.wikipedia.org/wiki/Natural_language

[WK3] Wikipedia,
Turing test,
http://en.wikipedia.org/wiki/Turing_test

[WK4] Wikipedia,
ELIZA,
http://en.wikipedia.org/wiki/ELIZA

[WK5] Wikipedia,
Computer Power and Human Reason,
http://en.wikipedia.org/wiki/Computer_Power_and_Human_Reason

[WK6] Wikipedia,
Watson (computer),
http://en.wikipedia.org/wiki/Watson_(computer)

[WK7] Wikipedia,
Aho-Corasick string matching algorithm,
http://en.wikipedia.org/wiki/Aho-Corasick_string_matching_algorithm
Referência usada para apresentar o algoritmo Aho-Corasick, na seção de "Fundamentação Teórica".

[WK8] Wikipedia,
Model-view-controller,
http://en.wikipedia.org/wiki/Model-view-controller

[WP1] Wikipedia,
Processamento de linguagem natural,
http://pt.wikipedia.org/wiki/Processamento_de_linguagem_natural

[WP2] Wikipedia,
Língua natural,
http://pt.wikipedia.org/wiki/Línguas_naturais

[WP3] Wikipedia,
Inteligência coletiva,
http://pt.wikipedia.org/wiki/Inteligência_coletiva
A citação de Pierre Lévy, transcrita na seção de "Inteligência Coletiva", foi retirada desta referência.

[DU] Duke University,
Scientific Writing Web Resource,
http://cgi.duke.edu/web/sciwriting/index.php
Este recurso sobre escrita científica e acadêmica, oferecido pela Duke University, é bastante completo e didático. Foi utilizado sobretudo na seção de "Fundamentação Teórica", para os exemplos de problemas de estilo. Além disso, várias das regras padrões do Sistema CATA foram retiradas desta referência.

[FK1] KON, Fabio,
Tradução de Textos de Computação e Informática para o Português
http://www.ime.usp.br/~kon/ResearchStudents/traducao.html
Uma das principais referências para este trabalho, é uma compilação dos erros de escrita mais comuns cometidos pelos alunos de Computação no que se refere a terminologia e tradução.

[FK2] KON, Fabio,
Dicas Sobre Escrita em Inglês,
http://www.ime.usp.br/~kon/ResearchStudents/dicasIngles.html
Uma das principais referências para este trabalho, como o próprio título denuncia, apresenta dicas para escrita de textos acadêmicos em Inglês.

[GH] Github,
albasalo/CATA,
https://github.com/albasalo/CATA
Este é o repositório no qual foi hospedado todo o código-fonte deste trabalho.