Utilização de informações lexicais extraídas automaticamente de *corpora* na análise sintática computacional do português

Using lexical information automatically extracted from corpora in the computational parsing of Portuguese

Leonel Figueiredo de Alencar
Universidade Federal do Ceará

Resumo

No desenvolvimento de analisadores sintáticos profundos para textos irrestritos, a principal dificuldade a ser vencida é a modelação do léxico. Tradicionalmente, duas estratégias têm sido usadas para lidar com a informação lexical na análise sintática automática: a compilação de milhares de entradas lexicais ou a formulação de centenas de regras morfológicas. Devido aos processos produtivos de formação de palavras, aos nomes próprios ou a grafias não padrão, a primeira estratégia, que subjaz aos analisadores do português do Brasil (PB) livremente descarregáveis da Internet, não é robusta. A última estratégia, por sua vez, constitui tarefa não trivial de engenharia do conhecimento, consumindo muito tempo. No momento, o PB não dispõe de um analisador sintático de ampla cobertura licenciado como *software* livre. Visando ao preenchimento o mais rápido possível dessa lacuna, argumentamos neste artigo que uma solução bem menos custosa e muito mais eficiente para o gargalo lexical consiste em simplesmente reaproveitar, como componente lexical do processamento sintático profundo, etiquetadores morfossintáticos livremente disponíveis.

Além disso, graças à ampla e gratuita disponibilidade de *corpora* morfossintaticamente anotados do PB e eficientes pacotes de aprendizado de máquina, a construção de etiquetadores de alta acurácia adicionais tornou-se uma tarefa que quase não demanda esforço. A fim de integrar facilmente o *output* de etiquetadores de diferentes arquiteturas em *parsers* tabulares de gramáticas livres de contexto compilados por meio do Natural Language Toolkit (NLTK), desenvolvemos um módulo em Python denominado ALEXP. Pelo que sabemos, o ALEXP é o primeiro *software* livre especialmente otimizado para o processamento do português a realizar essa tarefa. A funcionalidade da ferramenta é descrita por meio de protótipos de gramática do PB aplicados na análise de sentenças do mundo real, com resultados bastante promissores.

Palavras-chave

Linguística computacional, Processamento automático da linguagem natural, Etiquetagem morfossintática, Etiquetador morfossintático, Análise sintática automática, Gramática livre de contexto, Processamento computacional do português, Aquisição de conhecimento lexical, Aprendizado de máquina.

Abstract

Lexicon modeling is the main difficulty to overcome when building deep syntactic parsers for unrestricted text. Traditionally, two strategies have been used for tackling lexical information in the domain of unrestricted syntactic parsing: compiling thousands of lexical entries or formulating hundreds of morphological rules. Due to productive word-formation processes, proper names, and non-standard spellings, the former strategy, resorted to by freely downloadable parsers for Brazilian Portuguese (BP), is not robust. On the other hand, deploying the latter is a time-intensive and non-trivial knowledge engineering task. At present, there is no open-source licensed wide-coverage parser for BP. Aiming at filling this gap as soon as possible, we argue in this paper that a much less expensive and much more efficient solution to the lexicon

bottleneck in parsing is to simply reuse freely available morphosyntactic taggers as the system's lexical analyzer. Besides, thanks to the free and broad availability of POS-tagged *corpora* for BP and efficient machine learning packages, building additional high accurate taggers has become an almost effortless task. In order to easily integrate the output of taggers constructed in different architectures into context-free grammar chart parsers compiled with the Natural Language Toolkit (NLTK), we have developed a Python module named ALEXP. To the best of our knowledge, this is the first free software specially optimized for processing Portuguese to accomplish such a task. The tool's functionality is described by means of BP grammar prototypes applied to parsing real-world sentences, with very promising results.

Keywords

Computational linguistics, Natural language processing, Morphosyntactic tagging, POS tagging, Part-of-speech tagging, POS tagger, Syntactic parsing, Context-free grammar, Computational processing of Portuguese, Lexical knowledge acquisition, Machine learning.

1. Introdução

Este trabalho tematiza a aprendizagem de informações lexicais pelo computador a partir de textos e a utilização desses dados na análise sintática computacional, inserindo-se na confluência da gramática gerativa, processamento automático da linguagem natural (doravante PLN), linguística de *corpus* e inteligência artificial (IA). A questão central sobre a qual nos debruçamos resume-se da seguinte forma: Como integrar em um analisador sintático automático (*parser*) informações lexicais extraídas automaticamente de *corpora*?

Constitui objetivo da IA simular as habilidades cognitivas humanas, entre as quais a capacidade de aprender figura como uma das mais características. Logo, a aprendizagem ou aquisição automática de conhecimentos é uma das áreas de investigação mais importantes da IA.

Diante das exigências do processamento computacional da linguagem, diferentes abordagens da gramática gerativa oferecem modelos computacionalmente implementáveis do conhecimento sintático e lexical, permitindo a construção de analisadores automáticos de sentenças. Tradicionalmente, na gramática gerativa, esses modelos eram construídos manualmente pelos linguistas. Graças à subárea da IA conhecida como aprendizado de máquina (*machine learning*), cuja aplicação no PLN se disseminou na última década (JURAFSKY; MARTIN, 2009, p. 47), tornou-se não só possível, mas relativamente fácil a modelação computacional automática da sintaxe e do léxico indutivamente, a partir de *corpora* (LEMNITZER; WAGNER, 2004; NIVRE, 2010).

Pré-requisito para isso, contudo, é a existência de *corpora* representativos da língua cuja gramática pretendemos modelar computacionalmente. Na situação ideal, esses *corpora* encontram-se enriquecidos de anotações, por exemplo, sobre as classes de palavras.[1] Embora a linguística computacional ainda não tenha alcançado, no Brasil, o alto grau de desenvolvimento de países como EUA e Alemanha (para citar apenas dois dos exemplos mais proeminentes),[2] as

pesquisas em linguística de *corpus*, entre nós, avançaram muito na última década (TAGNIN; VALE, 2008). Fruto desse desenvolvimento é a existência de vastos *corpora* anotados, que podem ser utilizados, por exemplo, para a extração de informações lexicais.

O Natural Language Toolkit (NLTK) é uma biblioteca na linguagem de programação Python que funciona como uma caixa de ferramentas computacionais voltadas tanto para o processamento de *corpora* quanto para a análise computacional da linguagem em vários níveis, incluindo o sintático (BIRD; KLEIN; LOPER, 2009, 2011; PERKINS, 2010). Trata-se do mais didático, amigável e abrangente projeto de *software* livre na área de PLN. Outra vantagem do NLTK é a enorme flexibilidade que permite ao usuário, mesmo iniciante em programação, desenvolver seus próprios programas em Python, dada a incomparável versatilidade dessa linguagem para o processamento de textos (*text processing*),[3] e, graças ao caráter extensível e "plugável" de Python (CHUN, 2006, p. 8, 10), usá-los concomitantemente ou em substituição a determinados recursos do NLTK.

Na elaboração de um analisador sintático automático de ampla cobertura no âmbito do NLTK, a exemplo do que ocorre noutros sistemas de desenvolvimento de gramática, o componente de maior volume informacional é o léxico. Por conta disso, erros de natureza lexical respondem pela maior parte dos fracassos de muitos *parsers* na análise de textos reais, como relatado para o sistema MedScan (NOVICHKOVA; EGOROV; DARASELIA, 2003, p. 1705).[4]

Uma modelação totalmente manual do léxico demandaria uma quantidade impraticável de homens-horas. Tradicionalmente, duas estratégias têm sido usadas resolver esse problema no âmbito da análise sintática automática: a compilação de milhares de entradas lexicais ou a formulação de centenas de regras morfológicas. Diante dos desafios do processamento sintático profundo de textos irrestritos, a primeira estratégia de modelação do léxico não é robusta, uma vez que é incapaz de lidar com neologismos e variantes ortográficas não dicionarizados. A segunda estratégia, por outro lado, demanda muito tempo e sofisticada engenharia do conhecimento linguístico (BEESLEY; KARTTUNEN, 2003, p. 279).

Ante esse impasse, o NLTK, felizmente, disponibiliza técnicas de aprendizado de máquina para a construção de etiquetadores morfossintáticos. Como mostraremos neste trabalho, o *output* dessas ferramentas pode ser integrado ao componente sintático de uma gramática de estrutura sintagmática para a construção de um *parser*. Por outro lado, graças à extensibilidade e plugabilidade

de Python e à modularidade do NLTK, também o *output* de etiquetadores construídos noutras linguagens de programação pode constituir *input* para o processamento não só por um *parser*, mas por outros módulos dessa biblioteca, por exemplo, um *chunker*, um reconhecedor de entidades nomeadas (NER) ou um sistema de diálogo homem-máquina.

Neste trabalho, apresentamos uma proposta para fácil integração de etiquetadores de diferentes arquiteturas na análise sintática automática profunda em moldes gerativos. O procedimento é primeiramente exemplificado com um etiquetador de Brill construído por meio do NLTK e com um etiquetador HMM construído por meio do HunPos (HALÁCSY; KORNAI; ORAVECZ, 2007), um *software* de aprendizado de máquina implementado em Ocaml, distribuído sob licença livre, para o qual o NLTK oferece uma interface amigável.[5] O desempenho desses etiquetadores como analisadores lexicais de diferentes parsers é comparado em seguida com o do LX-Tagger (BRANCO; SILVA, 2004), disponibilizado livremente para pesquisa,[6] para o qual construímos uma interface do NLTK.

Para utilizar etiquetadores como analisadores lexicais, desenvolvemos o ALEXP, uma ferramenta de análise sintática automática implementada em Python. Essa ferramenta permite construir *parsers* de diferentes algoritmos com base em uma gramática de estrutura sintagmática e em um etiquetador morfossintático especificados pelo usuário. O ALEXP possibilita, portanto, reutilizar, no processamento computacional da sintaxe, ferramentas e recursos livremente disponíveis no âmbito da linguística computacional e da linguística de *corpus*. Esse programa torna dispensável, no desenvolvimento de um *parser*, a custosa elaboração de um componente morfológico para análise de palavras (nos moldes do que propõem, por exemplo, Beesley e Karttunen (2003) e Roark e Sproat (2007)) ou a compilação prévia de entradas lexicais, estratégia não robusta adotada, por exemplo, nos *parsers* de Martins, Nunes e Hasegawa (2003), Almeida *et al*. (2003) e Othero (2006).

O ALEXP facilita enormemente, portanto, a construção de *parsers* de gramáticas de estrutura sintagmática. Pelo que sabemos, trata-se da primeira abordagem a integrar a etiquetagem morfossintática à análise sintática profunda no NLTK. Desconhecemos outro sistema de *parsing* de linguagem natural distribuído sob licença livre que recorra a essa estratégia de modelação do conhecimento lexical, tenha a mesma flexibilidade na integração de

etiquetadores de diferentes arquiteturas e seja otimizado para o trabalho com a língua portuguesa.

Para Hausser (2000, p. 326), a utilização de um etiquetador morfossintático estocástico como analisador lexical instancia o que denomina *smart solution* no âmbito do PLN, tipo de solução cujas desvantagens não compensariam, segundo ele, a robustez e relativa facilidade de implementação, o que o leva a preferir uma abordagem baseada em regras. Ele alega que etiquetadores estocásticos não produzem resultados suficientemente precisos para funcionarem como *input* de um parser baseado em regras.

No entanto, o desenvolvimento do PLN na última década veio contrariar Hausser (2000). Como ressaltam Ljunglöf e Wirén (2010, p. 84), os *parsers* de ampla cobertura de gramáticas desenvolvidas no âmbito das principais teorias sintáticas computacionalmente implementadas, como a HPSG e a LFG, têm recorrido ao pré-processamento do *input* por meio de componentes estatísticos, por exemplo, etiquetadores morfossintáticos. Nossa proposta de utilizar um etiquetador morfossintático como analisador lexical de *parsers* para textos irrestritos se insere nessa linha, comparando-se, até um certo ponto, a algumas abordagens recentes no *parsing* do português. Por exemplo, no analisador sintático de que tratam Silva, Branco e Gonçalves (2010), o LX-Parser, é possível fornecer como *input* uma sentença pré-processada pelo etiquetador morfossintático LX-Tagger.[7] No analisador sintático superficial do português do Brasil (doravante PB) de Contier, Padovani e José Neto (2010), por sua vez, um "Identificador Morfológico" atribui etiquetas de classe de palavra a uma sentença toquenizada, etiquetas essas extraídas de um *corpus*. A diferença em relação à nossa abordagem é a maior flexibilidade do ALEXP, que não depende de um etiquetador específico, permitindo o reaproveitamento de etiquetadores externos disponíveis em diferentes arquiteturas. Os *parsers* de Martins, Nunes e Hasegawa (2003), Almeida *et al*. (2003) e Othero (2006), pelo contrário, em vez de utilizar etiquetadores existentes, modelam a informação lexical sobretudo ou exclusivamente por meio da listagem de entradas lexicais.

O presente trabalho insere-se na atual tendência de desenvolvimento de ferramentas robustas de PLN de forma rápida e sem muito esforço, graças à disponibilização gratuita de pacotes de aprendizado de máquina e *corpora* anotados, tendência essa exemplificada, para o português, de forma prototípica por Branco e Silva (2004), Silva, Branco e Gonçalves (2010), Silva *et al.* (2010) e Garcia e Gamallo (2010).[8]

No momento, o NLTK dispõe de gramáticas de cobertura sintática relativamente ampla para o inglês. Essas gramáticas, contudo, têm o léxico limitado a um domínio restrito. Há também minigramáticas (*toy grammars*) de línguas como espanhol e basco. Apesar dessas limitações, essas gramáticas constituem valioso instrumento para estudantes de sintaxe formal ou linguística computacional, haja vista as facilidades do NLTK para a construção de *parsers* e a análise automática de sentenças. Para o português, contudo, o NLTK ainda não oferece esse tipo de recurso.

Por outro lado, desconhecemos a existência de *software* livre (de código aberto) capaz de processar sintaticamente de forma profunda textos irrestritos (*unrestricted text*) na variedade brasileira da língua. O analisador sintático de Othero (2006), por exemplo, ao que sabemos o único *parser* do PB que talvez se enquadre na categoria de *software* livre, contempla apenas as formas da terceira pessoa no presente do indicativo de um subconjunto de verbos dicionarizados.

Visando a contribuir para preencher essa lacuna, construímos, conforme relatado em ALENCAR (2010; 2011), por meio do NLTK, diversos etiquetadores morfossintáticos aplicando as técnicas de aprendizado de máquina disponibilizadas por essa biblioteca a um vasto *corpus* anotado do português, o *Corpus* Histórico do Português Tycho Brahe (doravante CHPTB) (CORPUS, 2010). Esses etiquetadores integram o Aelius, um conjunto de módulos em Python para treino e aplicação de etiquetadores na anotação de textos em diversos formatos. Entre os etiquetadores que construímos nas arquiteturas de aprendizado de máquina nativamente implementadas no NLTK, o de melhor desempenho foi um etiquetador de Brill (doravante AeliusBRUBT), que alcançou índice de acurácia de 95% em uma amostra de textos literários do século XIX (ALENCAR, 2010; 2011). Esse desempenho foi superado por um etiquetador que posteriormente construímos por meio do HunPos, também integrado ao Aelius (doravante AeliusHunPos). Esse etiquetador permitiu obter melhores resultados na análise sintática automática.[9] Um índice de acurácia ainda maior na etiquetagem e, consequentemente, na análise sintática foi alcançado por meio do etiquetador LX-Tagger (BRANCO; SILVA, 2004). Esse etiquetador, que utiliza o aplicativo Java MXPOST (RATNAPARKHI, 1996), não é *software* livre, mas pode ser gratuitamente baixado da Internet e utilizado com base no Aelius, graças à interface do NLTK que construímos para o MXPOST.[10]

Este trabalho descreve como o ALEXP utiliza o Aelius para gerar, por meio de etiquetadores morfossintáticos, as informações lexicais de que um analisador

sintático necessita para construir representações arbóreas para as sentenças consideradas gramaticais com base nas regras de estruturação sintagmática fornecidas pelo usuário. Utilizando gramáticas ainda bastante simples, que só modelam uma fração dos fenômenos sintáticos do PB, o sistema produz resultados promissores, em se tratando de um protótipo, na análise de sentenças de mediana complexidade sintática contendo grafias não padrão, siglas, nomes próprios e neologismos não dicionarizados.

2. Análise sintática computacional do português: estado da arte

Nesta seção, fazemos, inicialmente, alguns esclarecimentos de natureza terminológica. Após a conceitualização de *parsing* profundo e etiquetagem morfossintática, precisamos a noção de *software* livre ou *software* de código aberto, no contexto da qual detectamos a inexistência de um *parser* de ampla cobertura do PB distribuído sob licença desse tipo. Expomos as vantagens desse paradigma de desenvolvimento de *software* de modo a encarecer a necessidade de preencher essa lacuna e ressaltar as facilidades que oferece para que isso seja levado a cabo. Finalmente, apontamos algumas deficiências das ferramentas existentes de processamento sintático do português na análise de exemplos de textos reais, deficiências essas que resultam, sobretudo, da cobertura lexical insuficiente. As seções seguintes evidenciam como a ampla disponibilização de recursos, principalmente no contexto do *software* livre, pode resolver esse problema.

No âmbito do processamento computacional em nível da sentença, distingue-se, inicialmente, entre a análise sintática superficial ou "rasa" (*shallow parsing*), cuja modalidade mais comum é conhecida como *chunking* (FIG. 1), e a análise sintática profunda *(deep parsing)* (FIG. 2), com base na menor ou maior complexidade das estruturas arbóreas atribuídas às sentenças.

Embora essa dicotomia corresponda comumente ao que se designa na literatura,[11] respectivamente, por análise sintática parcial (*partial parsing*) e análise sintática completa (*full parsing* ou *complete parsing*), Ljunglöf e Wirén (2010) utilizam como critério dessa última distinção não o grau de profundidade, mas o de completude das representações geradas pelo *parser*. Desse modo, a análise da Figura 1 não é só rasa, mas também parcial, uma vez que apenas os *chunks* do tipo NP são identificados. Uma análise em termos de *chunks*, porém, pode ser completa, abrangendo também PP, AP, VP etc., e apresentar um grau predefinido de complexidade (BIRD; KLEIN; LOPER, 2009, p. 277-279).

Em contrapartida, o *parsing*, como análise de estruturas acima do nível da palavra (sintagmas e *chunks*), distingue-se da análise em nível lexical (XIAO, 2010), que abrange a etiquetagem morfossintática (*POS-tagging*) e a etiquetagem morfológica *stricto sensu*. A primeira consiste na atribuição de rótulos de classes de palavras (*parts of speech*) a cada um dos *tokens* de uma sentença, levando em conta o contexto sentencial, ao passo que a segunda consiste na anotação das palavras pertencentes às categorias lexicais flexionáveis em termos de traços morfológicos como gênero, número, caso, tempo, etc. (FELDMAN; HANA, 2010, p. 5).[12] No exemplo da FIG. 3, as etiquetas morfossintáticas estão separadas das morfológicas por meio de hífen, conforme o sistema de anotação do CHPTB.

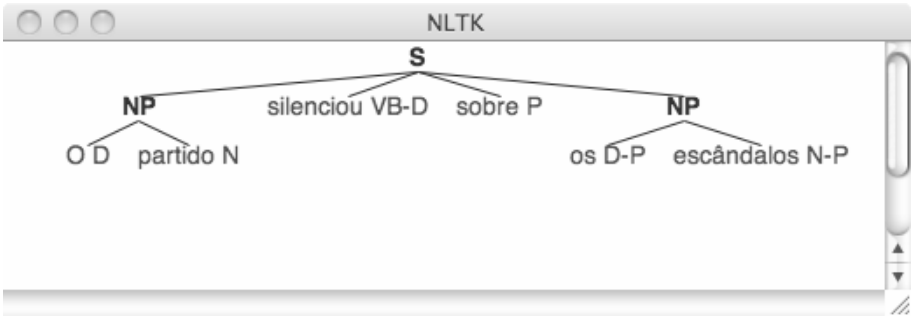

FIGURA 1: Exemplo de análise sintática rasa (*chunks*) gerada pelo NLTK com base em um *chunker* para sintagmas nominais

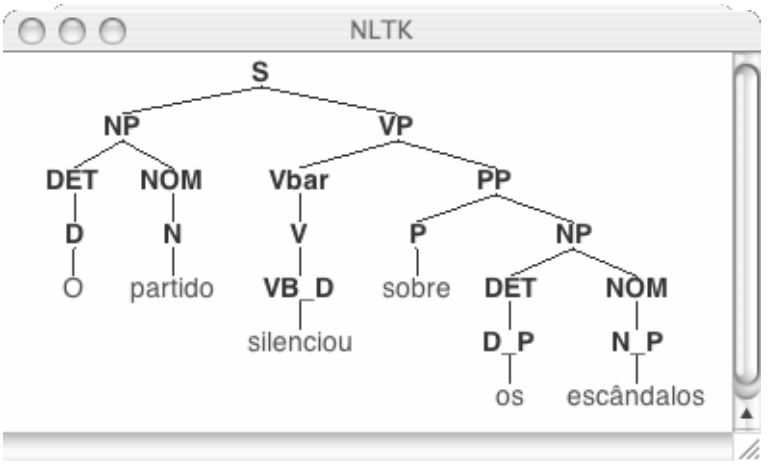

FIGURA 2: Exemplo de análise sintática profunda gerada pelo NLTK por meio do ALEXP[13]

```
>>> import alexp
>>> s=alexp.toqueniza("O partido silenciou sobre os escândalos")
>>> for w,t in alexp.etiquetaSentenca(s):
        print "%s/%s" % (w,t),


O/D partido/N silenciou/VB-D sobre/P os/D-P escândalos/N-P
>>>
```

FIGURA 3: Etiquetagem morfossintática pelo Aelius com base no ALEXP

Acreditamos que o desenvolvimento do processamento computacional do português, a exemplo de outras áreas da computação, pressupõe colocar à disposição *softwares* livres como o NLTK, que, pela sua natureza de biblioteca, possibilitam uma fácil integração a outros programas. No entanto, o que vem a ser um *software* livre, também chamado de *software* de código aberto?

Segundo a Free Software Foundation (FSF), a qualificação **livre** significa "liberdade para executar, copiar, distribuir, estudar, modificar e aperfeiçoar o software" (FREE SOFTWARE FOUNDATION, 2010). Como isso implica que o código seja aberto, os termos *software* livre, preferido pela FSF, e *software* de código aberto *(open source software)*, adotado pela Open Source Initiative (OSI) (OPEN, 2011), embora tenham conotações diferentes, passaram a ser, de um modo geral, equivalentes na prática.

Além disso, a licença de uso desse tipo de programa (no caso do NLTK, a Apache License, Version 2.0),[14] ao contrário do que ocorre com o código **proprietário**, autoriza modificações e redistribuição do *software* pela comunidade de pesquisadores. Sobre a relação entre *software* de código aberto e de código proprietário ou fechado, a OSI afirma:

> Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in. (OPEN, 2011)

Outro aspecto positivo é que projetos de *software* livre, não obstante terem à frente um ou mais desenvolvedores que o coordenam, são geralmente abertos a contribuições, tornando-se disponíveis por meio de plataformas como SourceForge[15] e Google Project Hosting,[16] que favorecem sobremaneira a

colaboração entre os desenvolvedores. O NLTK, cujo código-fonte está abrigado nesse último *site*, exemplifica isso de forma prototípica, uma vez que resulta dos esforços de dezenas de colaboradores de vários países.

Especialmente no caso do léxico, o desenvolvimento de ferramentas robustas de PLN demanda esforços gigantescos. Graças ao irrestrito compartilhamento de informações e recursos no âmbito do *software* livre, os avanços realizados por um projeto de pesquisa permitem alavancar outros projetos, formando um círculo virtuoso, de que não só o NLTK é fruto, mas também o FreeLing (CARRERAS *et al.*, 2004; PADRÓ *et al.*, 2010), entre vários outros exemplos. O código aberto, por outro lado, ao franquear acesso à implementação, numa linguagem concreta, dos diferentes algoritmos de PLN, não só permite a adaptação ou o aperfeiçoamento do código, como também constitui um recurso de valor inestimável na formação de quadros para a linguística computacional e a tecnologia da linguagem natural. Também sob esse prisma o NLTK é exemplar pelas facilidades de navegação no código-fonte, integrada à documentação da API (*application program interface*).

Pelo que sabemos, infelizmente não há nenhum *parser* profundo para textos irrestritos em PB que (i) esteja disponível gratuitamente para *download*, (ii) seja multiplataforma e (iii) tenha código aberto. No momento, o Projeto FreeLing,[17] biblioteca sob licença livre GPL, análogo ao NLTK, mas voltado não tanto ao usuário final quanto aos desenvolvedores de tecnologias da linguagem natural, torna disponível apenas um analisador morfológico e um etiquetador morfossintático para o português europeu (GARCIA; GAMALLO, 2010; PADRÓ *et al.*, 2010).

No entanto, nenhum dos analisadores do português que constam do levantamento de Menuzzi e Othero (2008) satisfaz todas as condições (i) – (iii) acima.[18] O Curupira (MARTINS; NUNES; HASEGAWA, 2003) e o PALAVRAS (BICK, 2000), por exemplo, são considerados robustos (ALMEIDA *et al.*, 2003), mas não se enquadram na categoria de *software* de código aberto, não estando o último livremente disponível para instalação no computador de qualquer usuário.[19]

Um projeto que talvez se enquadrasse na categoria de *software* livre seria o Selva (ALMEIDA *et al.*, 2003), mas, aparentemente, não teve prosseguimento. De fato, no desenvolvimento desse *parser*, Almeida *et al.* (2003) provavelmente partiram da seguinte constatação:

> Há surpreendentemente poucos projetos envolvendo a análise sintática automática do português [...]. Além disso, alguns desses projetos são comerciais, não estando disponíveis para pesquisa, ao passo que outros foram desenvolvidos visando domínios muito limitados. Temos conhecimento de apenas dois *parsers* acessíveis comparáveis ao nosso: Curupira e VISL. (ALMEIDA *et al.*, 2003, p. 103)

Os autores descrevem o Selva como um projeto ambicioso de *parser* do PB voltado para a análise de qualquer tipo de texto em prosa na norma culta, versando sobre qualquer tema, excluindo algumas construções que consideraram pouco comuns do tipo de (1).

(1) O carro Maria comprou. (ALMEIDA *et al.*, 2003, p. 103)

Ao destacarem que tanto o Curupira quanto o VISL não tiveram o código-fonte publicamente divulgado e ressaltarem a natureza comercial de outros projetos, Almeida *et al.* (2003) parecem sugerir que o Selva abraçaria a filosofia do *software* livre. No entanto, não há menção explícita no artigo sobre o tipo de licença do *parser* nem indicação sobre a sua distribuição.[20]

O Selva, implementado em Prolog, visa a atribuir às sentenças gramaticais (conforme a gramática do *parser*)[21] todas as análises sintaticamente possíveis, sem descartar análises semanticamente implausíveis, pois, como ressaltam, jogando com o célebre exemplo de Chomsky, em textos reais, "ocasionalmente ideias verdes podem dormir furiosamente" (ALMEIDA *et al.*, 2003, p. 102).

Tanto do ponto de vista gerativo quanto das abordagens gramaticais tradicionais, a gramática do *parser* discrepa em vários aspectos do que podemos considerar um denominador comum na análise dos fenômenos sintáticos. Por exemplo, o verbo principal juntamente com seus modificadores e objetos em construções com auxiliares é considerado "NP subordinado" que funciona como objeto direto. Consequentemente, o verbo *ir* em construções do tipo de "ele vai fazer isso" é classificado como verbo transitivo, cujo objeto é a ação.

O Selva é dotado de uma heurística para análise de palavras não listadas no léxico, limitando-se, contudo, a numerais e nomes próprios, e tem um mecanismo para contornar a incapacidade do procedimento de *parsing* descendente recursivo de Prolog em lidar com regras recursivas à esquerda. Isso, porém, não evitou que o *parser* entrasse em recursão infinita na análise de algumas sentenças, conforme relatam os autores.[22] Na avaliação da ferramenta que eles

fizeram, em um *corpus* de 80 sentenças, 52 foram analisadas corretamente, embora tenham sido atribuídas em média 51.4 análises por sentença, muitas das quais, segundo eles, semanticamente infundadas. Para Almeida *et al.* (2003), na esteira de Martins, Hasegawa e Nunes (2002), isso não representa um problema em si, uma vez que essas ambiguidades poderiam ser desfeitas em outros módulos de processamento.

Na comparação que Almeida *et al.* (2003) fazem entre o Selva e o VISL, destacam que o último, embora bem mais robusto, se limita a uma análise por sentença. Já para o Curupira, que também não realiza uma filtragem entre múltiplas análises geradas, eles relatam um desempenho inferior ao do próprio Selva.

Martins, Nunes e Hasegawa (2003, p. 179) definem o Curupira como "um *parser* robusto de propósito geral para o português do Brasil". Esse analisador sintático, que se baseia numa gramática livre de contexto de ampla cobertura com restrições relaxadas, visa a representar, para qualquer sentença dessa variedade, toda a gama de estruturas sintáticas que se podem atribuir às unidades morfossintáticas subjacentes. Essas análises representam não apenas a estrutura sintagmática das sentenças, mas também as relações funcionais de dependência, recorrendo a categorias próximas às da gramática tradicional, no modelo da Nomenclatura Gramatical Brasileira (MARTINS; HASEGAWA; NUNES, 2002, p. 2-4). O *parser* não faz uma desambiguação das análises, que podem ser semanticamente implausíveis, analisando também sentenças que violem restrições de concordância e regência (CURUPIRA, [2004]). No *design* do *parser*, contudo, procurou-se fazer com que ele apresentasse primeiro as análises sintáticas preferidas. A estratégia de parsing é recursivo-descendente e da esquerda para a direita. O léxico subdivide-se em dois componentes. O primeiro, constituído de um milhão e meio de vocábulos, foi compilado manualmente a partir do processamento automático de um *corpus* de 40 milhões de palavras. O segundo componente consiste de itens lexicais multipalavras e colocações. Na avaliação relatada por Martins, Nunes e Hasegawa (2003), o *parser* atribuiu pelo menos uma análise correta a 75% de um conjunto de 297 sentenças extraídas de jornais.

Numa avaliação que fizemos da versão 2.0 Beta do Curupira, esse *parser* não obteve um bom desempenho em sentenças com neologismos, devido à modelação estática da informação lexical.[23] Como vimos, o léxico do Curupira se resume a uma lista de formas extraídas de um *corpus*. A produtividade da morfologia derivacional do português não foi contemplada. Na sentença em (2), o *parser* classificou o substantivo *craqueiro* erroneamente como nome próprio. Em

todas as quatro análises atribuídas à sentença, a cadeia *uma vez um craqueiro* é analisada como sujeito da oração. Isso sugere que a gramática do *parser* não contempla NP temporais topicalizados, que constituem um fenômeno bastante corriqueiro no PB. No exemplo (3), a forma *dilmaram* não é analisada como verbo, mas como nome próprio. Esses exemplos sugerem que o Curupira, além de uma cobertura sintática insuficiente do PB, adota uma estratégia demasiado simplista de lidar com palavras desconhecidas, i.e. não listadas em seu léxico, que é classificá-las como nomes próprios. Essas limitações contrariam a finalidade do *parser*, destinado a analisar "qualquer sentença do português do Brasil […] independentemente de domínio [...]" (MARTINS; NUNES; HASEGAWA, 2003, p. 179).

(2) Uma vez um craqueiro veio me assaltar.[24]

(3) Muitos vereadores tucanos já dilmaram.

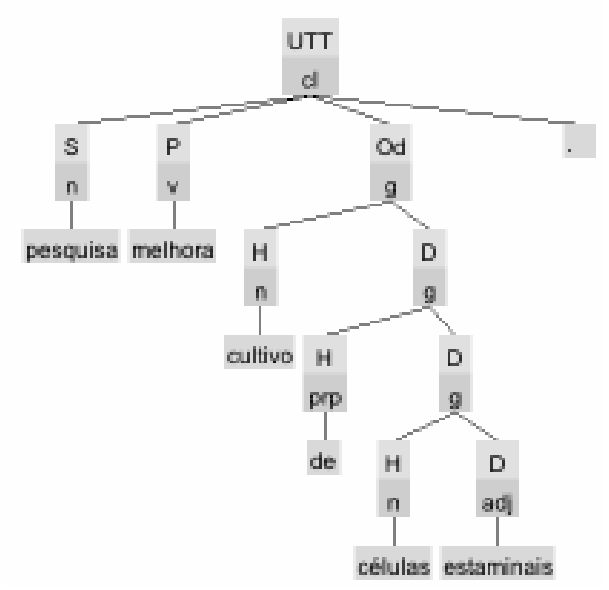

FIGURA 4: Análise de (5) pelo *parser* VISL

O contínuo desenvolvimento do *parser* PALAVRAS culminou no analisador sintático do Projeto VISL, que, diferentemente do Curupira, dispõe de uma interface na WWW para análise de textos.[25] Pudemos constatar, por meio da análise de sentenças de estrutura complexa ou não canônica e lexicalmente

difíceis (devido a siglas, nomes próprios e neologismos), que se trata de um *parser* extremamente robusto. De fato, foram analisadas corretamente, por exemplo, as sentenças (2), (3), (4) e (5) (ver representação arbórea dessa última na FIG. 4).

(4) O deputado questionou que o então governador também teria liberado o deslocamento dos cantores.

(5) Pesquisa melhora cultivo de células estaminais. (PES)[26]

Curiosamente, o *parser* do projeto VISL falhou, contudo, na análise da sentença (6), extraída de Freitas, Rocha e Bick (2008), que tratam justamente da aplicação do parser PALAVRAS na anotação sintática de um *corpus* do português. Como podemos constatar na FIG. 5 e FIG. 6, a forma da terceira pessoa do singular do presente do indicativo do verbo *afetar*, bastante corriqueira, foi erroneamente classificada como adjetivo.[27] Isso levou o *parser* a atribuir à sentença uma análise absurda, segundo a qual a construção constitui uma estrutura de coordenação (denominada *paratagma*) envolvendo um sintagma adjetival e um sintagma nominal descontínuo.

(6) Seca afeta pouco a produção de grãos. (FREITAS; ROCHA; BICK, 2008, p. 143)

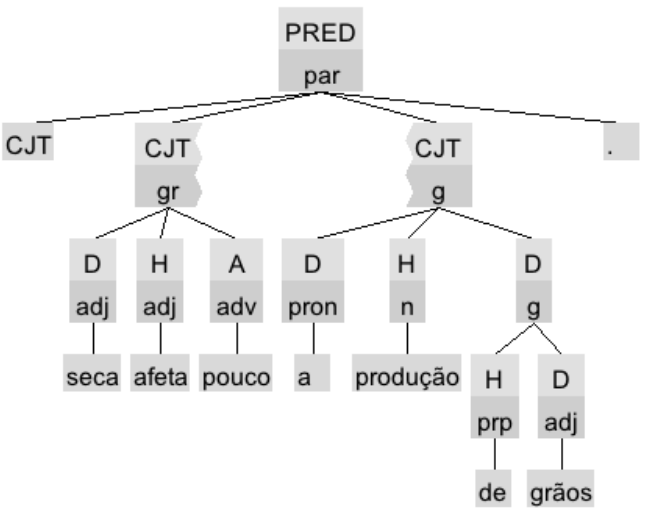

FIGURA 5: Representação arbórea da sentença (6) pelo *parser* do projeto VISL[28]

```
SOURCE: Running text
1. seca afeta pouco a produção de grãos.
A1
PRED:par
.
|-CJT:adjp
|-CJT:g(np)-
| |-D:adj("seco" F S)   seca
| |-H:adj("afeto" &lt;n&gt; F S)        afeta
| |-A:adv("pouco" &lt;quant&gt; &lt;fr:100&gt;) pouco
|-*CJT:g(np)
  |-D:pron(det "o" &lt;artd&gt; DET F S)        a
  |-H:n("produção" F S) produção
  |-D:g(pp)
    |-H:prp("de" &lt;f:0&gt;)   de
    |-D:adj("grão" M P) grãos
```

FIGURA 6: Representação textual da análise da sentença (6) pelo *parser* do projeto VISL

Outro *parser* listado por Menuzzi e Othero (2008) é o LX-Parser, um analisador sintagmático (*constituency parser*) que integra a LX-Suite, pacote de ferramentas para o processamento computacional do português europeu desenvolvido pelo LX-Center do Grupo de Fala e Linguagem Natural (NLX), do Departamento de Informática da Universidade de Lisboa.[29] Em sua versão atual, o *parser* resulta da comparação realizada por Silva *et al*. (2010) entre diferentes *parsers* dessa variedade da língua construídos com pouco esforço (o que eles denominam *out-of-the-box parsers*), indutivamente, por meio de pacotes de *software* livremente disponíveis que permitem treinar *parsers* em florestas sintáticas. Os autores concluem que o pacote de Berkeley permitiu alcançar o maior índice de acurácia. Como desdobramento desse trabalho, um *parser* para o português europeu foi treinado com o pacote de Stanford e disponibilizado gratuitamente no sítio do LX-Center da Universidade de Lisboa, tanto para análise *on-line* de pequenos trechos (menos de 500 caracteres pelo que pudemos apurar) quanto para acesso remoto via *Web service* ou instalação no computador do usuário (SILVA; BRANCO; GONÇALVES, 2010).[30] Embora Silva, Branco e Gonçalves (2010, p. 1962) considerem o LX-Parser "o primeiro *parser* probabilístico robusto do português livremente disponível", ele não constitui

software livre (na definição da FSF), uma vez que a respectiva licença proíbe a sua redistribuição ou a distribuição de produtos derivados.

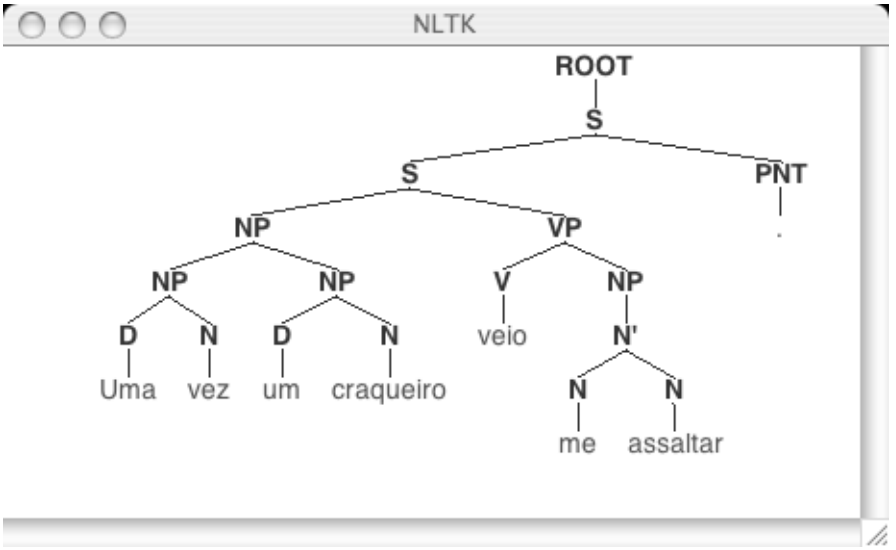

FIGURA 7: Representação arbórea produzida no NLTK para a análise atribuída a (2) pela versão *off-line* do LX-Parser

Na avaliação que fizemos, o LX-Parser, a exemplo do Curupira, não analisou corretamente o NP adverbial que encabeça (2), como evidencia a FIG. 7.[31] Em contrapartida, nessa árvore, o complemento do verbo *vir* está incorretamente classificado como NP.

O LX-Parser revelou-se também impreciso na análise de sentenças com neologismos, como mostra a FIG. 8. Nessa árvore, o quantificador *muitos* está classificado como verbo e a forma verbal *dilmaram*, como adjetivo. Trata-se de "erros duros" (*harte Fehler*, conforme NAUMANN, 2004, p. 155, n. 1) devido à disparidade entre as categorias quantificador e verbo, por um lado, e verbo e adjetivo, por outro.

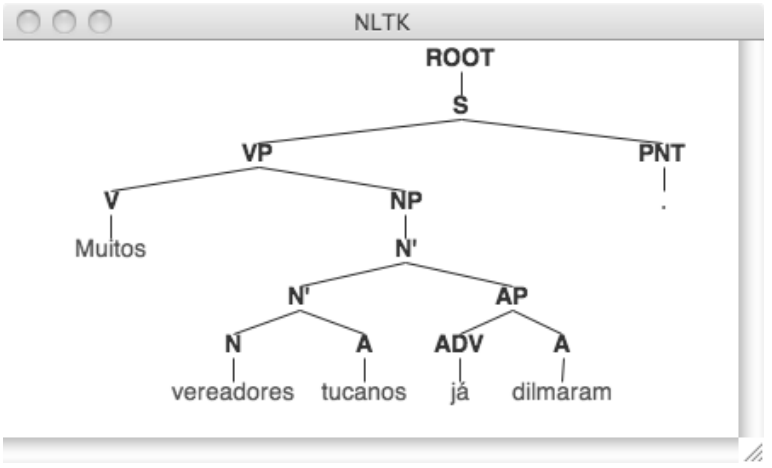

FIGURA 8: Análise de (3) pelo LX-Parser

No entanto, mesmo quando não há complicações lexicais aparentes, como nos exemplos (4), (5) e (6), também são cometidos erros "duros" na classificação das palavras, como podemos constatar nas FIG. 9, FIG. 10, FIG. 11 e FIG. 12.[32] Esses erros, por sua vez, impedem que seja atribuída uma árvore correta a toda ou a partes substanciais das sentenças analisadas.

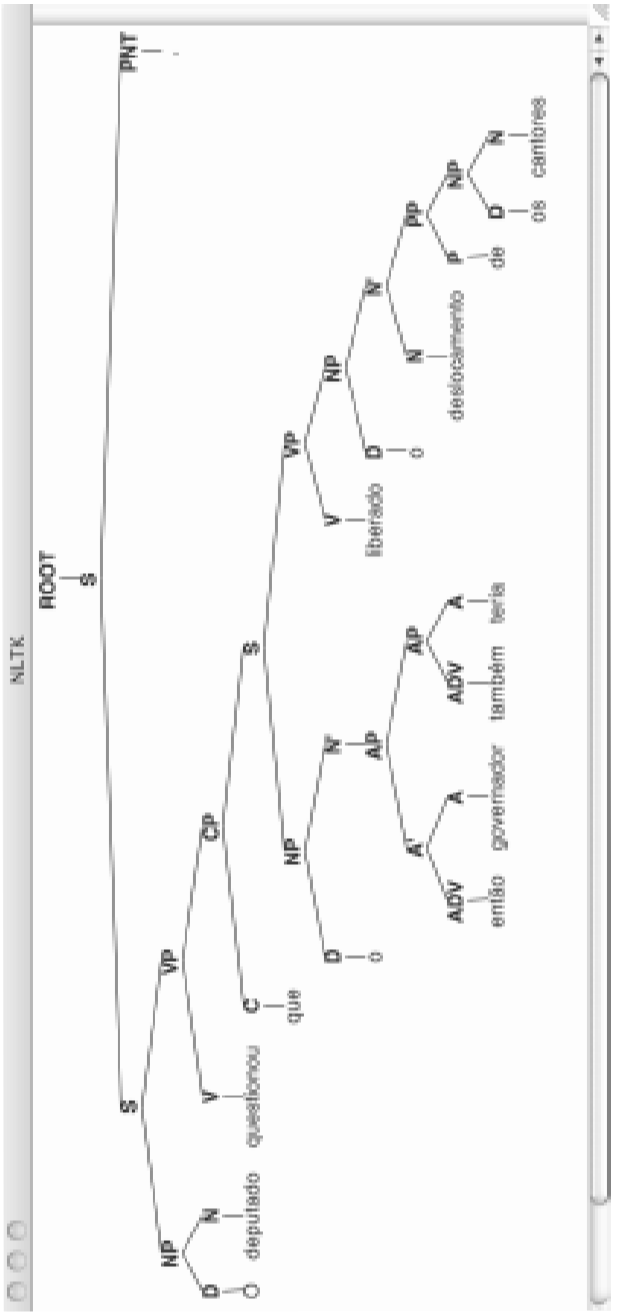

FIGURA 9: Primeira análise de (4) gerada pelo LX-Parser

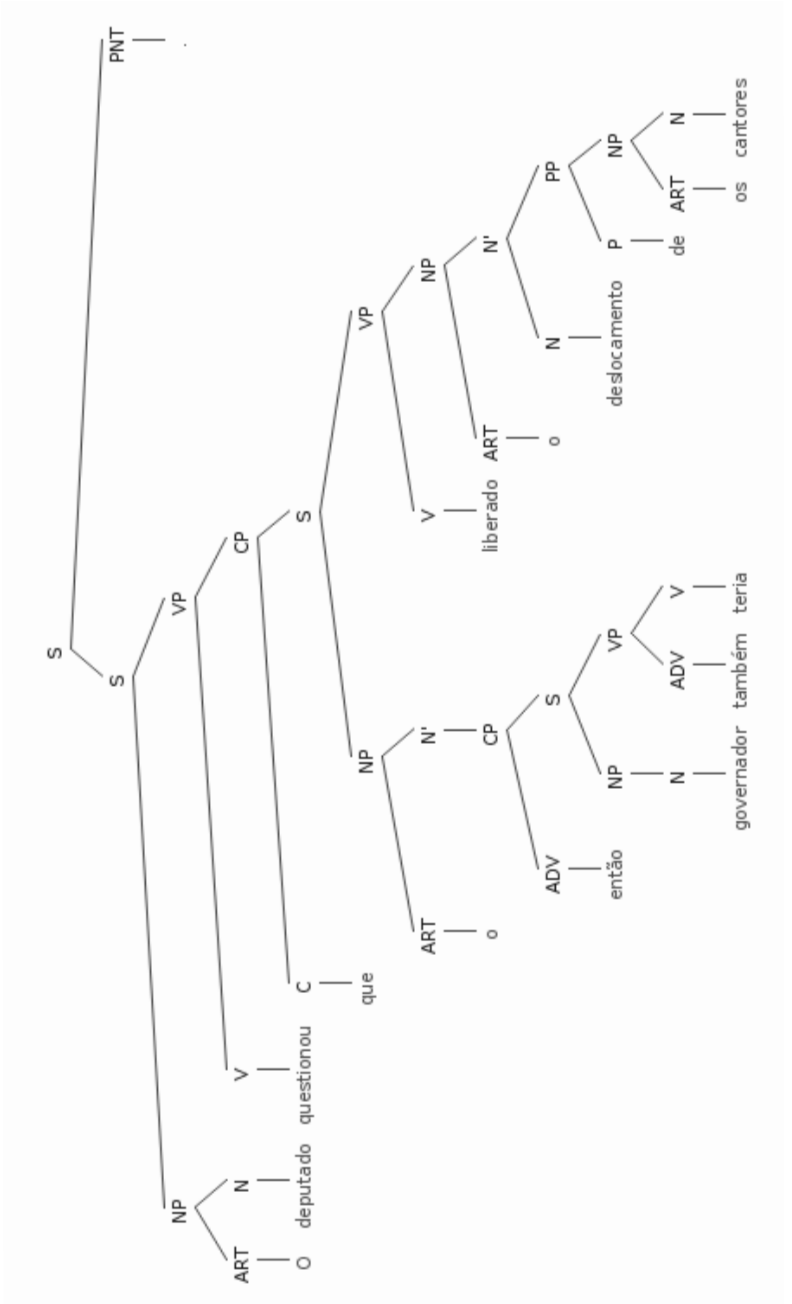

FIGURA 10: Segunda análise de (4) gerada pelo LX-Parser

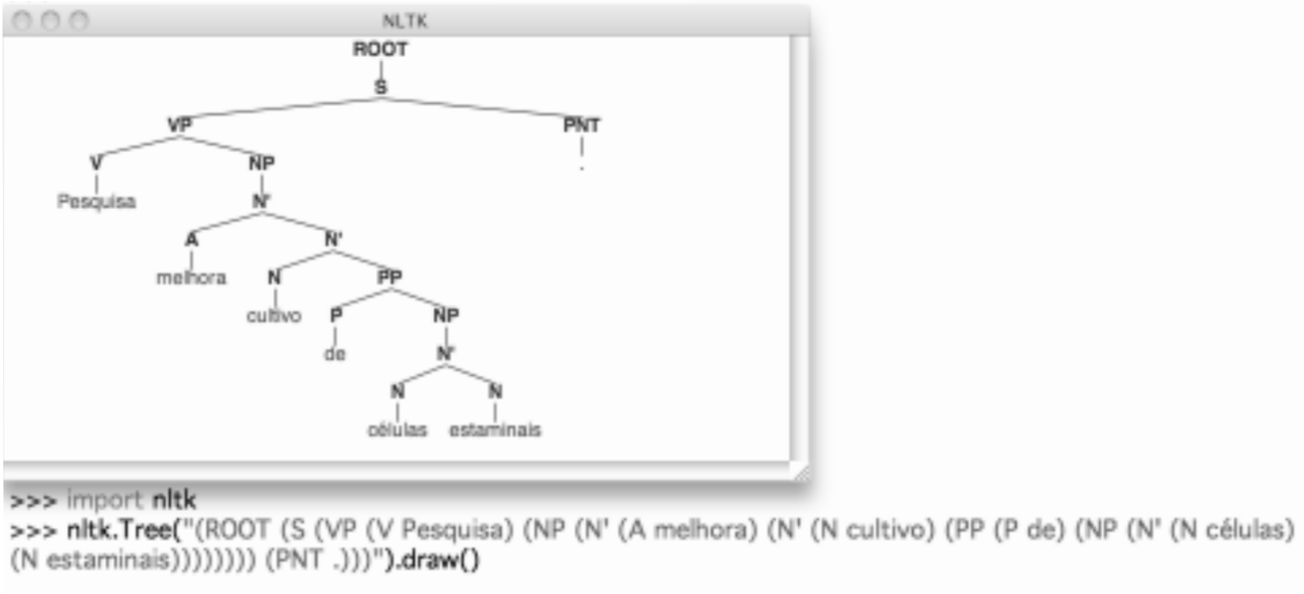

FIGURA 11: Representação arbórea produzida no NLTK para a análise atribuída a (5) pela versão *off-line* do LX-Parser

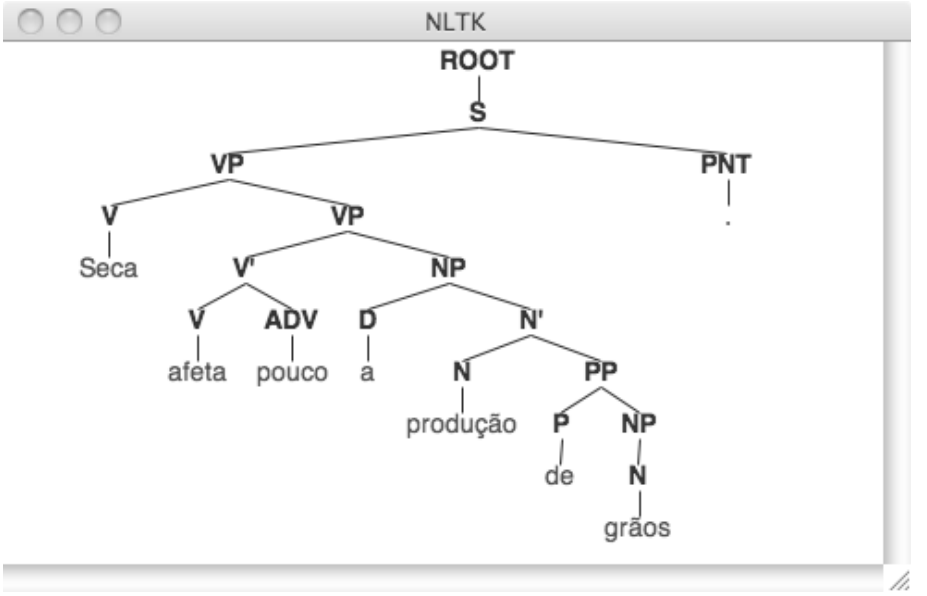

FIGURA 12: Representação arbórea produzida no NLTK para a análise atribuída a (6) pela versão *off-line* do LX-Parser

Pelo mesmo motivo que o LX-Parser, também não se enquadra na categoria de *software* livre a LXGram (BRANCO; COSTA, 2010), outra importante contribuição do NLX para o processamento computacional profundo do português, embora tenha o código-fonte divulgado e seja livremente disponível para *download* (LXGRAM, [s.d.]).[33] Desenvolvida no formalismo da HPSG e voltada para a análise automática tanto em nível sintático quanto semântico, trata-se, aparentemente, da mais abrangente gramática computacional do português, contemplando tanto a variedade europeia quanto a brasileira. Segundo Branco e Costa (2010), um *parser* compilado a partir da LXGram foi capaz de analisar 32% das sentenças de uma amostra de mais de 120.000 sentenças pré-processadas por um etiquetador morfossintático.

Contier, Padovani e José Neto (2010) descrevem o *Linguístico*, um "reconhecedor gramatical" com base em gramática de Celso Luft e implementado em Python que utiliza Tecnologia Adaptativa para verificar a gramaticalidade de uma cadeia dada como *input*, atribuindo-lhe uma representação estrutural. O

sistema é constituído de 5 módulos. Após a segmentação sentencial e a toquenização, o módulo Identificador Morfológico recorre ao *corpus* Bosque "como biblioteca de apoio" (p. 39) para atribuir uma etiqueta de POS, de um conjunto de 21 etiquetas, ao *output* do módulo Toquenizador, fornecendo, por sua vez, o *input* para o Agrupador, responsável pela "montagem" dos sintagmas. Por fim, o último módulo, o Reconhecedor Sintático, determina se a sequência de sintagmas instancia um padrão sentencial válido. Não são fornecidas informações sobre a avaliação da ferramenta. Os autores se limitam a exemplificar a aplicação do programa na análise de uma única sentença de 32 palavras, que produz a representação de (7).

(7) [[['A Câmara', 'SS', '2'], ['de Representantes do Congresso americano', 'SP', '2'], ['deu', 'V', '3'], ['início', 'SS', '4'], ['ao debate de um projeto de lei histórico para limitar as emissões de gases causadores do efeito estufa pela indústria dos Estados Unidos', 'SP', '9']]][34]

Em suma, podemos afirmar que a análise sintática automática de textos irrestritos em PB, do ponto de vista do PLN, é uma área que ainda demanda muitos esforços de pesquisa, dadas as limitações das ferramentas disponíveis, que ou constituem *software* proprietário ou são de acesso restrito, ou não são precisas o suficiente, gerando análises errôneas mesmo para exemplos aparentemente triviais.

Uma outra perspectiva de avaliação, esta bastante controversa, refere-se à adequação linguística das representações geradas pelas ferramentas. O Selva, o Curupira e o VISL atribuem às sentenças consideradas gramaticais representações que, na gramática gerativa, com o advento da teoria X-barra, foram descartadas como psicologicamente implausíveis e explanatoriamente inadequadas, uma vez que fogem ao esquema X-barra (GREWENDORF; HAMM; STERNEFELD, p. 1989, p. 193; GREWENDORF, 2002, p. 33). O *Linguístico* não chega a fazer uma análise sintática profunda, limitando-se, como evidencia o exemplo (7), a identificar *chunks*.

O LX-Parser segue alguns princípios do modelo X-barra, mas não adota outros de forma sistemática, como evidenciam a FIG. 9 e a FIG. 10. O princípio da endocentricidade é violado pela categoria S, definida exocentricamente, e pelo N' que, respectivamente na FIG. 9 e na FIG. 10, domina exaustivamente AP e CP. Nas representações $[_{NP}\ [_{D}$ o$]\ [_{N}$ deputado$]]$ e $[_{AP}\ [_{ADV}$ então$]\ [_{A}$ governador$]]$, também é desrespeitado o princípio da maximalidade, o qual reza que todo não

núcleo é uma projeção máxima (GREWENDORF; HAMM; STERNEFELD, p. 1989, p. 202).[35]

Do ponto de vista da teoria gerativa, um *parser* que se destaca, junto com o LX-Parser e a LXGram, pela maior adequação linguística das representações atribuídas às sentenças é o Grammar Play (OTHERO, 2006), uma vez que modela um relativamente extenso fragmento do PB com base na teoria X-barra, embora também ocasionalmente a viole, em parte devido a limitações do formalismo DCG da linguagem de programação Prolog em que o *parser* foi implementado (ALENCAR, 2008). Sem visar à analise textos irrestritos, essa ferramenta, que tem código aberto e é livremente disponível, destina-se ao ensino da sintaxe, não sendo, portanto, robusta, uma vez que tem léxico limitado e se restringe a um leque estreito de construções (MENUZZI; OTHERO; 2008).[36]

Em aplicações do PLN, representações sintáticas bastante hierarquizadas, tais como preconizadas nas diferentes versões da teoria X-barra em diferentes modelos da gramática gerativa, são comumente rejeitadas em favor de representações mais planas, de acordo com Jurafsky e Martin (2009, p. 430) por serem mais simples. Maier (2007), porém, ao treinar *parsers* baseados na PCFG em duas florestas sintáticas do alemão, constatou que estruturas mais hierarquizadas favorecem significativamente o desempenho do *parsing* em termos de *F-score*.

Por outro lado, *parsers* são desenvolvidos frequentemente com a finalidade de anotar *corpora*, para a compilação de florestas sintáticas, aplicação em que é comum a posição adotada por Freitas, Rocha e Bick (2008) na *Floresta Sintá(c)tica*. Visando a uma neutralidade teórica (que os próprios autores reconhecem ser impossível alcançar), as representações desse *corpus* aproximam-se da gramática tradicional. Consideramos, no entanto, problemática essa opção, dado o caráter não formalizado dos construtos desse modelo gramatical. Julgamos mais acertada a posição de Branco *et al.* (2010) no *corpus* CINTIL DeepGramBank. Esse *corpus* foi anotado por meio da LXGram com base na HPSG, um modelo gerativo extremamente complexo que integra diversos níveis de análise, tanto sintáticos quanto semânticos. No entanto, para não assoberbar os usuários do *corpus* com essa complexidade, ferramentas desenvolvidas no âmbito desse projeto permitem a extração de níveis individuais de representação, inclusive representações sintáticas simplificadas baseadas na teoria X-barra. A abordagem de Branco *et al.* (2010) vai ao encontro da posição de Hajicová *et al.* (2010, p. 171), para quem a anotação de *corpus* deve fundamentar-se em uma "teoria linguística bem definida", cuja consistência possa ser testada na própria anotação.

Concluímos esta seção apontando que a análise sintática automática do PB, do ponto de vista da gramática gerativa, constitui um vasto campo a ser explorado, pois inexiste um *parser* que implemente de forma consistente a teoria X-barra, não obstante a existência de descrições da língua com base nesse modelo que utilizam a CFG (e são, portanto, mais próximas de uma implementação computacional), como Othero (2009), que leva em conta a hipótese DP e outros desdobramentos da teoria X-barra "clássica". Um *parser* robusto com base em uma gramática do PB de ampla cobertura nos moldes da teoria X-barra seria de enorme importância para verificar a adequação desse modelo na análise de textos reais, seguindo a sugestão de Hajičová *et al.* (2010).

3. O problema da aquisição lexical na análise sintática automática

Nesta seção, precisamos a noção de aquisição lexical, mostrando como representa um problema para o PLN, na medida em que aplicações como *parsers* não podem simplesmente recorrer aos dicionários existentes, que, por mais abrangentes que sejam, não fornecem as informações necessárias para a análise de textos reais. Por meio de exemplos, mostramos que etiquetadores morfossintáticos, treinados em *corpora* por meio de técnicas de aprendizagem de máquina, podem fornecer informações inexistentes nos dicionários, colocando-se como alternativas não só viáveis, mas de baixíssimo custo para a solução do problema da modelação do léxico em sistemas de *parsing*.

Conforme Lemnitzer e Wagner (2004, p. 245), distinguem-se três acepções de **aquisição lexical** (*lexical acquisition*):

- Na tecnologia do texto: extração de descrições lexicais a partir de textos.
- Na psicolinguística: aquisição de dados necessários à construção do léxico mental.
- Na linguística de *corpus*: aquisição de informações lexicais ao longo do processo de anotação de textos.

No presente trabalho, focamos a primeira acepção, que é a mais diretamente relevante para a construção de aplicações da linguística computacional, tais como analisadores sintáticos, corretores gramaticais e tradutores automáticos, uma vez que esses programas "necessitam de informações lexicais de forma muito mais

abrangente e explícita do que usuários humanos de dicionários [...]" (LEMNITZER; WAGNER, 2004, p. 246). Dada a insuficiência, sob esse prisma, dos dicionários, sobre a qual nos detemos mais adiante, restaria produzir manualmente, para cada aplicação, as entradas lexicais necessárias. Todavia, tanto a gigantesca dimensão quanto o caráter dinâmico do léxico de línguas naturais vivas, constantemente enriquecidas de novas lexias, tornam impraticável a listagem manual, ou mesmo a modelação computacional por meio de regras formuladas manualmente, das informações lexicais exigidas por sistemas de PLN de grande envergadura, capazes de analisar textos autênticos de diferentes gêneros. Esse gargalo, no entanto, pode ser superado, graças à área de aprendizado de máquina, que oferece técnicas para diversos tipos de modelação automática da informação lexical. Neste trabalho, focamos uma dessas abordagens, que é a construção de etiquetadores morfossintáticos com base em *corpora* anotados.

Um exemplo do problema que o léxico representa para o desenvolvimento de sistemas de análise sintática automática é dado por Othero (2006), que descreve a implementação do *Grammar Play*, referido na seção . Na construção do léxico, Othero partiu de uma lista de 67.500 palavras na sua forma canônica, agrupadas em categorias lexicais (verbos, adjetivos, etc.). Conforme ressalta o autor, cada forma verbal infinitiva da lista desdobra-se em mais de 50 formas conjugadas. Embora razoavelmente abrangente do ponto de vista sintático, esse *parser* padece de uma grave limitação, uma vez que inclui apenas as formas da terceira pessoa do indicativo dos verbos, devido à dificuldade de implementar entradas para as formas conjugadas, mesmo recorrendo à modelação computacional da flexão.[37]

Dicionários tradicionais impressos são inadequados como fonte de dados lexicais para o PLN porque não apresentam informações confiáveis (quando muito) sobre as frequências de ocorrência, não oferecendo cobertura suficientemente ampla do vocabulário e seus diferentes usos (LEMNITZER; WAGNER, 2004, p. 247). Além disso, pecam pela subjetividade e falta de consistência e de atualidade das informações dadas (LEMNITZER; WAGNER, 2004, p. 247-248).

Vejamos um exemplo de descompasso entre dicionários recentes do português e o uso atual da língua. Conforme os dicionários tradicionais consultados, o verbo *questionar* admite apenas complementos de natureza nominal ou preposicional (HOUAISS; VILLAR, 2001; IDICIONÁRIO, 2011). O dicionário de Borba *et al.* (1997), dedicado a verbos e baseado em corpus, acrescenta a essa informação valencial a possibilidade de o complemento

ser uma oração, com a restrição de que deve ser introduzida pela conjunção *se*, configurando a moldura valencial *questionar* + [oração interrogativa indireta], como no exemplo seguinte:

(8) A senadora Marina Silva [...] questionou nesta sexta-feira se os partidos envolvidos em escândalos de corrupção irão fazer um "pacto de silêncio" durante a campanha […]. (RON)

No PB, porém, encontramos, em textos atuais, também o complementador *que*, introduzindo oração declarativa, e não interrogativa:

(9) O deputado Ivo Som [...] questionou que o então governador Ottomar Pinto também teria liberado o deslocamento dos cantores Margarete Menezes e Leonardo até suas cidades de destino [...]. (DEP)

Nas acepções dadas pelos dicionários, *questionar* funciona como *verbum interrogandi* (exemplo (8)), parafraseando-se por *perguntar*, ou como verbo transitivo direto, parafraseando-se por *fazer perguntas a*, *lançar dúvida sobre, levantar questões acerca de, pôr em questão*, *contestar*, etc. No entanto, essas paráfrases são inaplicáveis no exemplo (9). O uso do verbo *questionar* como *verbum dicendi*, que se poderia parafrasear talvez por *argumentar*, que constatamos nesse exemplo, não está consignado nem mesmo no iDicionário Aulete (2011), o qual, por ser *on-line*, pode ser atualizado com mais frequência.

Um sistema de análise sintática automática, como parte, por exemplo, de um programa de correção gramatical, que se limitasse às informações dos dicionários apontaria indevidamente um erro em (9). Pelo contrário, se alimentarmos o sistema com dados de *corpora*, aplicando técnicas do aprendizado de máquina para a aquisição de informações lexicais, a moldura valencial *questionar* + [oração declarativa] também será contemplada.

O contraste entre os seguintes exemplos é bastante ilustrativo da falta de cobertura dos dicionários em relação à múltipla categorização sintática de muitos lexemas:

(10) A UFC (Universidade Federal do Ceará) anunciou a concorrência para vagas remanescentes em 16 cursos, destinadas aos candidatos **classificáveis** do Vestibular 2010 […]. (UNI)

(11) Os **classificáveis** dos cursos de Sistemas de Informação [...] poderão ocupar as vagas remanescentes do Curso de Engenharia de Software. (UNI) [grifos nossos]

Enquanto em (10) *classificáveis* se enquadra na categoria adjetivo, em (11) a categoria lexical é de substantivo. No iDicionário Aulete (2011), porém, apenas a primeira categoria é atribuída ao verbete *classificável*, tratamento diferente do que o dicionário confere a *recebível*, classificado tanto como adjetivo quanto como substantivo. Um *parser* cujo léxico fosse extraído de um dicionário desse tipo não analisaria uma sentença como (11), se assumimos, como na teoria X-barra no âmbito da hipótese DP, que um NP tem um substantivo como núcleo. Esse *parser* só analisaria sentenças do tipo de (12).

(12) Os candidatos **classificáveis** dos cursos de Sistemas de Informação poderão ocupar as vagas remanescentes do Curso de Engenharia de Software.

Diferentemente dos dicionários, etiquetadores morfossintáticos baseados em arquiteturas robustas e treinados num *corpus* suficientemente representativo, como o LX-Tagger (BRANCO; SILVA, 2004), não têm dificuldade em lidar com múltiplas categorizações sintáticas de palavras como *classificável* e mesmo de palavras *non-sense* como *pitorocável* nos exemplos (13) e (14),[38] como podemos verificar na FIG. 13 e na FIG. 14 (CN é a etiqueta do LX-Tagger para substantivos comuns).

(13) Os livros pitorocáveis chegaram.

(14) Os pitorocáveis chegaram.

```
>>> from Aelius import carrega, AnotaCorpus
>>> lx=carrega("lxtagger")
>>> s1="Os classificáveis dos cursos de Sistemas de Informação poderão ocupar as vagas remanescentes do Curso de Engenharia de So
ftware".decode("utf-8")
>>> for w,t in AnotaCorpus.anota_sentencas([s1.split()],lx,"mxpost")[0]:
        print "%s/%s " % (w,t),


Os/DA classificáveis/CN de/PREP os/DA cursos/CN de/PREP Sistemas/PNM de/PREP Informação/PNM poderão/V ocupar/INF as
/DA vagas/CN remanescentes/ADJ de/PREP o/DA Curso/PNM de/PREP Engenharia/PNM de/PREP Software/PNM
>>> s2="Os candidatos classificáveis dos cursos de Sistemas de Informação poderão ocupar as vagas remanescentes do Curso de Enge
nharia de Software".decode("utf-8")
>>> for w,t in AnotaCorpus.anota_sentencas([s2.split()],lx,"mxpost")[0]:
        print "%s/%s " % (w,t),


Os/DA candidatos/CN classificáveis/ADJ de/PREP os/DA cursos/CN de/PREP Sistemas/PNM de/PREP Informação/PNM poderão/
V ocupar/INF as/DA vagas/CN remanescentes/ADJ de/PREP o/DA Curso/PNM de/PREP Engenharia/PNM de/PREP Software/PN
M
>>>
```

FIGURA 13: Etiquetagem morfossintática dos exemplos (11) e (12) pelo LX-Tagger com base no Aelius

```
>>> s="os pitorocáveis chegaram".decode("utf-8")
>>> AnotaCorpus.anota_sentencas([s.split()],lx,"mxpost")
[[('Os', 'DA'), ('pitoroc\xc3\xa1veis', 'CN'), ('chegaram', 'V')]]
>>> s="os livros pitorocáveis chegaram".decode("utf-8")
>>> AnotaCorpus.anota_sentencas([s.split()],lx,"mxpost")
[[('Os', 'DA'), ('livros', 'CN'), ('pitoroc\xc3\xa1veis', 'ADJ'), ('chegaram', 'V')]]
>>>
```

FIGURA 14: Etiquetagem morfossintática de sentenças com a palavra *non-sense* **pitorocável*

Dada a disponibilidade de *corpora* anotados morfossintaticamente, a construção de etiquetadores morfossintáticos tornou-se, nos últimos anos, uma tarefa quase trivial, graças a ferramentas de aprendizado de máquina colocadas à disposição livremente (muitas das quais constituindo *software* livre), como o MXPOST (RATNAPARKHI, 1996), o HunPos (HALÁCSY; KORNAI; ORAVECZ, 2007), entre vários outras, além das que integram o NLTK (BIRD; KLEIN; LOPER, 2009; 2011).

Por outro lado, a língua portuguesa dispõe, além do LX-Tagger (BRANCO e SILVA, 2004) e do Aelius (ALENCAR, 2010), sobre os quais nos detemos neste trabalho, de outros etiquetadores pré-treinados que se podem livremente descarregar da Internet, como o etiquetador do FreeLing (GARCIA, M.; GAMALLO, 2010).

O comando da *shell* do Unix em (15), que leva apenas 15 segundos para executar,[39] exemplifica a facilidade de treinar um etiquetador em um *corpus*, no caso, uma versão modificada do CHPTB compreendendo quase 1,5 milhão de *tokens* (ALENCAR, 2010, 2011). O etiquetador resultante, o AeliusHunPos, obteve acurácia acima de 95% em uma amostra de textos literários do século XIX, próximo, portanto, do estado da arte de 96% a 97% para línguas do tipo da portuguesa (GÜNGÖR, 2010, p. 207).[40] Aplicado aos exemplos de (11) a (14), o AeliusHunPos classificou corretamente, como adjetivo e substantivo, respectivamente, as duas ocorrências de *pitorocáveis* em (13) e (14), mas teve desempenho inferior ao LX-Tagger, falhando no reconhecimento de *classificáveis* em (11) como substantivo e atribuindo aos elementos do sintagma *vagas remanescentes*, respectivamente, as etiquetas ADJ-F-P e N-P.

(15) $ **cat chptb.tok | hunpos-train AeliusHunPos**

4. O formalismo da CFG e sua aplicação em protótipos de gramática do PB

Nesta seção, tratamos, inicialmente, do formalismo da CFG, explicando como se constroem *parsers* tabulares ascendentes para esse tipo de gramática no NLTK. Evidenciamos a dificuldade de modelação do léxico na construção de *parsers* profundos para textos irrestritos nesse ambiente. Em seguida, elaboramos um fragmento de CFG do PB, cobrindo um leque razoável de fenômenos medianamente complexos. Na próxima seção, utilizamos o ALEXP para construir, a partir desses protótipos, por meio do NLTK, diversos *parsers* tabulares ascendentes. O conhecimento lexical desses *parsers*, contudo, não é previamente modelado sob a forma de entradas lexicais, como nos *parsers* Selva, Curupira ou Grammar Play, ou por meio de regras morfológicas. Em vez disso, propomos uma alternativa muito menos custosa e bem mais eficiente, que é utilizar o *output* de diversos etiquetadores morfossintáticos aplicados sobre as sentenças dadas para análise. O reaproveitamento, no *parsing* profundo, dessas ferramentas, que se encontram livremente disponíveis, é possibilitado pelo ALEXP.

O NLTK oferece diversos formalismos para a construção de analisadores sintáticos. Na fase inicial de nosso analisador, utilizamos o formalismo da gramática livre de contexto (doravante CFG, do inglês *context-free grammar*), comumente utilizado na gramática gerativa e na linguística computacional para a implementação de protótipos capazes de modelar a estrutura sintagmática de uma língua natural.[41]

Uma gramática CFG é como o esqueleto a partir do qual podemos desenvolver, recorrendo a estruturas de traços no âmbito de formalismos como a LFG e a HPSG, gramáticas mais sofisticadas, capazes de modelar, de forma elegante e computacionalmente eficiente, fenômenos mais complexos como a concordância e a subcategorização.[42] Já *parsers* estatísticos treinados em florestas sintáticas (*treebanks*) comumente utilizam uma variante da CFG, a PCFG, em que as regras são associadas a probabilidades (NIVRE, 2010).

Simplificando, podemos definir uma CFG como uma gramática constituída de regras de reescrita (*rewriting rules*) de um dos seguintes tipos:

- A -> B, em que A é um símbolo *não terminal* e B, um símbolo não terminal ou uma concatenação de símbolos não terminais.
- C -> d, em que C é um símbolo não terminal e d, um símbolo *terminal*.[43]

Símbolos não terminais são aqueles que devem ser substituídos, mediata ou imediatamente, na derivação de uma sentença, por um símbolo terminal, que não pode ser substituído por outro item. No caso do PLN, os símbolos terminais representam os itens do léxico, ao passo que os símbolos não terminais pré-terminais constituem categorias lexicais e os demais não terminais, categorias sintagmáticas.

A título de exemplo, seja a minigramática (17),[44] que gera apenas a sentença (16):

(16) Os partidos silenciaram.

(17) Minigramática I
S -> NP VP
NP -> Det N
VP -> V
Det -> 'os'
N -> 'partidos'
V -> 'silenciaram'

As três primeiras regras descrevem a estrutura sintagmática (onde NP e VP são categorias sintagmáticas, Det, N e V, categorias lexicais), ao passo que as restantes modelam o léxico, constituindo entradas lexicais elementares.[45] Para gerar também uma sentença como "o partido silenciou sobre os escândalos", precisamos incluir em (17) uma regra para o sintagma preposicional (PP), outra para o sintagma verbal e mais entradas lexicais, resultando na gramática de (18).

(18) Minigramática II
S -> NP VP
VP -> V
VP -> V PP
NP -> Det N
PP -> P NP
Det -> 'o'
Det -> 'os'
N -> 'partido'
N -> 'partidos'
N -> 'escândalos'
V -> 'silenciou'
V -> 'silenciaram'
P -> 'sobre'

```
>>> import nltk
>>> cfg="""
S ->  NP VP
VP -> V
VP -> V PP
NP -> Det N
PP -> P NP
Det -> 'o'
Det -> 'os'
N -> 'partido'
N -> 'partidos'
N -> 'escândalos'
V -> 'silenciou'
V -> 'silenciaram'
P -> 'sobre'
"""
>>> gramatica=nltk.parse_cfg(cfg)
>>> type(gramatica)
<class 'nltk.grammar.ContextFreeGrammar'>
>>> gramatica
<Grammar with 13 productions>
>>> analisador=nltk.ChartParser(gramatica)
>>> type(analisador)
<class 'nltk.parse.chart.ChartParser'>
>>> analisador
<nltk.parse.chart.ChartParser object at 0x2689750>
>>> arvores=analisador.nbest_parse("os partidos silenciaram sobre os escândalos".split())
>>> nltk.draw.draw_trees(*arvores)
```

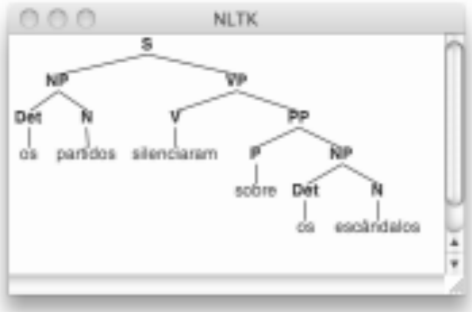

FIGURA 15: Compilação de *parser* pelo NLTK com base na gramática de (18) no ambiente IDLE de Python

Para a análise de cadeias (*strings*) de uma determinada língua livre de contexto (o tipo de língua na Hierarquia de Chomsky gerada por uma CFG), o NLTK oferece *classes* que implementam alguns dos algoritmos mais conhecidos. Na FIG. 15, exemplificamos a construção de um *parser* tabular ascendente (*botton-up chart parser*) para a gramática de (18) no ambiente IDLE, a amigável interface gráfica de Python. Essa estratégia de *parsing* é bem mais eficiente que a recursivo-descendente (JURAFSKY; MARTIN, 2009, p. 469), que, como vimos, subjaz aos *parsers* Selva e Curupira. Em contrapartida, diferentemente do formalismo DCG de Prolog, que não admite regras recursivas à esquerda, *parsers* tabulares ascendentes podem lidar com regras recursivas tanto à direita quanto à esquerda, de que veremos exemplos mais adiante.

Para construir o *parser* da FIG. 15, primeiramente importamos a biblioteca NLTK. Em seguida, atribuímos à variável cfg uma cadeia com as produções da gramática. A aplicação da função nltk.parse_cfg() sobre essa cadeia resulta em objeto da classe nltk. Grammar. ContextFreeGrammar, dado, por sua vez, como argumento do método de inicialização da classe nltk.ChartParser, que retorna um *parser* tabular ascendente. Esse *parser* é então aplicado sobre uma sentença e a representação arbórea produzida é exibida numa janela adicional.

Fica evidente o peso do léxico na construção de um *parser* quando calculamos a quantidade de entradas lexicais, elaboradas no formato de (18), necessárias para analisar um *corpus* de 100.000 palavras (constituído, por exemplo, de um conjunto de 10 artigos científicos, cada um com 10.000 palavras). Considerando um índice de diversidade lexical de 20% nesse *corpus*, precisaríamos de pelo menos 20.000 entradas lexicais para analisá-lo sintaticamente.

Mesmo dispondo de um léxico abrangente, que incluísse, sob a forma de regras no formato C -> d, todas as formas flexionadas dos verbetes de um dicionário frequentemente atualizado como iDicionário Aulete (2011), que já registra neologismos como *tucanar* e *lulismo*, um *parser* construído com base em uma CFG não analisaria sentenças como as seguintes (grifos nossos em todos os exemplos), que apresentam criações lexicais mais recentes, ainda não dicionarizadas:

(19) Dilma ainda não tucanou, mas muitos tucanos já **dilmaram** (DIL)

(20) […] os bravos tucanos "**Dilmaram**" [sic] por falta de liderança partidária. (RIB)

(21) O presidente do PT [...] afirmou que não existe "lulismo nem **dilmismo**". (VIA)

(22) Agora Micarla fez uma opção na sucessão presidencial. Ela se aproximou de Dilma Rousseff pessoalmente, [...] ela criou afinidades com o **dilmismo** que chega ao Estado com o PT. (DAN)

(23) Depois de quase dois anos fazendo um intensivão em **dilmês**, é natural que eu tenha um ouvido para **dilmices** mais apurado que a média. […] Mas até eu, **dilmista** de primeira hora, cheguei a ficar na dúvida […]. (ARA)

Outra dificuldade do processamento computacional de textos irrestritos é causada por desvios da norma culta ou por grafias não padrão, como nos seguintes exemplos:

(24) Uma vez um craqueiro veio me assaltar. Eu disse q [sic] só tinha uma oração pra oferecer p/ [sic] ele. (BAC)

(25) Craqueiro esfaqueado é preso por furto. […] a polícia acredita que a treta tenha rolado por acerto de contas do tráfico de drogas […] Ilton contou pros *homis* [sic] que caminhava pela rua quando foi abordado por um *dimaior* [sic] e outro *dimenor* [sic]. […] Mesmo ferido e sangrando, Ilton conseguiu *siscapar* [sic]. (TIA) [grifos no original)

(26) ESSA COMUNIDADE SURGIU PARA DESAFIAR O GRUPO DE ALMOFADINHAS BEM ARRUMADINHOS DO ROCK […] !!! NOS REFERIMOS AOS BEATLEMANIACOS MEMBROS DO P.D.R.B.C. […] !!! (GOM) [em caixa alta no original]

De modo a dar conta do problema que representa a modelação do léxico na construção de um *parser* abrangente do PB, em vez de construir milhares de entradas lexicais ou centenas de regras morfológicas, utilizamos etiquetadores livremente disponíveis, como o AeliusBRUBT, treinado por meio do NLTK no CHPTB (ALENCAR, 2010).

Aplicado aos exemplos de (4), (19), (21) e (23), o AeliusBRUBT produz o seguinte *output*, em que destacamos os erros de etiquetagem cometidos por meio do símbolo @ seguido da etiqueta correta:

(27) O/D deputado/N questionou/VB-D que/C o/D então/ADV governador/N também/ADV teria/TR-R liberado/VB-AN**@VB-PP** o/D deslocamento/N dos/P+D-P cantores/N-P ./.

(28) Dilma/N ainda/ADV não/NEG tucanou/VB-D ,/, mas/CONJ muitos/Q-P tucanos/ADJ-P**@N-P** já/ADV dilmaram/VB-D ./.

(29) O/D presidente/N do/P+D PT/NPR afirmou/VB-D que/C não/NEG existe/VB-P "/QT lulismo/N nem/CONJ-NEG dilmismo/N "/QT ./.

(30) Depois/ADV de/P quase/ADV dois/NUM anos/N-P fazendo/VB-G um/D-UM intensivão/N em/P dilmês/N-P**@N** ,/, é/SR-P natural/ADJ-G que/C eu/PRO tenha/TR-SP um/D-UM ouvido/N para/P dilmices/N-P mais/ADV-R apurado/VB-AN que/C a/D-F média/ADJ-F**@N** ./. Mas/CONJ até/FP eu/PRO ,/, dilmista/N de/D-F primeira/ADJ-F hora/N ,/, cheguei/VB-P a/P ficar/VB na/P+D-F dúvida/N ./.

O primeiro erro do AeliusBRUBT na etiquetagem das sentenças de (27) a (30) resultou da classificação da forma *liberado* em (4) como particípio adjetival (VB-AN), em vez de particípio verbal (VB-PP). Enquanto o primeiro funciona como núcleo de sintagma adjetival, o último exerce o papel de núcleo de um sintagma verbal. Os demais erros decorreram da falsa classificação de adjetivos e substantivos. Na próxima seção, retomaremos a questão dos erros da etiquetagem automática no contexto do *parsing* de textos irrestritos.

Para a construção das regras de estrutura sintagmática, compilamos um conjunto de 36 sentenças formado de exemplos coletados na Internet, como os de (19) a (25), que apresentam grafias não padrão, nomes próprios, siglas ou neologismos não dicionarizados, incluindo outras sentenças com estruturas paralelas às extraídas da WWW, mas com variações de vocabulário.

(31)

i. Pesquisa melhora cultivo de células estaminais.
ii. Micarla fez uma opção na sucessão presidencial.
iii. Ela se aproximou de Dilma.
iv. Ela criou afinidades com o dilmismo.
v. Dilma ainda não tucanou.
vi. O prefeito tucanou.
vii. O nosso prefeito tucanou.
viii. Muitos tucanos já dilmaram.
ix. Muitos vereadores tucanos já dilmaram.
x. 16 prefeitos do PSDB dilmaram.

xi. 16 membros do PSDB dilmaram.
xii. Os prefeitos do PSDB da região dilmaram.
xiii. Os vereadores do PSDB da região dilmaram.
xiv. Os dilmistas vão votar no PT.
xv. Os bravos tucanos dilmaram por falta de liderança partidária.
xvi. Os bravos vereadores dilmaram por falta de liderança partidária.
xvii. Eles dilmaram por falta de liderança partidária.
xviii. O jornal noticiou a dilmação das favelas.
xix. O meu jornal noticiou a dilmação das suas favelas.
xx. O jornalista divulgou que os vereadores da cidade do milionário também tinham dilmado as favelas.
xxi. O médico disse que o poeta teria autorizado o pagamento do cheque.
xxii. O deputado questionou que o então governador também teria liberado o deslocamento dos cantores.
xxiii. O dilmista fez um intensivão em dilmês.
xxiv. Ele tem um ouvido apurado para dilmices.
xxv. O presidente do PT afirmou que não existe dilmismo.
xxvi. O presidente do PT afirmou que o dilmismo não existe.
xxvii. O escritor dilmista fez um intensivão em dilmês.
xxviii. Nos referimos aos beatlemaníacos membros do PDRBC.
xxix. A periguete esforçou-se para agradar o cleptopolítico durante o lulalato.
xxx. Craqueiro esfaqueado é preso por furto.
xxxi. A polícia acredita que a treta tenha rolado por acerto de contas do tráfico de drogas.
xxxii. Ilton contou pros homis que foi abordado por um dimenor.
xxxiii. Ilton conseguiu siscapar.
xxxiv. Uma vez um craqueiro veio me assaltar.
xxxv. Eu disse que só tinha uma oração pra oferecer pra ele.
xxxvi. Eu disse q só tinha uma oração pra oferecer p/ ele.

Na fase inicial de desenvolvimento de nossa gramática, restringimo-nos a sentenças declarativas sem estruturas de coordenação, mas incluindo construções com verbos auxiliares e a complementação verbal por meio de orações não finitas ou introduzidas pelo complementador (C) *que*. A estrutura do sintagma nominal é relativamente complexa, incluindo numerais e quantificadores (QUANT), partículas de foco (FP), possessivos (POSS) e sintagmas adjetivais e adverbiais (respectivamente AP e ADVP) como modificadores.

Uma limitação desse modelo inicial é que, da teoria X-barra, implementamos diferentes níveis de projeção apenas para algumas categorias, sem adotar uma ramificação estritamente binária. Outros princípios da teoria X-barra como a endocentricidade e a maximalidade também não foram considerados de forma estrita. Deixamos para uma próxima fase a implementação de um fragmento mais amplo do PB seguindo estritamente esse modelo sintático, incluindo desdobramentos da teoria X-barra clássica, como as hipóteses do DP, do sujeito interno ao VP, do *split-Infl* etc. (GREWENDORF, 2002).

No formalismo da CFG do NLTK, o único recurso de que dispomos para simplificar a elaboração das regras da gramática é o operador de disjunção lógica "|", cuja aplicação na minigramática (18) resulta na seguinte versão mais compacta:

(32) Minigramática II – versão compacta
S -> NP VP
NP -> Det N
VP -> V | V PP
PP -> P NP
Det -> 'os' | 'o'
N -> 'partidos' | 'partido' | 'escândalos'
V -> 'silenciaram' | 'silenciou'

Em manuais introdutórios de sintaxe gerativa, utilizam-se operadores que permitem simplificar ainda mais as regras de uma CFG, ao mesmo tempo em que tornam o formalismo mais expressivo. Os seguintes exemplos foram extraídos, respectivamente, de Carnie (2002, p. 41) e Sag, Wasow e Bender (2003, p. 27):

(33)
S -> {NP/S'} (T) VP
NP -> (D) (AP+) N (PP+)

(34)
NP -> (D) A* N PP*

A notação D -> {A/B/C} equivale a D -> A, D -> B e D -> C. Os parênteses indicam a opcionalidade de uma categoria. Logo, D -> (B) C equivale a D -> C e D -> B C. Os símbolos "*" e "+" funcionam como multiplicadores que assumem qualquer número natural como valor, começando, respectivamente, em

0 e 1. Dessa forma, uma regra como A -> B C*, onde C representa uma categoria individual ou uma disjunção como {E/F}, equivale a A -> B, A -> B C, A -> B C C, A -> B C C C etc. A regra A -> B C+ expande-se de forma análoga, com exceção da regra A -> B, pois a categoria C deve ocorrer pelo menos uma vez.

Infelizmente, o NLTK não suporta regras no formato de (33). A fim de poder usufruir das vantagens desses recursos notacionais, construímos módulo em Python, denominado c2n, que converte essa notação na do NLTK.[46] Desse modo, nosso modelo da estrutura sintagmática das sentenças de (31) resume-se nas 14 regras seguintes.[47]

(35) Gramática cfg.syn (parte I)

```
S -> (NP) VP
S -> {ADVP/AP/PP} S
S' -> C S
NP -> (DET) (POSS) (QUANT) {ADVP/AP}* NOM {PP/AP}*
NP -> NP FP
NP -> FP NP
NP -> {PRO/DEM}
INFP -> (P) INF' {NP/PP/ADVP/AP}* (S') (INFP)
INF' -> (FP) (SE) (CL) INF
VP -> ADVP* V' {NP/PP/ADVP/AP}* (S') (INFP)
V' -> (NEG) (FP) (SE) (CL) V (SE) (CL)
PP -> {P/PA} NP
AP -> (ADVP) A
ADVP -> ADV
```

Além dessas regras, a gramática de nosso protótipo contém ao todo 17 regras como as de (36), que expandem as categorias lexicais de (35) nas categorias morfossintáticas correspondentes do CHPTB, quando as primeiras comportam mais de um membro nesse *corpus*.[48] Por exemplo, as categorias NEG, C, FP, P, SE e CL são pré-terminais, não sendo passíveis, portanto, de expansão senão por um terminal (i.e. um item lexical), diferentemente de categorias como A, NOM, DET, INF e V, subclassificadas conforme os traços flexionais de número, gênero, modo, tempo, etc.[49]

(36) Gramática cfg.syn (parte II – fragmento)
A -> {ADJ/ADJ_P/ADJ_F/ADJ_F_P/ADJ_G/ADJ_G_P}
A -> {VB_AN/VB_AN_P/VB_AN_F/VB_AN_F_P}
NOM -> {N/N_P/NPR/NPR_P}
DET -> {D/D_P/D_F/D_F_P/D_UM/D_UM_F}
INF -> {VB_PP/VP_G/VB/VB_F}

Como ressaltamos, a gramática não contém regras do tipo C -> d, que introduzem os itens lexicais. Essas regras são construídas com base no resultado da aplicação de etiquetadores na etiquetagem das sentenças dadas para análise pelo ALEXP, como veremos na seção seguinte.

5. Funcionamento e avaliação do ALEXP

Esta seção explica o funcionamento do ALEXP e apresenta os resultados de uma avaliação preliminar de diversos *parsers* construídos no NLTK por meio da ferramenta, tomando como ponto de partida o protótipo de gramática da seção anterior e utilizando, como analisadores lexicais, diversos etiquetadores livremente disponíveis. Inicialmente, discutimos o desempenho de *parsers* construídos com a gramática cfg.syn (ver exemplos (35) e (36)) e as sentenças (31), etiquetadas primeiro pelo AeliusBRUBT e, depois, pelo AeliusHunPos. Verificamos também o impacto, na análise sintática automática dessas sentenças, de algumas alterações nessa gramática. Finalmente, avaliamos o desempenho de *parsers* construídos com versões adaptadas dessas gramáticas e o *output* do etiquetador LX-Tagger.

Como vimos, o NLTK não suporta o formato compacto de descrição da sintaxe de uma língua natural por meio da CFG, exemplificado em (33). Para construir um *parser* do NLTK com base na gramática cfg.syn, é necessário expandir as regras que contêm os operadores de multiplicação "+", "(", ")" e "*" e os de disjunção "{", "}" e "/", transformando-as em regras no formato do NLTK. Fazer essa conversão manualmente seria um processo bastante penoso e sujeito a falhas. Em vez disso, convertemos um formato em outro de modo automático por meio do módulo c2n. Em seguida, para compilação do *parser*, utilizamos o módulo ALEXP (acrônimo de *analisador léxico-sintático do português*), que construímos especialmente para possibilitar a integração da etiquetagem morfossintática como componente lexical da análise sintática automática.

Como o número de regras de uma CFG precisa ser finito, é necessário no c2n definir o valor máximo dos multiplicadores "*" e "+", o que é feito pela variável global MAXIMO, cujo valor *default* é 3, passível de alteração pelo usuário.

Por meio do c2n, compilamos uma versão da gramática de (35) e (36) no formato do NLTK mantendo 3 como valor máximo dos multiplicadores "*" e "+":

(37)

```
>>> import c2n
>>> c2n.escreve_gramatica("cfg.syn")
NLTK Grammar successfully processed and saved in cfg.syn.nltk
```

A gramática no formato do NLTK gerada pelo c2n, denominada cfg.syn.nltk, é dada então como *input* para o ALEXP (FIG. 16). Esse módulo dispõe de uma função por meio da qual podemos construir um *parser* com base em uma gramática CFG no formato do NLTK e do vocabulário que é extraído de um conjunto de sentenças, que são etiquetadas pelo Aelius, acionado automaticamente pelo ALEXP. Caso o parâmetro predefinido ETIQUETADOR não seja alterado, a etiquetagem é feita pelo AeliusBRUBT, como no exemplo da FIG. 16. Após aplicação do *parser* às sentenças do arquivo sentencas.txt, é exibida para cada uma a quantidade de análises geradas em formato de árvores. Uma versão desse arquivo com as sentenças anotadas (nomeado sentencas.nltk.txt) é criada no diretório de trabalho.

```
>>> alexp.analisaBlocoDeSentencas("gramaticas/cfg.syn.nltk",nao_anotado="sentencas.txt")
Arquivo anotado:
sentencas.nltk.txt
Nr. 2: 5 análise(s): Micarla fez uma opção na sucessão presidencial
Nr. 3: 1 análise(s): Ela se aproximou de Dilma
Nr. 4: 2 análise(s): Ela criou afinidades com o dilmismo
Nr. 5: 1 análise(s): Dilma ainda não tucanou
Nr. 6: 1 análise(s): O prefeito tucanou
Nr. 7: 1 análise(s): O nosso prefeito tucanou
Nr. 9: 1 análise(s): Muitos vereadores tucanos já dilmaram
Nr. 11: 1 análise(s): 16 membros do PSDB dilmaram
Nr. 13: 2 análise(s): Os vereadores do PSDB da região dilmaram
Nr. 14: 1 análise(s): Os dilmistas vão votar no PT
Nr. 16: 5 análise(s): Os bravos vereadores dilmaram por falta de liderança partidária
Nr. 17: 5 análise(s): Eles dilmaram por falta de liderança partidária
Nr. 18: 4 análise(s): O jornal noticiou a dilmação das favelas
Nr. 19: 4 análise(s): O meu jornal noticiou a dilmação das suas favelas
Nr. 20: 4 análise(s): O jornalista divulgou que os vereadores da cidade do milionário também tinham dilmado as favelas
Nr. 21: 4 análise(s): O médico disse que o poeta teria autorizado o pagamento do cheque
Nr. 22: 4 análise(s): O deputado questionou que o então governador também teria liberado o deslocamento dos cantores
Nr. 23: 2 análise(s): O dilmista fez um intensivão em dilmês
Nr. 24: 3 análise(s): Ele tem um ouvido apurado para dilmices
Nr. 25: 1 análise(s): O presidente do PT afirmou que não existe dilmismo
Nr. 26: 1 análise(s): O presidente do PT afirmou que o dilmismo não existe
Nr. 28: 2 análise(s): Nos referimos aos beatlemaníacos membros do PDRBC
Nr. 30: 1 análise(s): Craqueiro esfaqueado é preso por furto
Nr. 31: 52 análise(s): A polícia acredita que a treta tenha rolado por acerto de contas do tráfico de drogas
Nr. 32: 1 análise(s): Ilton contou pros homis que foi abordado por um dimenor
Nr. 33: 1 análise(s): Ilton conseguiu siscapar
Nr. 35: 2 análise(s): Eu disse que só tinha uma oração pra oferecer pra ele

75.00% analisadas de um total de 36 sentenças
4.15 análise(s) por sentença
```

FIGURA 16: Análise das sentenças de (31) conforme a gramática cfg.syn.nltk

Como podemos constatar na FIG. 16, o *parser* construído com base na gramática cfg.syn.nltk e nas sentenças etiquetadas pelo AeliusBRUBT analisou 75% das 36 sentenças de (31). A FIG. 17 traz as sentenças não analisadas.

Consideramos razoável esse desempenho inicial, em se tratando de um protótipo aplicado na análise de sentenças não triviais. Ressalte-se que a média de análises por sentença analisada, de 4.15, foi bem mais baixa que a relatada por Almeida *et al.* (2003).

```
Nr. 1: 0 análise(s): Pesquisa melhora cultivo de células estaminais
Nr. 8: 0 análise(s): Muitos tucanos já dilmaram
Nr. 10: 0 análise(s): 16 prefeitos do PSDB dilmaram
Nr. 12: 0 análise(s): Os prefeitos do PSDB da região dilmaram
Nr. 15: 0 análise(s): Os bravos tucanos dilmaram por falta de liderança partidária
Nr. 27: 0 análise(s): O escritor dilmista fez um intensivão em dilmês
Nr. 29: 0 análise(s): A periguete esforçou se para agradar o cleptopolítico durante o lulalato
Nr. 34: 0 análise(s): Uma vez um craqueiro veio me assaltar
Nr. 36: 0 análise(s): Eu disse q só tinha uma oração pra oferecer p/ ele
```

FIGURA 17: Sentenças sem análise pelo *parser* da gramática gerada com base em cfg.syn.nltk e a etiquetagem das sentenças pelo AeliusBRUBT

O ALEXP oferece mais diversos recursos, numa interface amigável, para análise sintática automática com gramáticas de estrutura sintagmática e etiquetadores morfossintáticos funcionando como analisadores lexicais. Por exemplo, uma vez construído um *parser* como exemplificado na FIG. 16, podemos explorar os resultados da análise das sentenças por meio da exibição gráfica das árvores geradas. Para produzir a árvore da FIG. 18, executamos o comando (38). De modo análogo foram geradas a FIG. 19 e a FIG. 20, que contrastam quanto ao comportamento do reflexivo, sintaticamente autônomo (apesar de clítico) no primeiro exemplo, incorporado ao verbo no segundo. Essa última sentença não foi analisada corretamente nem pelo VISL nem pelo LX-Tagger, que classificaram a forma contraída não padrão *siscapar* como substantivo.[50]

(38)

```
>>> alexp.mostraArvores(9)
```

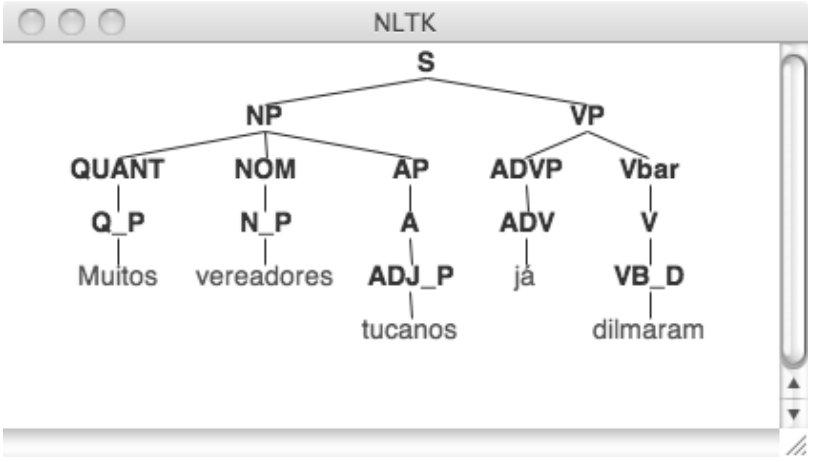

FIGURA 18: Análise de (31) (ix)

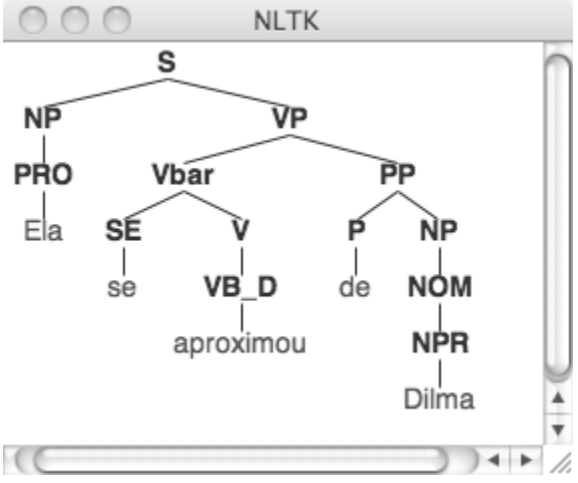

FIGURA 19: Análise de (31) (iii)

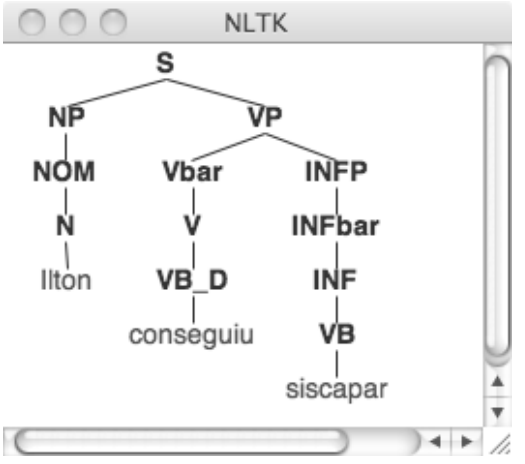

FIGURA 20: Análise de (31) (xxxiii)

Na FIG. 21, exemplificamos a extração, de modo interativo, de algumas propriedades do *parser* tabular ascendente gerado pelo ALEXP na FIG. 16. Como podemos ver, a gramática desse *parser* tem 4171 produções, das quais 152 são lexicais. As produções lexicais foram geradas automaticamente pelo ALEXP por meio da etiquetagem morfossintática, pelo AeliusBRUBT, das sentenças dadas para análise. As demais 4019 produções foram geradas pelo c2n a partir da gramática cfg.syn. Os dois últimos comandos mostram como extrair as entradas lexicais de uma determinada categoria lexical, no caso, o particípio de tempos compostos.

```
>>> gramatica,parser=alexp.ANALISADORES[0]
>>> gramatica
'gramaticas/cfg.syn.nltk'
>>> parser
<nltk.parse.chart.ChartParser object at 0x368b050>
>>> cfg=parser.grammar()
>>> cfg
<Grammar with 4171 productions>
>>> lexico=[p for p in cfg.productions() if p.is_lexical()]
>>> len(lexico)
152
>>> participios=[p for p in lexico if p.lhs().symbol() == "VB_PP"]
>>> for p in participios:
        print p


VB_PP -> u'liberado'
VB_PP -> u'rolado'
VB_PP -> u'dilmado'
VB_PP -> u'autorizado'
>>>
```

FIGURA 21: Exploração de propriedades do *parser* gerado a partir da gramática cfg.syn.nltk e da etiquetagem das sentenças (31)

Na FIG. 16, utilizamos o ALEXP para analisar, de uma só vez, um conjunto de sentenças armazenadas em um arquivo. Vejamos agora como analisar uma sentença individual por meio do ALEXP. Consideramos a sentença (39), que instancia, conforme a gramática cfg.syn (ver (35)), a regra recursiva à esquerda NP -> NP FP.

(39) Ela mesmo dilmou.[51]

Os comandos em (40) mostram como analisar automaticamente essa sentença. Primeiro, importamos o módulo AnotaCorpus do Aelius, cujo toquenizador armazenamos na variável t. Em seguida, após atribuir a uma variável s1, sob a forma de cadeia unicode, a sentença que pretendemos analisar, aplicamos a essa cadeia a função analisaSentenca do ALEXP, cujo *output* é uma lista de árvores. Finalmente, exibimos essas árvores (FIG. 22).

(40)

```
>>> from Aelius import AnotaCorpus
>>> t=AnotaCorpus.TokPort.tokenize
>>> s1="Ela mesmo dilmou".decode("utf-8")
>>> a=alexp.analisaSentenca(t(s1),"gramaticas/cfg.syn.nltk")
>>> alexp.exibeArvores(a)
```

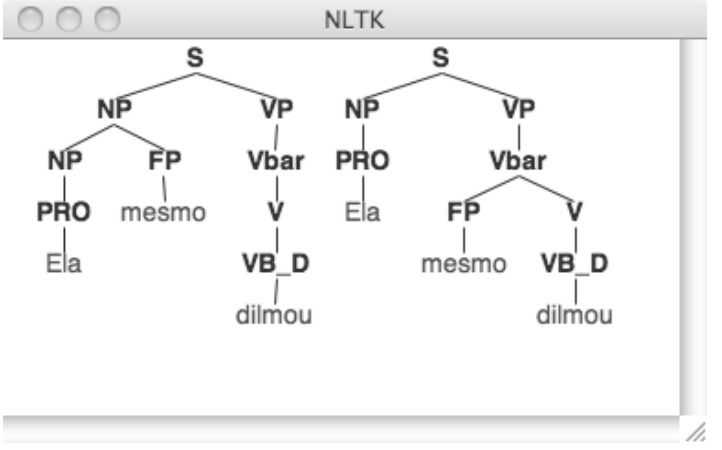

FIGURA 22: Processamento de regra recursiva à direita na análise do exemplo (39) conforme a gramática cfg.syn

Como podemos constatar na FIG. 22, a regra recursiva à esquerda NP -> NP FP da gramática cfg.syn (ver (35)) não apresenta problema algum para o algoritmo de *parsing* tabular ascendente do NLTK, ao contrário do que ocorre com o algoritmo recursivo descendente de Prolog no processamento de DCGs.

Também a regra recursiva à direita NP -> FP NP, instanciada no exemplo (41), é processada pelo *parser*, como evidencia a FIG. 23.

(41) Mesmo ela dilmou.

FIGURA 23: Análises do exemplo (41) a partir da etiquetagem automática e da etiquetagem manual

Nesse caso, o etiquetador AeliusBRUBT erroneamente atribuiu a *mesmo* a etiqueta ADJ, como podemos ver na representação da estrutura de constituintes sob a forma de parênteses rotulados. Quando, na construção do *parser*, utilizamos uma versão anotada manualmente da sentença (armazenada no arquivo exemplo.txt), obtém-se a árvore esperada, exibida em formato gráfico no alto da FIG. 23.

Um exame das análises da FIG. 22 bem como de outras produzidas pelo *parser* apontou para problemas na formulação de algumas regras da gramática cfg.syn.nltk. A FIG. 22 expõe uma assimetria indesejável no tratamento da categoria FP, que é adjunto de NP, mas irmã do núcleo INF. Um outro problema são as ambiguidades espúrias induzidas por algumas regras, como a seguinte:

(42) VP -> ADVP* V' {NP/PP/ADVP/AP}* (S') (INFP)

A ambiguidade estrutural é um problema onipresente no *parsing* das línguas naturais (JURAFSKY; MARTIN, 2009, p. 467). O caso clássico envolve a adjunção de PP (*PP-attachment*), fenômeno prototipicamente exemplificado na sentença (31) (xxiii) (FIG. 24), bem como nos exemplos (31) (iv) e (31) (xiii).

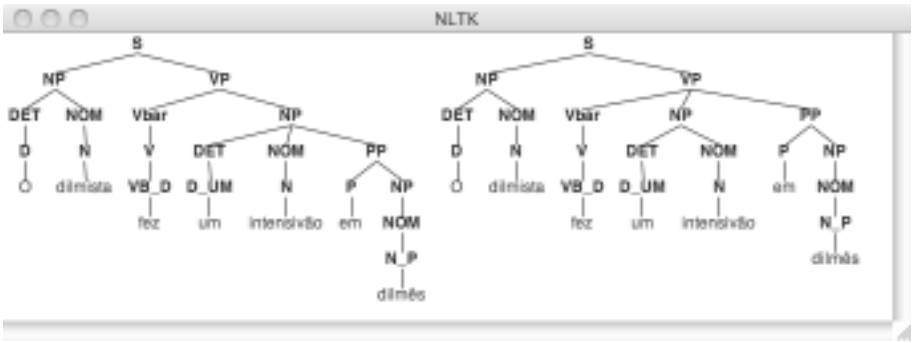

FIGURA 24: Ambiguidade da sentença (31) (xxiii)

Para dirimir ambiguidades como a da FIG. 24, é preciso recorrer a traços gramaticais dos itens lexicais como a subcategorização (capaz de resolver, em favor da árvore à esquerda, a ambiguidade de (31) (xxiii), pois o PP é complemento nominal) bem como a informações semânticas e pragmáticas ou estatísticas (LEMNITZER; WAGNER, 2004, p. 246-247, 256-258).

Por outro lado, a ambiguidade do tipo espúrio, como a gerada pela regra (42), não reflete uma propriedade estrutural da sintaxe da língua portuguesa, mas decorre de um modelagem formal inadequada. Com uma versão da gramática que desdobra essa regra nas três seguintes,[52] conseguimos reduzir a ambiguidade para 2.93, e a quantidade de análises para a sentença (31) (xxxi) diminuiu pela metade:

(43) gramática cfg02.syn (fragmento)
VP -> ADVP* V' {NP/PP/ADVP/AP}*
VP -> ADVP* V' {NP/PP/ADVP/AP}* S'
VP -> ADVP* V' {NP/PP/ADVP/AP}* INFP

Com exceção da sentença (31) (xxxiv), corretamente etiquetada, as sentenças que não foram reconhecidas pelo *parser* apresentaram erros de etiquetagem. Por exemplo, o AeliusBRUBT atribuiu a *tucanos* e *prefeitos*, erroneamente, a etiqueta ADJ-P em vez de N-P. Outro problema foram as grafias não padrão *q* e *p/* em (31) (xxxvi).

Um erro de etiquetagem levou o *parser* a atribuir à sentença (31) (xxxii) uma análise linguisticamente incorreta. De fato, a contração não padrão *pros* foi erroneamente etiquetada como N-P (substantivo no plural) pelo AeliusBRUBT (FIG. 25).

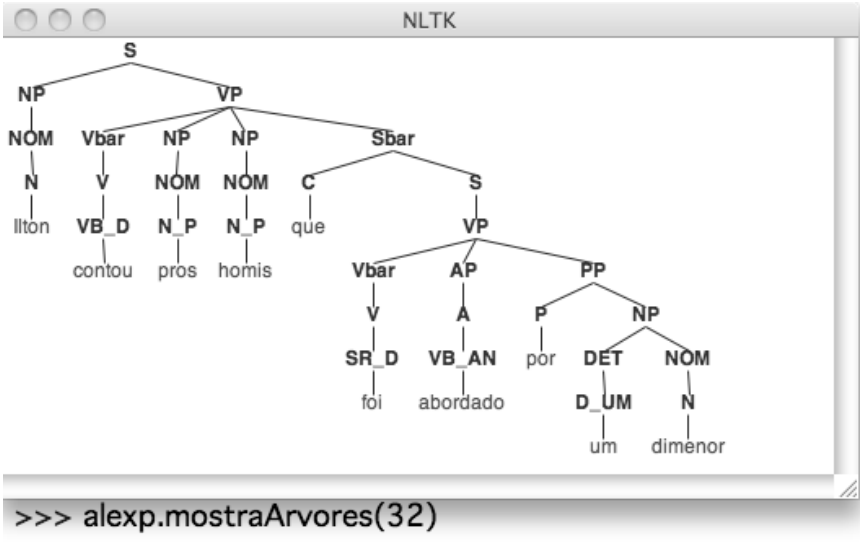

FIGURA 25: Análise da sentença (31) (xxxii)

Já o problema de (31) (xxxiv) foi a falta de cobertura da gramática, que não contempla um NP topicalizado funcionando como adverbial, mas apenas ADVP, AP e PP:

(44) gramática cfg.syn (fragmento) (ver (35))
S -> {ADVP/AP/PP} S

Incluindo um NP na disjunção do lado direito dessa regra (ver (45)), elevamos o percentual de sentenças analisadas para 80.56%, mas a ambiguidade aumenta consideravelmente, atingindo a média de 8.14 análises por sentença analisada.

(45) gramática cfg03.syn (fragmento)
S -> {ADVP/AP/PP/NP} S

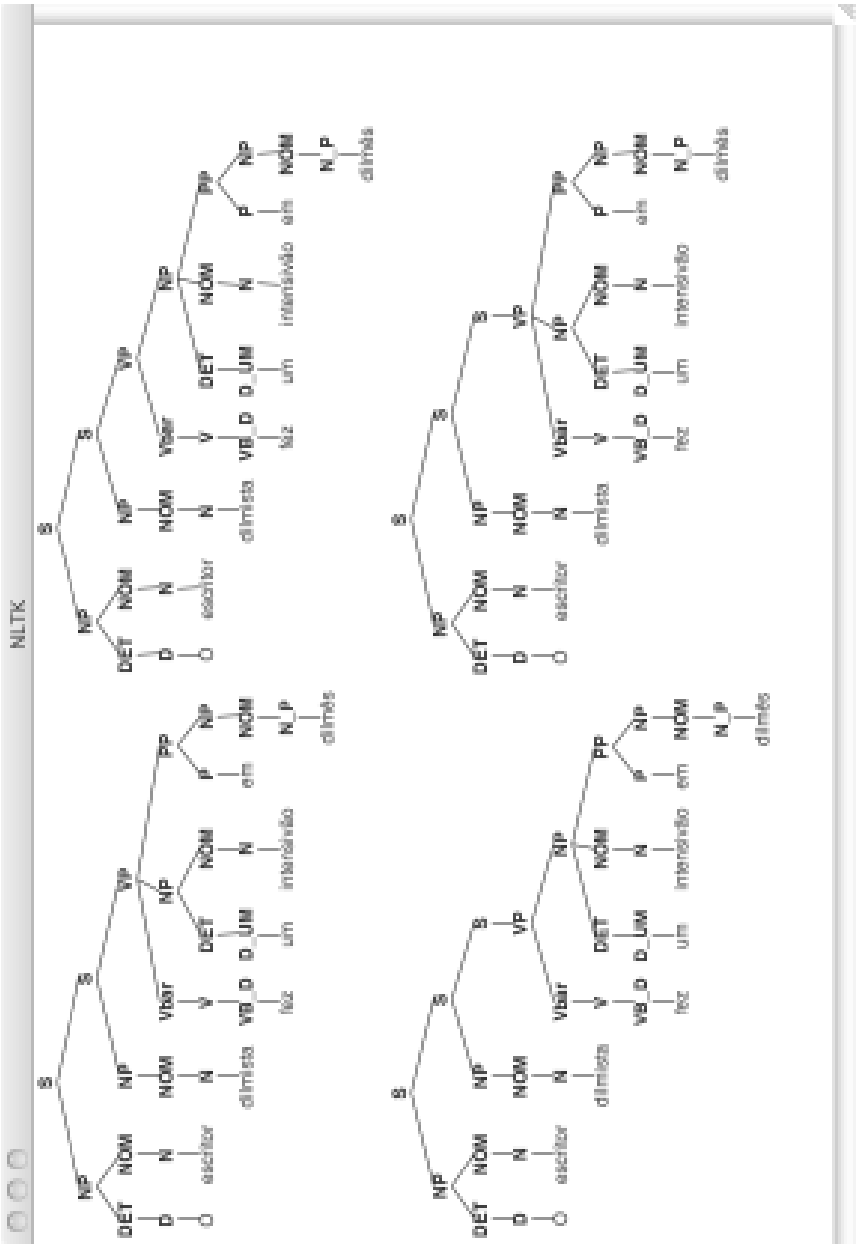

FIGURA 26: Análises da sentença (31) (xxvii) conforme a gramática cfg03.syn

A sentença (31) (xxvii), que deixou de ser analisada pelo *parser* da gramática cfg.syn porque o etiquetador AeliusBRUBT não classificou *dilmista* como adjetivo, mas como substantivo, passa a ser analisada, junto com a sentença (31) (xxxiv), pelo *parser* da gramática cfg03.syn. No entanto, nenhuma das quatro análises atribuídas à sentença é a esperada, pois apresenta o NP *o escritor* sempre em adjunção a S e *dilmista* como núcleo de um outro NP e não de um AP modificador do primeiro NP (FIG. 26).

Uma consequência positiva da regra (45), introduzida na gramática cfg03.syn para dar conta de NPs topicalizados em adjunção a S, é que o exemplo (1), não contemplado pelo Selva (ALMEIDA *et al*., 2003), passa a ser analisado por um *parser* construído com base nessa gramática e nos *tokens* dessa sentença etiquetados pelo AeliusBRUBT. Na FIG. 27, a árvore esperada é a da direita.

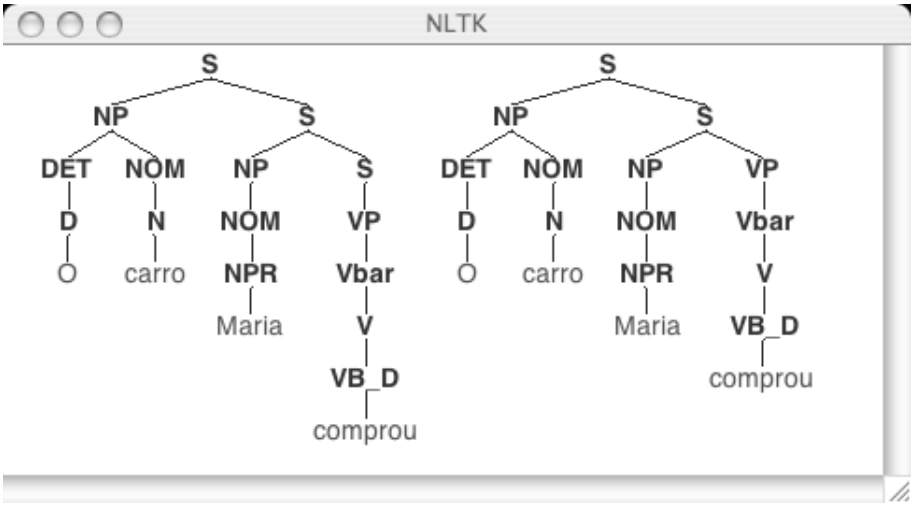

FIGURA 27: Análise da sentença (1) conforme a gramática cfg03.syn

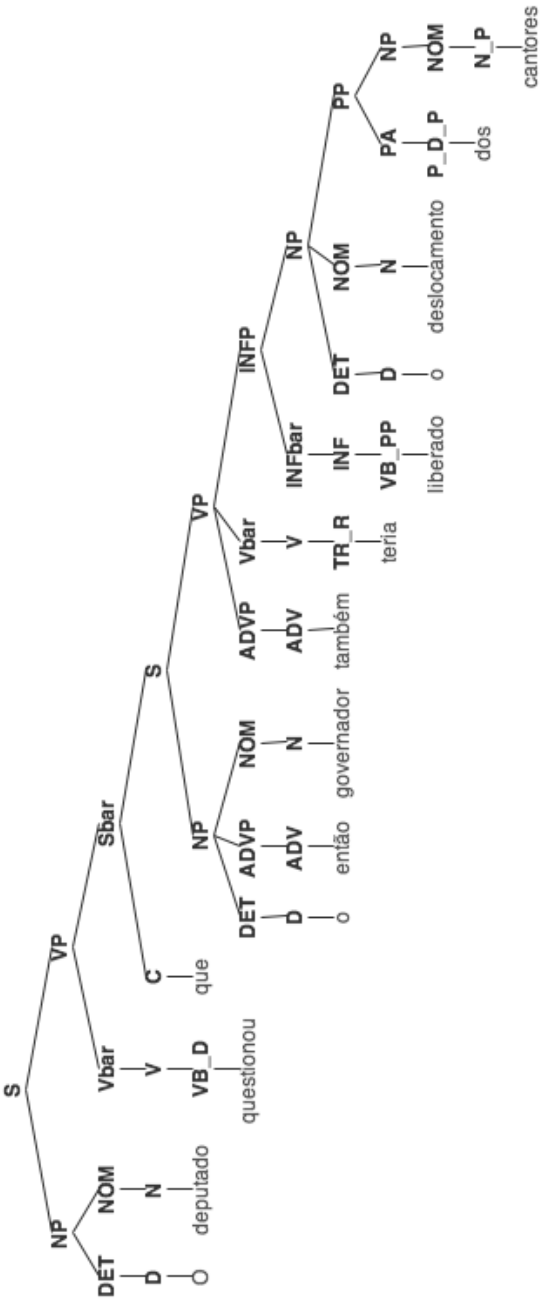

FIGURA 28: Análise da sentença (31) (xxii)

Observamos na FIG. 28 que a sentença (31) (xxii) foi analisada corretamente, apesar de o AeliusBRUBT etiquetá-la de forma errônea, atribuindo ao particípio a etiqueta VB-AN em vez de VB-PP, como vimos no exemplo (27). No entanto, a FIG. 28, produzida pelo ALEXP, exibe o particípio com a etiqueta correta. Como se explica isso? A razão é simples: no ALEXP, implementamos função que corrige alguns dos erros do Aelius.

Vejamos agora que impacto na qualidade do *parsing* resulta da utilização dos etiquetadores AeliusHunPos e LX-Tagger. Para tanto, inicialmente comparamos, por meio dos comandos em (46), os desempenhos desses etiquetadores, na etiquetagem das sentenças de (31), com o do AeliusBRUBT. Como podemos constatar, ganha-se quase 1% de acurácia ao utilizar o AeliusHunPos em vez do AeliusBRUBT. Um acréscimo adicional de pouco menos de 1% na acurácia da etiquetagem é obtido por meio do LX-Tagger.

(46)

```
>>> from Aelius.Avalia import testa_etiquetador as avalia
>>> lx="UTF-8_Model_Cintil_Written"
>>> avalia("AeliusBRUBT.pkl","nltk",ouro="sents.chptb.gold.txt")
Total de erros: 17
Total de palavras:316
Acurácia:94.620253
>>> avalia("pt-model.hunpos","hunpos",ouro="sents.chptb.gold.txt")
Total de erros: 14
Total de palavras:316
Acurácia:95.569620
>>> avalia(lx,"mxpost",ouro="sents.lxtagger.gold.txt")
Total de erros: 12
Total de palavras:334
Acurácia:96.407186
```

A tabela 1 e a tabela 2 permitem comparar o desempenho do AeliusBRUBT e do AeliusHunPos na análise sintática automática. Em todas as três gramáticas, o desempenho do AeliusHunPos se sobressai. Com a gramática cfg03.syn, esse etiquetador supera o desempenho do primeiro em mais de 10%.

TABELA 1
Desempenho do etiquetador AeliusBRUBT como analisador lexical na análise sintática automática

Gramática	Percentual de sentenças analisadas	Análises por sentença
cfg.syn	75,00%	4,15
cfg02.syn	75,00%	2,93
cfg03.syn	80,56%	8,14

TABELA 2
Desempenho do etiquetador AeliusHunPos como analisador lexical na análise sintática automática

Gramática	Percentual de sentenças analisadas	Análises por sentença
cfg.syn	88,89%	2,28
cfg02.syn	88,89%	2,31
cfg03.syn	91,67%	6,55

A utilização do LX-Tagger como analisador lexical no ALEXP exige algumas modificações na gramática cfg.syn, uma vez que esse etiquetador utiliza um conjunto de etiquetas bastante diferente daquele do CHPTB, base do AeliusBRUBT e do AeliusHunPos. Há, por um lado, distinções que o CHPTB estabelece que não são feitas pelo LX-Tagger, como é o caso da categoria FP (partícula de foco). Por outro lado, apenas o LX-Tagger distingue entre verbos principais e auxiliares. A etiqueta VAUX é aplicada por esse etiquetador aos verbos *ter* e *haver* em tempos compostos. No CHPTB, esses dois verbos recebem sempre as etiquetas TER e HV, respectivamente, mesmo quando funcionam como verbos principais. Outra distinção importante é que a etiquetagem do LX-Tagger não contempla os traços morfológicos das categorias lexicais flexionáveis.[53] Desse modo, formas finitas do CHPTB, como VB-P, VB-D, VB-R, VB-SP etc. são uniformemente etiquetadas pelo LX-Tagger como V.

A fim de facilitar a comparação entre análises realizadas por *parsers* baseados no AeliusBRUBT e no AeliusHunPos, por um lado, e no LX-Tagger, por outro, recorremos a uma função do Aelius que converte uma anotação nos moldes do LX-Tagger em uma anotação baseada numa versão simplificada do conjunto de etiquetas do CHPTB. A utilização dessa função, que, entre outros mapeamentos,

converte a etiqueta V do LX-Tagger em V-FIN (uma etiqueta "guarda-chuva" para as etiquetas de verbos finitos no CHPTB), é exemplificada na FIG. 29.

A FIG. 29 exemplifica também como anotar uma sentença pelo LX-Tagger a partir do Aelius (compare com a FIG. 13). Esse pacote em Python oferece uma interface mais amigável que a linha de comando do Unix, necessária para executar os *scripts* que integram a distribuição do LX-Tagger. Uma outra vantagem de utilizar o Aelius para etiquetar, por meio do LX-Tagger, sentenças individuais ou mesmo anotar morfossintaticamente textos inteiros é que a toquenização é feita pelo próprio Aelius, sendo implementada em Python e, portanto, multiplataforma. A toquenização original do LX-Tagger, pelo contrário, recorre a um script da *shell* do Unix.

```
>>> sent12=“Os prefeitos do PSDB da região dilmaram.”.decode(“utf8”)
>>> from Aelius import carrega,SimplificaEtiquetas, AnotaCorpus
>>> t=AnotaCorpus.TokPort.tokenize(sent12)
>>> t
[u'Os', u'prefeitos', u'do', u'PSDB', u'da', u'regi\xe3o', u'dilmaram', u'.']
>>> lx=carrega(“lxtagger”)
>>> AnotaCorpus.anota_sentencas([t],lx,“mxpost”)
[[('Os', 'DA'), ('prefeitos', 'CN'), ('de', 'PREP'), ('o', 'DA'), ('PSDB', 'PNM'), ('de', 'PREP'), ('a', 'DA'), ('regi\xc3\xa3o', 'CN'), ('dil
maram', 'V'), ('.', 'PNT')]]
>>> pos1=AnotaCorpus.anota_sentencas([t],lx,“mxpost”)[0]
>>> simplifica=SimplificaEtiquetas.LXTagger2CHPTB
>>> pos2=[(p,simplifica(e)) for p,e in pos1]
>>> pos2
[('Os', 'D'), ('prefeitos', 'N'), ('de', 'P'), ('o', 'D'), ('PSDB', 'NPR'), ('de', 'P'), ('a', 'D'), ('regi\xc3\xa3o', 'N'), ('dilmaram', 'VB-FIN'
), ('.', '.')]
>>>
```

FIGURA 29: Anotação de (31) (xii) pelo LX-Tagger a partir do Aelius e conversão automática em formato simplificado do CHPTB

Observe que o LX-Tagger, bem mais preciso que o AeliusBRUBT, não comete nenhum erro na etiquetagem da sentença (31) (xii). Vimos que esse último etiquetador atribuiu a *prefeitos*, erroneamente, a etiqueta ADJ, o que impediu a construção, pelo *parser*, de uma análise para a sentença. A distinção entre substantivos e adjetivos, que representa uma das maiores dificuldades para o AeliusBRUBT, é resolvida pelo LX-Tagger mesmo em casos não triviais como os das FIG. 13 e 14. Esse último etiquetador também se sai melhor que o VISL e o LX-Parser na análise da sentença (6), cometendo um único erro, que é classificar o substantivo inicial como nome próprio.

No último comando da FIG. 29, exibe-se o resultado da conversão do sistema de anotação do LX-Tagger para a versão simplificada do sistema do CHPTB, por meio da função LXTagger2CHPTB do Aelius.

Graças a essa função, para construir um *parser* que utilize o *output* do LX-Tagger, precisamos fazer apenas poucas adaptações na gramática cfg.syn. Eis o resultado da adaptação dessa gramática para ser usada com o LX-Tagger no ALEXP:

(47) gramática cfg.lx.syn (Parte I)

```
S -> (NP) VP
S -> {ADVP/AP/PP} S
S' -> C S
NP -> (DET) (POSS) (QUANT) {ADVP/AP}* NOM {PP/AP}*
NP -> {PRO/DEM}
INFP -> (P) INF' {NP/PP/ADVP/AP}* (S') (INFP)
INF' -> (CL) INF
VP -> ADVP* V' {NP/PP/ADVP/AP}* (S') (INFP)
V' -> (CL) V (CL)
PP -> P NP
AP -> (ADVP) A
ADVP -> ADV
```

(48) gramática cfg.lx.syn (Parte II)

```
A -> {ADJ/VB_AN}
NOM -> {N/NPR}
V -> {VB_FIN/VAUX}
INF -> {VB_PP/VP_G/VB}
DET -> D
```

QUANT -> {Q/NUM}
POSS -> PRO_S

A tabela 3 apresenta o desempenho de três *parsers* construídos combinando o *output* produzido pelo LX-Tagger na etiquetagem das sentenças (31) com a gramática cfg.lx.syn bem como com adaptações nos mesmos moldes, respectivamente, das gramáticas cfg02.syn e cfg03.syn.

TABELA 3
Desempenho do LX-Tagger como analisador lexical
na análise sintática automática

Gramática	**Percentual de sentenças analisadas**	**Análises por sentença**
cfg.lx.syn	94,44%	4,18
cfg02.lx.syn	94,44%	3,12
cfg03.lx.syn	97,22%	9,83

As FIG. 30, 31 e 32 apresentam análises atribuídas pelo primeiro desses *parsers* a três sentenças para as quais o *parser* da FIG. 16, baseado no *output* do etiquetador AeliusBRUBT, não gerou análise alguma devido a erros de etiquetagem. Devido às grafias abreviadas *q* e *p/*, analisadas como substantivo ou verbo, nem o LX-Tagger nem o VISL analisaram corretamente a sentença da FIG. 32.

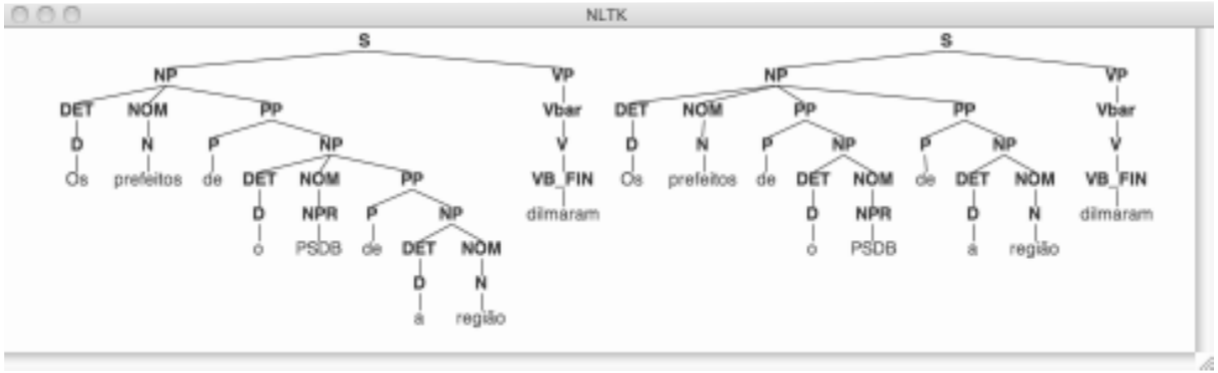

FIGURA 30: Análises do exemplo (31) (xii) por meio do Aelius com etiquetagem pelo LX-Tagger

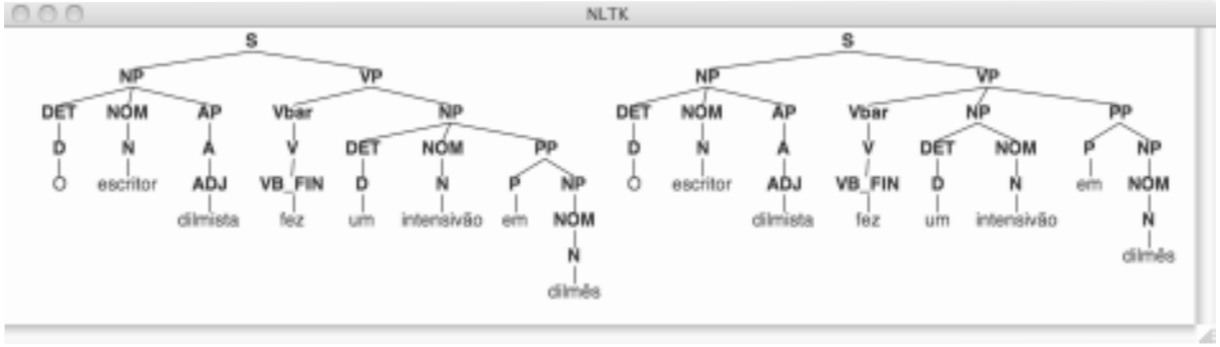

FIGURA 31: Análises do exemplo (31) (xxvii) por meio do Aelius com etiquetagem pelo LX-Tagger

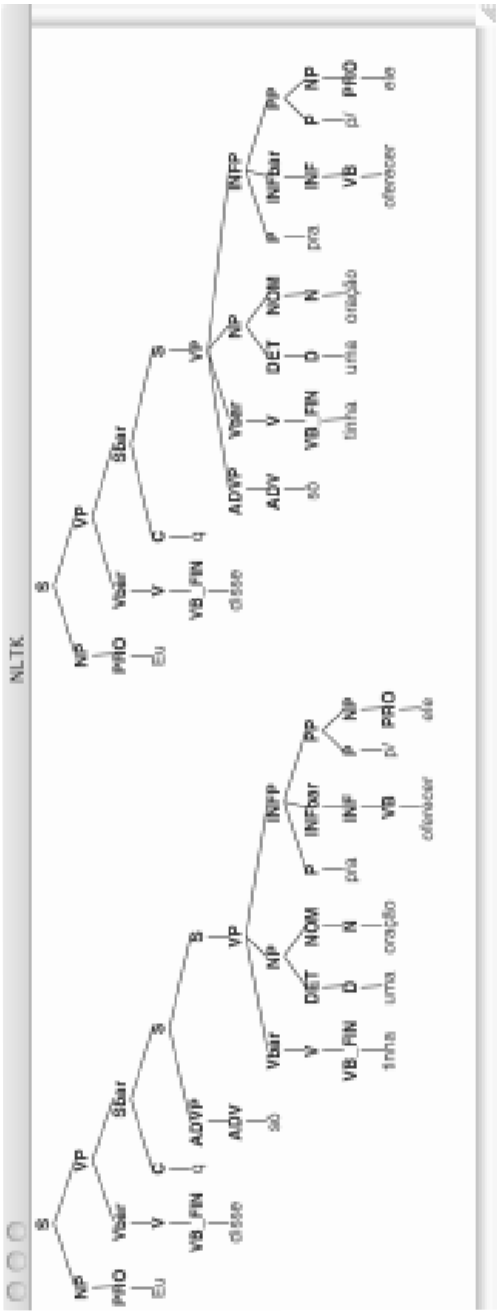

FIGURA 32: Análises de (31) (xxxvi) pelo *parser* construído com base na gramática cfg02.lx.syn e da etiquetagem do LX-Tagger

Na FIG. 33, mostramos como construir um *parser* a partir da gramática cfg03.lx.syn.nltk e da etiquetagem das sentenças de (31) pelo LX-Tagger, convertida na versão simplificada do conjunto de etiquetas do CHPTB. São apresentados os resultados da análise das 16 primeiras sentenças.

```
>>> import alexp
>>> from Aelius import carrega
>>> lx=carrega("lxtagger")
>>> alexp.configuraEtiquetador(lx,"mxpost")
>>> alexp.analisaBlocoDeSentencas("gramaticas/cfg03.lx.syn.nltk",anotado="sentencas.lxtagger2chptb.txt")
Nr. 1: 4 análise(s): Pesquisa melhora cultivo de células estaminais
Nr. 2: 10 análise(s): Micarla fez uma opção em a sucessão presidencial
Nr. 3: 2 análise(s): Ela se aproximou de Dilma
Nr. 4: 4 análise(s): Ela criou afinidades com o dilmismo
Nr. 5: 4 análise(s): Dilma ainda não tucanou
Nr. 6: 2 análise(s): O prefeito tucanou
Nr. 7: 2 análise(s): O nosso prefeito tucanou
Nr. 8: 3 análise(s): Muitos tucanos já dilmaram
Nr. 9: 8 análise(s): Muitos vereadores tucanos já dilmaram
Nr. 10: 3 análise(s): 16 prefeitos de o PSDB dilmaram
Nr. 11: 3 análise(s): 16 membros de o PSDB dilmaram
Nr. 12: 7 análise(s): Os prefeitos de o PSDB de a região dilmaram
Nr. 13: 7 análise(s): Os vereadores de o PSDB de a região dilmaram
Nr. 14: 2 análise(s): Os dilmistas vão votar em o PT
Nr. 15: 35 análise(s): Os bravos tucanos dilmaram por falta de liderança partidária
Nr. 16: 20 análise(s): Os bravos vereadores dilmaram por falta de liderança partidária
```

FIGURA 33: Análise das sentenças de (31) pelo parser construído a partir da gramática cfg03.lx.syn.nltk e da etiquetagem do LX-Tagger (Parte I).

A FIG. 34 traz o resultado do *parsing* das demais sentenças de (31). Como podemos constatar, apenas a sentença (31) (xxviii) não obteve análise alguma. Isso decorreu do fato de o LX-Tagger, treinado em um *corpus* de português europeu, não ter reconhecido a forma *nos* como pronome, mas como contração de preposição e artigo, uma vez que essa colocação pronominal, típica da variedade brasileira, é inusual na variedade europeia.

FIGURA 34: Análise das sentenças de (31) pelo *parser* construído com base na gramática cfg03.lx.syn.nltk e a etiquetagem do LX-Tagger (Parte II)

Para a sentença (6), à qual, como vimos na seção 2, tanto o VISL quanto o LX-Parser atribuíram análises estapafúrdias, o ALEXP gera, com base na gramática da FIG. 33e FIG. 34, uma análise bastante próxima da esperada:

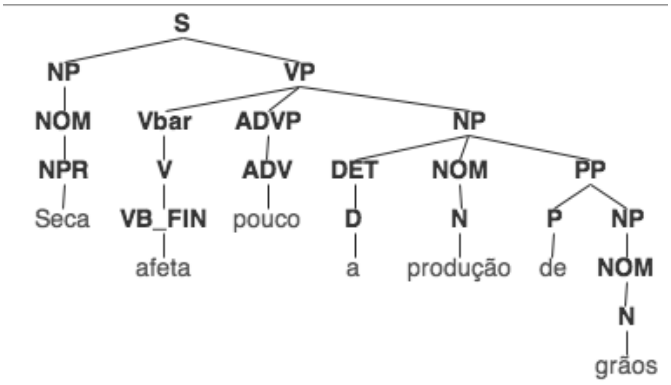

FIGURA 35: Análise da sentença (6) com base na gramática cfg03.lx.syn.nltk e a etiquetagem do LX-Tagger

6. Conclusão

Neste artigo, mostramos como utilizar, na análise sintática computacional de textos irrestritos, informações lexicais extraídas de *corpora* da língua portuguesa, de modo a poder prescindir tanto da elaboração de um analisador morfológico quanto da compilação de centenas de milhares de entradas lexicais. Para tanto, desenvolvemos o ALEXP, módulo em Python capaz de integrar o *output* de etiquetadores morfossintáticos de diversas arquiteturas em um sistema de *parsing* profundo. Com isso, visamos a contribuir para preencher lacuna no repertório de ferramentas para o processamento computacional do PB, que não conta com analisador sintático profundo de ampla cobertura distribuído sob licença livre. Por outro lado, os *parsers* que avaliamos revelaram-se imprecisos na análise não só de sentenças com neologismos e outros itens não dicionarizados, mas também de construções corriqueiras do PB.

Exemplificamos o funcionamento do ALEXP tanto com um etiquetador desenvolvido por meio do NLTK, no caso o AeliusBRUBT (ALENCAR, 2010), quanto com um etiquetador da arquitetura HunPos (o AeliusHunPos) e outro da arquitetura MXPOST, o LX-Tagger (BRANCO; SILVA, 2004). A partir do

output desses etiquetadores na etiquetagem morfossintática de um conjunto de sentenças dadas para análise, o ALEXP gera entradas lexicais sob a forma de regras livres de contexto. Essas sentenças são, então, analisadas por um *parser* produzido pelo NLTK com base nessas entradas lexicais e nas regras de estruturação sintagmática fornecidas pelo usuário. Um módulo que desenvolvemos, o c2n, simplifica enormemente a elaboração de regras livres de contexto capazes de serem processadas pelo NLTK.

Desse modo, o ALEXP desonera significativamente a construção de *parsers* para textos irrestritos, cujo maior gargalo é a análise lexical. Para alcançar isso, o ALEXP integra diferentes módulos que constituem *software* livre ou são livremente disponíveis, maximizando esforços, o que é de extrema importância sobretudo no âmbito do processamento computacional do português, ainda bastante carente de recursos.

Uma característica marcante dos textos do mundo real, para além da diversidade do vocabulário, é a ocorrência de grafias não padrão, abreviaturas, nomes próprios e neologismos não dicionarizados. No PLN, isso torna infrutífera a mera compilação de entradas lexicais, exigindo um tratamento computacional das formas desviantes bem como da formação de novas palavras.

Neste trabalho, na modelação do conhecimento lexical necessário para o *parsing* profundo no âmbito de gramáticas livres de contexto, prescindimos da elaboração de centenas de milhares de entradas lexicais. Como alternativa a esse custoso procedimento, também não recorremos à morfologia computacional, pois isso implicaria formular centenas de regras para obter um bom desempenho na análise lexical de textos irrestritos. Em vez disso, optamos por aproveitar os resultados das pesquisas em aprendizado de máquina, que permitem, hoje, construir etiquetadores eficientes com pouco ou nenhum esforço, a partir de *corpora* anotados morfossintaticamente.

O ALEXP, graças à utilização de ferramentas desse tipo livremente disponíveis, é capaz de lidar de forma bastante eficiente com as dificuldades lexicais típicas de textos irrestritos. Por exemplo, sentenças com palavras não dicionarizadas como *dilmaram* (forma de *dilmar* na acepção "tornar-se partidário de Dilma"), *dilmice, dilmismo*, etc. foram analisadas com sucesso. Um alto índice de acerto também foi obtido na análise de sentenças com formas não padrão como *q* "que", *p/* "para", *siscapar* "se escapar" e *dimenor* "de menor".

O analisador, contudo, fracassou nos casos em que os etiquetadores avaliados cometeram erros de etiquetagem. Na versão atual do ALEXP, apenas

alguns desses erros são corrigidos em um módulo de pós-etiquetamento. Um outro problema constatado na avaliação da ferramenta foi o número excessivo de árvores atribuídas a algumas sentenças, quando se amplia a cobertura da gramática, uma consequência da utilização do formalismo da CFG, que não permite uma modelação direta das estruturas de traços das categorias sintáticas.

No atual estágio de desenvolvimento, o ALEXP não passa de um protótipo, haja vista a relativamente pequena cobertura das gramáticas que avaliamos neste trabalho bem como a necessidade de melhorar a correção dos erros mais frequentes na etiquetagem dos textos. Graças à plugabilidade de Python e do NLTK, contudo, o programa não depende exclusivamente dos etiquetadores avaliados. A etiquetagem dos textos a serem analisados pode ser delegada a outros etiquetadores do NLTK ou que façam interface com o NLTK, como mostramos por meio da interface do NLTK que construímos para o MXPOST, graças à qual pudemos utilizar o LX-Tagger, que se revelou o mais preciso dos três etiquetadores avaliados. Em contrapartida, o ALEXP pode construir *parsers* também com base em regras de estruturação sintagmática elaboradas pelo usuário.

O aperfeiçoamento dos etiquetadores do Aelius, a construção de interfaces para outros etiquetadores mais robustos que venham a ser livremente disponibilizados, a implementação de regras de pós-etiquetamento visando a uma maior acurácia da análise morfossintática, a ampliação da gramática inicial e a construção de um modelo fundamentado na teoria X-barra e implementado no formalismo da gramática livre de contexto baseada em traços são as etapas seguintes de nosso projeto, que em breve será colocado à disposição da comunidade, sob licença livre aprovada tanto pela FSF quanto pela OSI, no repositório SourceForge.[54]

Notas

[1] Por meio de técnicas de aprendizado não supervisionado também é possível extrair informações linguísticas de *corpora* não anotados. No entanto, especialmente no âmbito do processamento computacional da sintaxe, as técnicas de aprendizado supervisionado, que recorrem a *corpora* anotados, permitem, no momento, alcançar melhores resultados (NIVRE, 2010, p. 257).

[2] A esse respeito, Di Felippo e Dias-da-Silva (2009, p. 18) afirmam: "O PLN no Brasil, isto e, o processamento computacional do portugues do Brasil (PB), em especial, ainda e escasso em tecnologias em relacao ao ingles".

[3] Python conquistou o lugar ocupado nesse quesito, até há alguns anos, por Perl. Sobre a aplicação de Python no processamento de textos, ver, por exemplo, Mertz (2003).

[4] A esse respeito, discordamos de Almeida *et al.* (2003, p. 102), que consideram a sintaxe como maior fonte de dificuldade para a construção de *parsers* robustos de textos reais.

[5] URL <http://code.google.com/p/hunpos/>.

[6] URL <http://lxcenter.di.fc.ul.pt/tools/en/LXTaggerEN.html>.

[7] A esse respeito, ver também Branco e Costa (2010).

[8] O *output* de etiquetadores morfossintáticos, originalmente concebidos para anotar *corpora*, tem sido usado como componente de diversos sistemas de PLN. Um exemplo de aplicação dessas ferramentas na tradução automática estatística baseada em sintagmas é Allauzen e Bonneau-Maynard (2008).

[9] O Aelius está disponível, sob licença livre, em <http://aelius.sourceforge.net/>.

[10] URL <http://lxcenter.di.fc.ul.pt/tools/pt/LXTaggerPT.html>.

[11] Ver, por exemplo, Jurafsky e Martin (2009, p. 484).

[12] Comumente, o termo *etiquetagem morfossintática* (ou *POS-tagging*) é empregado de forma lata, abrangendo a etiquetagem morfológica (ULE; HINRICHS, 2004, p. 219; LEMNITZER; ZINSMEISTER, 2006, p. 66-67; HAJIČOVÁ *et al.*, 2010, p. 168). De maneira inversa, porém menos freq*u*entemente, o termo *etiquetagem morfológica* é usado como hiperônimo do primeiro (NAMIUTI, 2004; VOUTILAINEN, 2004, p. 220; FELDMAN; HANA, 2010, p. 2). Para Xiao (2010, p. 157-158), a anotação morfológica tem como escopo unidades sublexicais como prefixos, sufixos e raízes, ao passo que a etiquetagem morfossintática incide sobre unidades lexicais.

[13] Árvore gerada com base na gramática (18) abaixo. Uma análise com base na teoria X-barra no âmbito do modelo de Princípios e Parâmetros, por exemplo, utilizando categorias como DP, TP, AgrsP, AgroP e CP seria ainda mais hierarquizada (GREWENDORF, 2002, p. 39).

[14] Disponível em: <http://www.apache.org/licenses/LICENSE-2.0>.

[15] URL http://sourceforge.net/.

[16] URL http://code.google.com/hosting/ .

[17] Disponível em: <http://nlp.lsi.upc.edu/freeling/>.

[18] Menuzzi e Othero (2008) mencionam também o GojolParser, remetendo à página na Linguateca onde o desenvolvedor do *parser* esclarece que se trata de ferramenta comercial, sobre a qual não há ainda publicação. Desse modo, desconsideramos esse analisador na nossa avaliação.

[19] O código-fonte do Curupira, na linguagem C++, não é passível de distribuição; apenas o código-objeto do *parser* é distribuído gratuitamente para fins acadêmicos (Ronaldo Martins, comunicação pessoal por *e-mail* em 9/4/2011).

[20] Uma busca pelo *parser* na Internet revelou-se infrutífera. Mensagens a respeito enviadas aos autores não foram respondidas.

[21] O *parser* parte do princípio que todo *input* é uma sentença bem-formada do português. A gramática do *parser* não codifica, por exemplo, a subcategorização.

[22] Almeida *et al*. (2003) afirmam que, em 28 sentenças do conjunto de teste, "the parser failed to terminate, or did not find the correct parsing [...]" (p. 108).

[23] Agradecemos a Maria das Graças Volpe Nunes por nos ter franqueado acesso ao *parser*.

[24] Extraído do exemplo (24).

[25] Disponível em <http://beta.visl.sdu.dk/visl/pt/parsing/automatic/trees.php>. No sítio do VISL, a análise *on-line* de sentenças é de livre acesso. A interação remota com essas ferramentas, contudo, é restrita a usuários aprovados pelo responsável pelo projeto e sujeita à cobrança de taxas.

[26] Esta e as demais abreviaturas que identificam as fontes de onde se extraíram os exemplos encontram-se expandidas no apêndice deste trabalho.

[27] A mesma sentença com o verbo no perfeito do indicativo foi analisada corretamente.

[28] Análise realizada *on-line* em 15/06/2011.

[29] URL <http://nlx.di.fc.ul.pt/>.

[30] Disponível em: <http://lxcenter.di.fc.ul.pt/services/en/LXServicesParser.html>.

[31] A inserção de uma vírgula após o NP adverbial não impediu o *parser* de incidir no mesmo erro de considerar esse sintagma como constituinte do NP sujeito.

[32] As árvores da FIG. 9 e da FIG. 8 foram geradas por meio do serviço *on-line* do LX-Parser em 06/04/2011; a da FIG. 10, em 11/05/2011. A árvore da FIG. 8 e a da FIG. 9 são idênticas às atribuídas pela versão *off-line* do *parser*, descarregada em 16/06/2011.

[33] Branco e Costa (2010, p. 86) afirmam que a LXGram é "distribuída sob licença de *software* de código aberto". A licença da gramática, porém, proíbe a sua redistribuição ou a distribuição de "produtos ou serviços" derivados (LXGRAM, [s.d.]), o que a torna incompatível com as noções de *software* livre e de *software* de código aberto da FSF e OSI, respectivamente.

[34] SS= sintagma substantivo, SP = sintagma preposicional.

[35] No Minimalismo, o princípio da maximalidade é relaxado, na medida em que as categorias D e ADV desses dois sintagmas são reinterpretadas como projeções máximas; o princípio da endocentricidade, porém, é preservado nesse modelo (FUKUI, 2003, p. 394-398).

[36] Disponível para *download* em: <http://sites.google.com/site/gabrielothero/home/publicacoes>.

[37] A geração de formas flexionadas por meio de regras formuladas manualmente é uma das áreas da morfologia computacional (BEESLEY; KARTTUNEN, 2003; ROARK; SPROAT, 2006, HIPPISLEY, 2010).

[38] Em consulta realizada em 20/06/2011, o Google não apresentou nenhum resultado para as palavras-chave *pitorocável* e *pitorocáveis*.

[39] Medição realizada em um MacBook Apple com processador 2.4 Ghz Intel Core 2 Duo, 2GB de RAM e sistema operacional Mac OS 10.6.7

[40] Mais detalhes sobre o AeliusHunPos constarão de um outro trabalho, em preparo.

[41] Ver, por exemplo, Bresnan (2001), Sag, Wasow e Bender (2003), Carnie (2002), Klenk (2003) e Othero (2009).

[42] Sobre a LFG e a HPSG, ver, por exemplo, Bresnan (2001), Falk (2001) e Müller (2007).

[43] Regras como A -> C d E são também livres de contexto. O NLTK, porém, não processa esse tipo de regra, limitando-se a um subconjunto da CFG que constitui uma variante da Forma Normal de Chomsky (CNF) (LJUNGLÖF; WIRÉN, 2010, p. 61-62).

[44] S = sentença, NP = sintagma nominal, VP = sintagma verbal, N = substantivo, V= verbo, Det = determinante.

[45] Numa gramática abrangente do português, é preciso incluir bem mais informações do que a categoria lexical de cada item. Na linguística computacional, essas informações são modeladas por meio de traços, no âmbito de formalismos como LFG, HPSG, etc.

[46] Em outro trabalho, trataremos do módulo c2n mais detalhadamente.

[47] NOM=substantivo, POSS= possessivo, PRO=pronome pessoal, DEM=demonstrativo, FP= partícula de foco, CL=clítico, SE=clítico reflexivo *se*, V=verbo finito, INF=verbo não finito, PA=contração de preposição e artigo, NEG=partícula de negação, QUANT= quantificador.

[48] Hifens e outros símbolos não alfabéticos utilizados nesse *corpus*, não aceitos pelo NLTK, são convertidos em "_". Desse modo, D_UM_F corresponde à etiqueta D-UM-F do CHPTB, que se aplica ao artigo indefinido feminino singular (CORPUS, 2010).

[49] Tivemos de lançar mão do símbolo NOM para designar os substantivos, em vez do símbolo tradicional N, porque este designa substantivos no singular no CHPTB, contrastando com N-P, substantivos no plural.

[50] Na mesma construção, o infinitivo *escapar* é analisado como verbo pelo VISL quando o reflexivo está ausente ou em próclise separado do verbo.

[51] A falta da concordância, prescrita pela norma culta, entre a partícula de foco e o pronome é frequente na linguagem coloquial da Internet.

[52] Essa expansão é feita também para a regra do INFP.

[53] Na LX-Suite, os traços de flexão são anotados por uma outra ferramenta.

[54] Disponível em: <http://sourceforge.net/>.

Referências

ALENCAR, L. F. de. Resenha de "Teoria X-barra: descrição do português e aplicação computacional", de Gabriel de Ávila Othero. *Revista Virtual de Estudos da Linguagem – ReVEL*, v. 6, n. 10, mar. 2008. Disponível em: <http://www.revel. inf.br/site2007 /_pdf/11/ resenhas/revel_10_resenha_othero.pdf>. Acesso em: 30 maio 2011.

ALENCAR, L. F. de. *Aelius*: uma ferramenta para anotação automática de *corpora* usando o NLTK. Trabalho apresentado ao IX Encontro de Linguística de *Corpus*, Porto Alegre, PUCRS, 8 e 9 de outubro de 2010. [S.l.]: [s.n.], 2010. Disponível em: <http://corpuslg.org/gelc/elc2010.php>. Acesso em: 22 set. 2011.

ALENCAR, L. F. de. *Aelius*: uma ferramenta para anotação automática de *corpora* usando o NLTK. Trabalho submetido para publicação nos Anais do IX Encontro de Linguística de *Corpus*, PUCRS, Porto Alegre, 8 e 9 de outubro de 2010. 2011.

ALLAUZEN, A.; BONNEAU-MAYNARD, H. Training and Evaluation of POS Taggers on the French MULTITAG Corpus. LANGUAGE RESOURCES AND EVALUATION CONFERENCE, n. 6, 2008, Marrakech, Morocco. *Proceedings*... [s.l.]: ELRA, 2008. Disponível em: <http://www.lrec-conf.org/proceedings/lrec2008/ pdf/856_paper.pdf>. Acesso em: 8 mar. 2011.

ALMEIDA, S. *et al*. Selva: A New Syntactic Parser for Portuguese. In: MAMEDE, N. *et al*. (Ed.). INTERNATIONAL WORKSHOP ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, n. 6, 2003, Faro. *Proceedings*... Berlin; Heidelberg: Springer, 2003. p. 102-109.

BEESLEY, K. R.; KARTTUNEN, L. *Finite state morphology*. Stanford, CSLI Publications, 2003. 510 p.

BICK, E. *The parsing system "Palavras"*: automatic grammatical analysis of Portuguese in a Constraint Grammar framework. 2000. Tese (Dr. phil.) – Department of Linguistics, University of Århus, Århus, Dinamarca, 2000. 505 p. Disponível em: <beta.visl.sdu.dk/pdf/PLP20-amilo. ps.pdf>. Acesso em: 27 out. 2009.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural language processing with Python*: analyzing text with the Natural Language Toolkit. Sebastopol: O'Reilly, 2009. 502 p.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Toolkit*. [s.l]: [s.n.], 2011. Disponível em: <http://www.nltk.org>. Acesso em: 24 jan. 2011.

BORBA, F. da S. *et al*. (Org.). 2. ed. *Dicionário gramatical de verbos do português contemporâneo do Brasil*. São Paulo: Fundação Editora da UNESP, 1991. 1373 p.

BRANCO, A.; SILVA, J. 2004. Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for Portuguese. In: LINO, M. T. *et al*. (Ed.). INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION, n. 4, 2004, Lisboa. *Proceedings*... Paris: ELRA, 2004. p. 507-510.

BRANCO, A.; COSTA, F. LXGram: A Deep Linguistic Processing Grammar for Portuguese. In: PARDO, T. A. S. *et al*. (Ed.). INTERNATIONAL CONFERENCE ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, n. 9, 2010, Porto Alegre. *Proceedings*... Berlin; Heidelberg: Springer, 2010. p. 86-89.

BRANCO, A. *et al*. Developing a Deep Linguistic Databank Supporting a Collection of Treebanks: the CINTIL DeepGramBank. LANGUAGE RESOURCES AND EVALUATION CONFERENCE, n. 7, 2010, La Valletta, Malta. *Proceedings*... [s.l.]: ELRA, 2010. p. 1810-1815. Disponível em: <http://www.lrec-conf.org/proceedings/lrec2010/pdf/154_Paper.pdf>. Acesso em: 26. abr. 2011.

BRESNAN, J. *Lexical-Functional Syntax*. Malden, Mass.; Oxford: Blackwell, 2001. 446 p.

CARNIE, A. *Syntax*: A generative introduction. Oxford: Blackwell, 2002. 390 p.

CARRERAS, X. *et al*. FreeLing: An Open-Source Suite of Language Analyzers. In: LINO, M. T. *et al*. (Ed.). INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION, n. 4, 2004, Lisboa. *Proceedings*... Paris: ELRA, 2004. Disponível em:<http://www.lsi.upc.edu/~nlp/papers/carreras04.pdf>. Acesso em: 27 maio 2011.

CHUN, W. J. *Core Python programming*. 2. ed. Upper Saddle River, NJ: Prentice Hall, 2006. 1078 p.

CONTIER, A.; PADOVANI, D.; JOSÉ NETO, J. Tecnologia Adaptativa Aplicada ao Processamento da Linguagem Natural. WORKSHOP DE TECNOLOGIA ADAPTATIVA, n. 4, 2010, São Paulo. *Memórias*... São Paulo: EPUSP, 2010. p. 35-42.

CORPUS Histórico do Português Tycho Brahe. Campinas: Instituto de Estudos da Linguagem/Universidade Estadual de Campinas, 2010. Disponível em: <http:// www. tycho. iel.unicamp.br/~tycho/corpus/>. Acesso em 30. set. 2010.

CURUPIRA: A Functional Parser for Brazilian Portuguese. São Carlos: Núcleo Interinstitucional de Linguística Computacional, [2004]. Disponível em: <http:// www.nilc. icmc. usp.br/nilc/tools/curupira.html>. Acesso em: 1. jun. 2011.

DI FELIPPO, A.; DIAS-DA-SILVA, B. C. O processamento automático de línguas naturais enquanto engenharia do conhecimento linguístico. *Calidoscópio*, São Leopoldo, v. 7, n. 3, p. 183-191, set./dez. 2009.

FALK, Y. N. *Lexical-functional grammar*: an introduction to parallel constraint-based syntax. Stanford, CSLI Publications, 2001. 237 p.

FELDMAN, A.; HANA, J. *A resource-light approach to morpho-syntactic tagging*. Amsterdam; New York: Rodopi, 2010. 185 p.

FREE SOFTWARE FOUNDATION. *The Free Software Definition*. [s.l.]: [s.n.], 2010. Disponível em:<http://www.gnu.org/philosophy/free-sw.html>. Acesso em: 16 de jun. 2011.

FREITAS, C.; ROCHA, P.; BICK, E. Um mundo novo na Floresta Sintá(c)tica: o treebank do Português. *Calidoscópio*, São Leopoldo, v. 6, n. 3, p. 142-148, set/dez 2008.

FUKUI, N. Phrase structure. In: BALTIN, M.; COLLINS, C. (Ed.). *The Handbook of Contemporary Syntactic Theory*. Malden, MA: Blackwell, 2003. p. 374-406.

GARCIA, M.; GAMALLO, P. Análise morfossintática para português europeu e galego: problemas, soluções e avaliação. *LinguaMÁTICA*, Braga, v. 2, n. 2, p. 59-67, jun. 2010.

GREWENDORF, G.; HAMM, F.; STERNEFELD, W. *Sprachliches Wissen*: eine Einführung in moderne Theorien der grammatischen Beschreibung. 3. ed. Frankfurt am Main: Suhrkamp, 1989. 467 p.

GREWENDORF, G. *Minimalistische Syntax*. Tübingen; Basel: A. Francke, 2002. 344 p.

GÜNGÖR, T. Part-of-Speech Tagging. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 205-235.

HAJIČOVÁ, E. *et al*. Treebank annotation. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 167-188.

HALÁCSY, P.; KORNAI, A. ; ORAVECZ, C. HunPos: an open source trigram tagger. ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL

LINGUISTICS, n. 45, 2007, Praga. *Proceedings...* Stroudsburg: Association for Computational Linguistics, 2007. p. 209-212.

HAUSSER, R. *Grundlagen der Computerlinguistik*. Berlin: Springer, 2000. 572 p.

HIPPISLEY, A. Lexical analysis. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 31-58.

HOUAISS; VILLAR, 2001. *IDICIONÁRIO Aulete*. Rio de Janeiro: Lexikon, 2011. Disponível em: <http://aulete.uol.com.br>. Acesso em: 23 jun. 2011.

JURAFSKY, D.; MARTIN, J.H. *Speech and language processing*: an introduction to natural language processing, computational linguistics, and speech recognition. 2. ed. London: Pearson International, 2009. 1024 p.

KLENK, U. *Generative Syntax*. Tübingen: Narr, 2003. 261 p.

LEMNITZER, L.; ZINSMEISTER, H. *Korpuslinguistik*: eine Einführung. Tübingen: Narr, 2006. 220 p.

LXGRAM. Lisboa: Universidade de Lisboa, Departamento de Informática, [s.d.]. Disponível em: <http://nlxgroup.di.fc.ul.pt/lxgram/>. Acesso em: 3 jun. 2011.

LEMNITZER, L.; WAGNER, A. Akquisition lexikalischen Wissens. In: H. LOBIN; L. LEMNITZER (Ed.). *Texttechnologie*: Perspektiven und Anwendungen. Tübingen, Stauffenburg, 2004. p. 245-266.

LJUNGLÖF, P. ; WIRÉN, M. Syntactic parsing. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 59-91.

MAIER, W. NeGra und TüBa-D/Z: ein Vergleich. In: REHM, G.; WITT, A.; LEMNITZER, L. (Ed.). Data structures for linguistic resources and applications. BIENNIAL GLDV CONFERENCE 2007. *Proceedings*. Tübingen: Gunter Narr, 2007. p. 29-38.

MARTINS, R.; HASEGAWA, R.; NUNES, G. *Curupira*: um parser funcional para a língua portuguesa. São Carlos: Núcleo Interinstitucional de Linguística Computacional, 2002. Disponível em: <http://www.nilc.icmc.usp.br/nilc/download/nilc-tr-02-26.zip>. Acesso em: 1 jun. 2011.

MARTINS, R.; NUNES, G.; HASEGAWA, R. Curupira: A Functional Parser for Brazilian Portuguese. In: MAMEDE, N. *et al.* (Ed.). INTERNATIONAL WORKSHOP ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, n. 6, 2003, Faro. *Proceedings...* Berlin; Heidelberg: Springer, 2003. p. 179-183.

MENUZZI, S. M.;OTHERO, G. A.; Sintaxe X-barra: uma aplicação computacional. *Working papers em Linguística*, Florianópolis, v. 9, p. 15-29, 2008.

MERTZ, D. *Text Processing in Python*. Upper Saddle River, NJ: Addison-Wesley, 2003. 520 p.

MÜLLER, S. *Head-Driven Phrase Structure Grammar*: eine Einführung. Tübingen: Stauffenburg, 2007. 440 p.

NAMIUTI, C. O *Corpus* Anotado do Português Histórico: um Avanço para as Pesquisas em Linguística Histórica do Português. *Revista Virtual de Estudos da Linguagem*, v.2, n. 3, 2004. Disponível em: <http://www.revel.inf.br/>. Acesso em: 1. abr. 2011.

NAUMANN, S. XML-basierte Tools zur Entwicklung und Pflege syntaktisch annotierter Korpora. In: MEHLER, A.; LOBIN, H. (Eds.). *Automatische Textanalyse*: Systeme und Methoden zur Annotation und Analyse natürlichsprachlicer Texte. Wiesbaden: VS Verlag für Sozialwissenschaften, 2004. p. 153-166.

NOVICHKOVA, S; EGOROV, S.; DARASELIA, N. MedScan: a natural language processing engine for MEDLINE abstracts. *Bioinformatics*, Oxford, v. 19, n. 13, p. 1699-1706, 2003.

NIVRE, J. Statistical parsing. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 237-266.

OPEN Source Initiative. [s.l.]: [s.n.], 2011. Disponível em:<http://www.opensource.org/>. Acesso em: 16 de jun. 2011.

OTHERO, G. A. *Teoria X-barra*: descrição do português e aplicação computacional. São Paulo: Contexto, 2006. 160 p.

OTHERO, G. A. *A gramática da frase em português*: algumas reflexões para a formalização da estrutura frasal em português. Porto Alegre: Edipucrs, 2009. 160 p. Disponível em:< http://www.pucrs.br/edipucrs/gramaticadafrase.pdf>. Acesso em: 2 ago. 2010.

PADRÓ, L. *et al*. FreeLing 2.1: Five Years of Open-Source Language Processing Tools. LANGUAGE RESOURCES AND EVALUATION CONFERENCE, n. 7, 2010, La Valletta, Malta. *Proceedings*... [s.l.]: ELRA, 2010. p. 931-936. Disponível em: <http://www.lrec-conf.org/proceedings/lrec2010/pdf/14_Paper.pdf> Acesso em: 1. fev. 2011.

PERKINS, J. *Python Text Processing with NLTK 2.0 Cookbook*. Birmingham, UK: Packt, 2010. 256 p.

RATNAPARKHI, A. A Maximum Entropy Model for Part-Of-Speech Tagging. EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, 1996, Philadelphia, Pennsylvannia. *Proceedings*... Pennsylvannia: University of Pennsylvannia, 1996. p. 133-142. Disponível em: <http://acl.ldc.upenn.edu/W/W96/W96-0213.pdf>. Acesso em:<2. Jun. 2011>

ROARK, B.; SPROAT, R. *Computational approaches to morphology and syntax*. Oxford: Oxford University Press, 2007. 316 p.

RON RONCAGLIA, D. Marina questiona se partidos farão "pacto de silêncio" sobre mensalões. *Folha online*. Disponível em: <http://noticias.bol.uol.com.br/brasil/2010/02/26/marina-questiona-se-partidos-farao-pacto-de-silencio-sobre-mensaloes.jhtm>. Acesso em: 28 mar. 2011.

SAG, I. A.; WASOW, T. ; BENDER, E. *Syntactic theory*: a formal introduction. 2. ed. Stanford: CSLI Publications, 2003. 608 p.

SILVA, J.; BRANCO, A.; GONÇALVES, P. Top-Performing Robust Constituency Parsing of Portuguese: freely available in as many ways as you can get it. LANGUAGE RESOURCES AND EVALUATION CONFERENCE, n. 7, 2010, La Valletta, Malta. *Proceedings*... [s.l.]: ELRA, 2010. p. 1960-1963. Disponível em: < http://www.lrec-conf.org /proceedings /lrec2010/pdf/136_Paper.pdf>. Acesso em: 26. abr. 2011.

SILVA, J. *et al*. Out-of-the-Box Robust Parsing of Portuguese. In: PARDO, T. A. S. et al. (Ed.). INTERNATIONAL CONFERENCE ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, n. 9, 2010, Porto Alegre. *Proceedings*... Berlin; Heidelberg: Springer, 2010. p. 75-85.

TAGNIN, S. E. O.; VALE, O. A. (Org.). *Avanços da Linguística de* Corpus *no Brasil*. São Paulo: Humanitas, 2008. 437 p.

ULE, T.; HINRICHS, E. Linguistische Annotation. In: LOBIN, H.; LEMNITZER, L. (Ed.). *Texttechnologie*: Perspektiven und Anwendungen. Tübingen: Stauffenburg, 2004. p. 217-243.

VOUTILAINEN, A. Part-of-speech tagging. In: MITKOV, R. (Ed.). *The Oxford handbook of computational linguistics*. Oxford: Oxford University Press, 2004. p. 219-232.

XIAO, R. Corpus creation. In: INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. 2. ed. Boca Raton, FL: Chapman & Hall/CRC, 2010. p. 147-165.

Apêndice: Textos de onde foram extraídos exemplos

Siglas Referências

ARA ARAÚJO, C. A. O revisor mental instantâneo de Dilma sumiu. *Coluna do Augusto Nunes*. 20/03/2011. Disponível em:<http://veja.abril.com.br/blog/augusto-nunes/direto-ao-ponto/o-revisor-mental-instantaneo-de-dilma-sumiu/>. Acesso em: 25 mar. 2011.

BAC BACON, R. Uma vez um craqueiro veio me assaltar. Disponível em: <http://twitter.com/#!/eloisazzz/statuses/29537672277>. Acesso em: 4 maio 2011.

DAN DANTAS, A. R. "Opção de Micarla nos distancia politicamente". *Tribuna do Norte*. 20/03/2011. Disponível em: <http://www.tribunadonorte.com.br/noticia/opcao-de-micarla- nos -distancia-politicamente /176054>. Acesso em: 25 mar. 2011.

DEP DEPUTADA Marilia denuncia uso de dinheiro público em festa particular da primeira-dama. *Folha de Boa Vista*, Boa Vista, 14/04/2010. Disponível em: <http:// www. folhabv. com.br/noticia.php?id=84248>. Acesso em: 28 mar. 2011.

DIL DILMA ainda não tucanou, mas muitos tucanos já dilmaram. *Observatório Urbano Belo Horizonte.* 2011. Disponível em: <http://observatoriourbanobh. blogspot.com /2011 /03/dilma-ainda-nao - tucanou-mas-muitos.html>. Acesso em: 25 mar. 2011.

GOM BÓRIS, R.; GOMES, T. *The Who is better than Beatles*. Disponível em: <http://www. Orkut. com/Community?cmm=70017&hl=pt-BR5/4/2011>. Acesso em: 5 abr. 2011.

PES PESQUISA melhora cultivo de células estaminais a partir de tecidos adultos. *Suplemento Especial Semana da Ciência e Tecnologia*. 2009. Disponível em: <http:// www.prof2000.pt/users/jdsa03/olho/0910/novembro/supciencia.htm#estaminais>. Acesso em: 11 abr. 2011.

RIB RIBEIRO, J. A. O cabeção Serra esqueceu a lei do filósofo Chico Heráclio. 2010. Disponível em: <http:// www. blogdomagno. com.br/componentes/paginas/imprimir_pagina_codigo.php?cod_ pagina=66174>. Acesso em: 24 maio 2011.

RON RONCAGLIA, D. Marina questiona se partidos farão "pacto de silêncio" sobre mensalões. *Folha Online*. Acesso em: 28 mar. 2011.

TIA TIAGO. Craqueiro esfaqueado é preso por furto. 2009. Disponível em: <http://blog.diarinho.com.br/craqueiro-esfaqueado-preso-por-furto>. Acesso em: 4 maio 2011.

UNI UFC convoca os classificáveis do Vestibular 2010. Disponível em: <http://noticias.universia.com.br/destaque/noticia/2010/02/17/411825/fc-convoca-os-classificaveis-do-vestibular-2010.html>. Acesso em: 17 maio 2011.

VIA VIANNA, A. J. Não existe lulismo nem dilmismo, diz presidente do PT. *Istoé Dinheiro*. 10/02/2011. Disponível em: <http://www. istoedinheiro .com. br/ noticias /49044_ NAO+ EXISTE+LULISMO+NEM+DILMISMO +DIZ+PRESIDENTE+DO+PT>. Acesso em: 25 mar. 2011.

Submissão do artigo: 23/6/2011
Aprovação do artigo: 10/9/2011