DONATUS: UMA INTERFACE AMIGÁVEL PARA O ESTUDO DA SINTAXE FORMAL UTILIZANDO A BIBLIOTECA EM PYTHON DO NLTK

Leonel Figueiredo de ALENCAR*

- RESUMO: Este trabalho objetiva, primeiramente, evidenciar a utilidade da CFG e da FCFG no estudo da sintaxe formal. A aplicação de parsers baseados nesses formalismos na análise de um corpus pode revelar consequências de uma dada análise que de outro modo passariam despercebidas. O NLTK é uma caixa de ferramentas para o PLN em Python que possibilita a construção de parsers em diferentes arquiteturas. No entanto, para uma utilização não trivial dessa biblioteca na análise sintática automática são necessários conhecimentos de programação. Para permitir o acesso de não programadores à implementação e testagem de parsers, desenvolvemos o Donatus, uma interface gráfica amigável para as facilidades de *parsing* do NLTK, dotada de recursos adicionais que a tornam interessante também para programadores. Como exemplo do funcionamento da ferramenta e demonstração da sua relevância na investigação sintática formal, comparamos implementações de duas análises alternativas da modificação adjetival em português. A primeira abordagem, baseada na Teoria X-barra tradicional, produziu um grande número de pseudoambiguidades. Esse problema foi evitado por um parser baseado em abordagem no âmbito do Programa Minimalista. Sem o recurso do computador, essa diferença entre as duas abordagens não seria facilmente revelada.

- PALAVRAS-CHAVE: Linguística computacional. Sintaxe formal. Gramática gerativa. Teoria X-barra. Gramática livre de contexto. Gramática de unificação. Modificação adjetival.

Introdução[1]

A partir de 1980, durante quase duas décadas, a linguagem de programação Prolog reinou quase hegemonicamente nos cursos introdutórios sobre o processamento automático da linguagem natural (doravante PLN). Isso se deveu, sobretudo, à relativa simplicidade do formalismo DCG, que permite uma implementação quase direta de analisadores e geradores de sentenças a partir de uma descrição dos sintagmas de uma língua por meio de uma gramática livre de contexto (doravante CFG, do inglês *context-free grammar*)[2]. Graças à natureza declarativa de Prolog, uma mesma CFG, implementada no formalismo

* UFC – Universidade Federal do Ceará. Fortaleza – CE – Brasil. 60020-181 – leonel.de.alencar@ufc.br

[1] Agradecemos as valiosas sugestões e os instigantes questionamentos de um dos pareceristas.

[2] Othero (2006) constitui um exemplo em português da utilização da DCG num contexto didático.

DCG, serve tanto para a análise quanto para a geração, sem que o linguista precise se preocupar com a elaboração dos respectivos algoritmos (BLACKBURN; BOS; STRIEGNITZ, 2006). Por outro lado, o formalismo DCG permite modelar de forma econômica, por meio da associação de traços morfossintáticos às regras e aos itens lexicais, fenômenos de difícil implementação na CFG, como, por exemplo, a concordância e a subcategorização.

A principal e bastante conhecida limitação do formalismo DCG é que Prolog utiliza, no processamento dessas gramáticas, uma estratégia de *parsing* recursivo-descendente, na qual regras recursivas à esquerda podem levar o parser a um *laço* infinito. Para resolver esse tipo de problema na implementação de uma dada CFG, é preciso recorrer a soluções arbitrárias, linguisticamente contraintuitivas (OTHERO, 2006; BLACKBURN; BOS; STRIEGNITZ, 2006). Desse modo, a utilização da DCG na modelação da Teoria X-barra a partir da CFG torna-se inviável num contexto didático, uma vez que o fenômeno da adjunção à direita nas línguas naturais envolve esse tipo de regra, conforme podemos constatar no exemplo (1), extraído de Radford (1988, p.274)[3].

(1) X' -> X' YP

Analogamente ao que propõe Klenk (2003, p.84) para o espanhol, Othero (2006, p.41) e Othero (2009, p.55) postulam várias instanciações de (1) para o português, como, por exemplo, a regra (2), exemplificada em (3)[4].

(2) N' -> N' AP

(3) uma comida italiana gostosa

Além dessa deficiência, é um fato bastante conhecido que algoritmos de *parsing* recursivo-descendentes são extremamente ineficientes, pois não armazenam análises parciais nem utilizam o input para orientar a aplicação das regras da gramática (JURAFSKY; MARTIN, 2009, p.466).

Visando superar essas desvantagens da DCG, diversos softwares de desenvolvimento de gramáticas implementando algoritmos de *parsing* mais sofisticados foram elaborados e livremente disponibilizados nas duas últimas décadas. Desse modo, a utilização da DCG no ensino da sintaxe formal, abstraindo de razões históricas, perdeu o atrativo de que antes gozava.

---

[3] É possível evitar esse problema construindo um parser ascendente para interpretação da DCG, em substituição ao parser descendente *default* de Prolog (BLACKBURN; BOS; STRIEGNITZ, 2006, p.155). Com isso, porém, a atratividade inicial da DCG desaparece.

[4] Para Mioto, Silva e Lopes (2005, p.95), APs, PPs não complementos e orações relativas constituem adjuntos da projeção máxima NP e não da projeção intermediária N'.

Atualmente, o pacote mais abrangente e amigável de PLN é o Natural Language Toolkit (doravante NLTK), uma vasta biblioteca implementada na linguagem de programação Python, constituindo um conjunto de ferramentas para a análise automática de textos nos mais diferentes níveis (BIRD; KLEIN; LOPER, 2009, 2012). No âmbito da sintaxe, o NLTK implementa os algoritmos de *parsing* mais conhecidos, tanto para a CFG clássica quanto para extensões desse formalismo. Por meio de interfaces acessíveis a linguistas sem conhecimentos de programação, é possível construir analisadores sintáticos elementares em diversas arquiteturas, como, por exemplo, o *parsing* utilizando "tabelas" (*charts*), que não sofre da limitação, exemplificada em (2), do algoritmo do Prolog para processar a DCG, sendo, ao mesmo tempo, muito mais eficiente. Outras ferramentas relevantes para o processamento da sintaxe, como etiquetadores, *chunkers* etc. são igualmente disponibilizadas.

O presente artigo tem como objetivo principal subsidiar a utilização do NLTK no ensino e aprendizagem da sintaxe formal em cursos de graduação e pós-graduação na área de Letras e Linguística. Apesar da preocupação didática dos autores dessa biblioteca, conhecimentos de nível intermediário a avançado em Python são indispensáveis para uma utilização mais sofisticada do NLTK na implementação computacional e testagem de gramáticas de maior complexidade.

Como uma ponte sobre o fosso que geralmente impede um envolvimento maior de estudantes típicos de Letras e Linguística com a sintaxe formal e seu tratamento no computador, desenvolvemos o Donatus, que apresentamos neste artigo. Esse programa oferece uma interface gráfica amigável que permite utilizar o NLTK, de forma não trivial, no processamento automático da sintaxe sem precisar digitar comando algum em Python, oferecendo, igualmente, recursos não disponíveis nessa biblioteca.

Além desta introdução e da conclusão, o restante do trabalho está estruturado em cinco seções. Na próxima seção, historiamos, em largas pinceladas, o papel da CFG no gerativismo, apresentado-a como um dos fundamentos da iniciação ao estudo da sintaxe formal, independentemente da perspectiva teórica. Em seguida, tratamos de extensões da CFG baseadas na unificação, concentrando-nos naquele que é o formalismo gramatical do NLTK mais poderoso e mais interessante sob a perspectiva gerativa: a gramática livre de contexto baseada em estruturas de traços (doravante FCFG, do inglês *feature-based context-free grammar*). A FCFG permite superar as várias limitações da CFG sem comprometer o *parsing*, comprometimento esse implicado por formalismos mais poderosos como, por exemplo, a gramática transformacional. Antes de explicar o funcionamento do Donatus na quarta seção, damos uma amostra, na terceira, de como construir um parser para uma gramática e de como aplicá-lo na análise de expressões linguísticas, utilizando o NLTK a partir da *shell* de Python. Com isso, evidencia-

se a motivação para a construção do Donatus, cuja utilização é vantajosa não só para usuários sem conhecimentos de programação em Python, mas também para aqueles familiarizados com essa linguagem.

De modo a tornar mais palpável a relevância, para o estudo da sintaxe formal, da análise sintática automática de um modo geral e da utilização do Donatus em particular, focamos, na terceira e quinta seções, um problema sintático específico: a modificação adjetival em português. Mostramos como construir analisadores em diferentes arquiteturas de *parsing* para duas abordagens dessa questão. A primeira abordagem implementa, na CFG, a análise tradicional do fenômeno no quadro da teoria X-barra. Uma desvantagem dessa análise é rapidamente evidenciada pela aplicação do parser, por meio do Donatus, a exemplos triviais, o que, sem o recurso do computador, demandaria muita paciência e elevadas habilidades matemáticas. A segunda abordagem contorna essa desvantagem por meio de uma implementação não transformacional, no formalismo FCFG, de uma abordagem baseada no Programa Minimalista, que testamos com sucesso no Donatus.

A CFG no estudo da sintaxe formal

A noção de gramática como dispositivo constituído de regras operando sobre um inventário de símbolos para formar cadeias (*strings*) de uma língua formal tem sua origem na lógica matemática com os trabalhos de Axel Thue e Emil Post, no início do século XX (MATEESCU; SALOMAA, 1997). Na década de 50, a teoria das línguas formais recebeu impulsos significativos dos primeiros trabalhos de Chomsky, passando a ter aplicação não só na linguística, especificamente na descrição da sintaxe das línguas naturais, mas também na ciência da computação, notadamente na construção de compiladores.

Chomsky (1957) examina a adequação de vários desses dispositivos (gramáticas gerativas na acepção matemática do termo) para descrever as línguas naturais. Um desses modelos é a CFG, por meio da qual podemos definir um número infinito de sentenças de uma língua como o português por meio de regras no formato geral X -> Y, como as de (1) e (2).

Nesse trabalho, Chomsky conclui que a CFG, sozinha, não é adequada como teoria (modelo) de uma língua natural, necessitando ser complementada por regras transformacionais. Na versão de seu modelo conhecida como Teoria Padrão, lançada em 1965, ele propõe que a derivação de uma sentença começa num componente de base, onde uma CFG gera árvores cujos nós terminais (representando categorias como V ou N) são preenchidos por entradas lexicais em consonância com regras de subcategorização, formuladas em termos de uma gramática sensível ao contexto (*context-sensitive grammar*, doravante CSG).

Essas árvores, por sua vez, constituem o input de operações de transformação, relacionando, por exemplo, uma sentença na voz ativa com a sua forma passiva (CHOMSKY, 1973)[5].

Para exemplificar a distinção entre CFG e CSG, sejam as regras (4) – (6). Em português, na derivação de um sintagma verbal, passível de expansão pela regra livre de contexto (4), a categoria V pode ser expandida por uma das regras sensíveis ao contexto (5) ou (6) (CHOMSKY, 1973, p.122):

(4) VP -> V (NP)

(5) V -> [V,+Transitivo]/ ___ NP

(6) V -> [V,–Transitivo]/ ___ #

Conforme (5), o símbolo V expande-se no complexo de traços [V,+Transitivo] se e somente se esse nódulo preceder um NP, ao passo que (6) assegura a substituição de V por [V,–Transitivo] apenas quando esse nódulo constituir o último elemento da derivação. Com isso, estruturas agramaticais do tipo de (7) e (8) são excluídas, pois os verbos *matar* e *morrer* são caracterizados, no léxico, como [+V, +__NP] e [+V, +__#], respectivamente (CHOMSKY, 1973, p.126).

*(7) *O gato matou.*[6]

*(8) *O gato morreu os ratos.*

No modelo Princípios e Parâmetros (doravante P&P), os diferentes tipos de transformações cederam lugar a um único tipo: o movimento alfa. Uma das motivações para essas operações de movimento é dar conta da interpretação de constituintes que aparecem fora do lugar onde são interpretados, como a expressão nominal *nós alunos*, alvo do quantificador *todos* em (9), como evidencia (10).

(9) nós alunos tínhamos [$_{VP}$ todos elogiado a dentista]

(10) todos nós alunos tínhamos [$_{VP}$ elogiado a dentista]

Como em (9) a expressão nominal não está adjacente ao quantificador, mas o precede de forma não imediata, postula-se que sofreu deslocamento da sua posição originária, interna ao VP (GREWENDORF, 2002, p.48-49). Em (10), tanto o quantificador como a expressão nominal teriam sido deslocados para o início da sentença. Outra motivação importante para as transformações nos diferentes

[5] Neste trabalho, por questão de espaço, temos de nos limitar a uma versão um tanto simplificada do modelo.

[6] Pressupomos que não há, nessa sentença, uma categoria vazia representando um objeto direto elíptico, que obviaria a agramaticalidade da estrutura.

modelos gerativo-transformacionais é o tratamento da concordância (SAG; WASOW; BENDER, 2003, p.41).

Com o advento da Teoria X-barra, a CFG perdeu, na década de 1970, a importância de que antes gozava na gramática gerativo-transformacional. De fato, a Teoria X-barra reduziu o vasto aparato de regras de estrutura sintagmática dos modelos anteriores a uns poucos esquemas de regras de aplicação universal, como as regras do especificador e do complemento, respectivamente em (11) e (12) (RADFORD, 1988, p.277). Esses esquemas eram vistos como restrições sobre o formato das regras de estrutura sintagmática das línguas particulares. (FUKUI, 2003, p.536-561).

(11) X'' -> X', (YP)

*(12) X' -> X, YP**

No modelo P&P, as regras de estrutura sintagmática são consideradas redundantes, uma vez que duplicam a informação de subcategorização contida no léxico, sendo, portanto, eliminadas da teoria (FUKUI, 2003, p.536-561).

No Programa Minimalista (doravante PM), toda a combinatória de itens lexicais para formar estruturas maiores, até constituir uma sentença, é reduzida a uma única operação: Compor (*Merge*). (CHOMSKY, 1995). Nesse modelo, a própria noção de estrutura sintagmática é posta em questão (CHAMETZKY, 2003).

Até o final da década de 1970, a CFG constituiu componente obrigatório dos manuais de ensino de sintaxe formal, dada a influência do paradigma chomskyano nos estudos da linguagem natural a partir de uma perspectiva lógico-matemática. (ver, por exemplo, Bach, 1974).

Na década seguinte, a CFG continuou ainda a ser bastante explorada em livros introdutórios do modelo P&P, como, por exemplo, Radford (1988), apesar de o autor também considerar dispensáveis as regras de estrutura sintagmática. Esse livro-texto constitui um exemplo paradigmático da importância didática da formalização da sintaxe por meio de uma CFG, uma vez que utiliza regras livres de contexto para descrever o Componente Categorial de um amplo fragmento do inglês, estando presentes em 7 dos 10 capítulos. Os esquemas da Teoria X-barra são introduzidos apenas no final da primeira metade do livro. Antes disso, as principais estruturas sintagmáticas do inglês são descritas por meio da CFG.

Na metodologia proposta por Radford (1988), os alunos procedem indutivamente: primeiro aprendem a descrever os padrões sintagmáticos básicos do inglês por meio de uma CFG para depois realizar generalizações sobre essas regras, conforme o espírito da Teoria X-barra. Ao procederem dessa

forma, refazem o percurso histórico do modelo gerativo-transformacional, que se tem caracterizado, principalmente, pela busca de teorias que se pretendem cada vez mais econômicas. Para o aluno, fica mais palpável avaliar o estágio atual da teoria, no que diz respeito à consecução desse objetivo, se o pode comparar com os estágios de desenvolvimento precedentes. As habilidades cognitivas adquiridas ou exercitadas nessa comparação serão, igualmente, importantes quando se defrontar com a necessidade de julgar análises divergentes de um mesmo fenômeno linguístico ou desdobramentos rivais da teoria atual.

Essa mesma estratégia didática orienta as introduções à sintaxe gerativa de Raposo (1992), subjazendo também a Carnie (2002), apesar de publicado em plena vigência do Minimalismo, tematizado por um dos capítulos. Mioto, Silva e Lopes (2005), pelo contrário, em seu manual de introdução ao modelo P&P, adotam estratégia oposta, limitando-se a explicarem a Teoria X-barra por meio de árvores, ignorando por completo a CFG, mas sem oferecerem um mecanismo alternativo para a geração das sentenças, como a operação Compor do PM.

Para o estudo do processamento computacional da sintaxe, a familiaridade com a CFG é um pré-requisito indispensável. Primeiro, ocupa lugar central na teoria das línguas formais, base para o estudo dos algoritmos de *parsing* (WINTNER, 2010; LJUNGLÖF; WIRÉN, 2010). Segundo, a CFG constitui um dos fundamentos dos modelos gramaticais baseados na unificação (como a LFG e a HPSG), de que trataremos na seção seguinte. Desse modo, os respectivos manuais introdutórios (FALK, 2001; SAG; WASOW; BENDER, 2003) ou os compêndios de sintaxe formal ou gerativa não restritos à vertente transformacionalista. (CARNIE, 2002; KLENK, 2003) não poderiam deixar de lhe dedicar o espaço que antes constatávamos nos manuais de gramática gerativo-transformacional na década de 1970 e 1980. Finalmente, a CFG constitui um dos fundamentos do *parsing* probabilístico e da compilação de *treebanks* (JURAFSKY; MARTIN, 2009; NIVRE, 2010) bem como de várias arquiteturas da tradução automática estatística ou baseada em exemplos (WAY, 2010).

Diante de tudo isso, não surpreende a ênfase do NLTK nesse formalismo. Por meio dessa biblioteca, podemos construir diversos tipos de parsers tanto para a CFG quanto para a FCFG, objeto da próxima seção. Esses parsers possibilitam, a seu turno, verificar automaticamente a validade de uma determinada análise de um fenômeno sintático particular, verificação essa dificilmente realizável manualmente, transformando o computador em valioso aliado no estudo da sintaxe formal.

Extensões da CFG baseadas na unificação

Como vimos, Chomsky, na Teoria Padrão, procurou contornar as limitações da CFG, utilizada para descrever as estruturas arbóreas do componente de base, pela inclusão, no modelo, de regras de subcategorização no formato da CSG, de modo a dar conta de restrições de coocorrência entre os itens lexicais. Outros fenômenos, principalmente relações sistemáticas de natureza semântica e estrutural entre pares de construções, levaram-no a propor transformações, operações que manipulam, de diversas formas, as árvores geradas pelo componente de base. No modelo P&P, passou-se a admitir apenas transformações de movimento.

Contrariamente à opinião comum, a que dão voz, por exemplo, Rodrigues e Augusto (2009), o paradigma gerativo não se limita à teoria transformacional chomskyana. A partir de meados da década de 1970, face às limitações da CFG, vários sintaticistas gerativos seguiram um caminho diferente ao adotado por Chomsky. Em vez de recorrer a regras sensíveis ao contexto ou a transformações, aperfeiçoaram a CFG por meio do enriquecimento das regras dessa gramática com estruturas de traços, estruturas essas sujeitas à operação matemática de unificação (KLENK, 2003; LJUNGLÖF; WIRÉN, 2010)[7].

Razões de ordem tanto computacional quanto psicolinguística desempenharam um papel importante nesse desdobramento. Conforme Klenk (2003, p.79-80), gramáticas transformacionais equivalem a gramáticas do tipo RE (abreviatura de recursivamente enumeráveis), que gera a família RE de línguas, o tipo mais geral na Hierarquia de Chomsky, representada em (13) (WINTNER, 2010, p.39), onde -> expressa a relação de inclusão.

(13) REG (ou L_3) -> CFG (ou L_2) -> CSG (ou L_1) -> RE (ou L_0)

Uma das conclusões mais importantes da teoria das línguas formais, disciplina matemática que constitui um dos fundamentos da ciência da computação, é que o chamado *problema do elemento*, que consiste em determinar se uma dada cadeia pertence a uma dada língua, é "indecidível" para as gramáticas do tipo RE. Desse modo, dadas uma gramática arbitrária G_1 pertencente a RE e uma cadeia arbitrária *x*, não é possível determinar algoritmicamente se *x* pertence ou não a G_1 (KLENK, 2003, p.80; MATEESCU; SALOMAA, 1997, p.31). Em consonância com isso, Ljunglöf e Wirén (2010, p.83) ressaltam que a unidirecionalidade do mapeamento de estruturas profundas sobre as cadeias de superfície na gramática

[7] Entre as teorias baseadas na unificação de traços, destacam-se a HPSG e a LFG, consideradas modelos gerativos (CARNIE, 2002, p.335). A última, para Falk (2001), constitui uma variedade de gramática gerativa que se fundamenta na incompatibilidade entre pressupostos transformacionais e a formulação de uma teoria da Gramática Universal, sendo, pela rigorosa formalização, mais fiel ao ideário geratividade que o P&P e o PM. A HPSG compartilha dessa característica da LFG (SAG; WASOW; BENDER, 2003).

transformacional tornam o modelo dificilmente suscetível ao *parsing*, que implica procedimento inverso. Pratt-Hartmann (2010, p.63) destaca a dificuldade de obtenção de resultados definitivos sobre a complexidade computacional de modelos transformacionais como o P&P e o PM que não são definidos com o rigor formal de modelos como a CFG e CSG (FALK, 2001, p.65). Embora admita a decidibilidade do problema do elemento para gramáticas transformacionais de menor complexidade, Pratt-Hartmann considera indecidível a classe das gramáticas transformacionais como um todo. Para as demais famílias de gramáticas, incluindo a CSG e a CFG (que geram respectivamente L_1 e L_2), ao contrário, o problema do elemento é sempre decidível, viabilizando o *parsing* computacionalmente.

No entanto, o problema da CSG é a complexidade do algoritmo necessário para determinar se uma cadeia pertence ou não a uma CSG particular, o que se reflete em diferenças dramáticas no tempo de processamento, comparando com a CFG. Na CSG, o tempo para processar uma cadeia de comprimento n é calculado exponencialmente por meio de k^n, onde k é um valor constante que independe do comprimento do input, ao passo que, na CFG, o tempo de processamento é calculado polinomialmente por meio de n^k. As consequências em termos de tempo de processamento resultantes desses dois tipos de complexidade são apresentadas na Tabela 1, admitindo-se que cada operação de processamento consome 10^{-6} segundo e k=2.

Tabela 1 – Comparação entre o tempo de processamento na complexidade polinomial e na complexidade exponencial

n	n^2	2^n
10	0.0001 segundo	0.0010 segundo
20	0.0004 segundo	1.05 segundo
30	0.0009 segundo	17.92 minutos
40	0.0016 segundo	12.74 dias
50	0.0025 segundo	35.75 anos

Fonte: Klabunde (2004, p.88).

A Tabela 1 deixa claro por que a CSG é impraticável para aplicações computacionais, pois, pela complexidade exponencial, uma cadeia de extensão 50 levaria mais de 35 anos para ser processada. A CFG, pelo contrário, é de complexidade polinomial n^3 (KLABUNDE, 2004, p.89), pelo que uma cadeia de igual extensão é processada em 0.125 segundo.

Nas gramáticas de unificação, entre as quais se destacam, atualmente, a LFG e a HPSG, formalismos que "recheiam" um esqueleto no formato da CFG com estruturas de traços, as questões tanto semânticas quanto morfossintáticas que motivam uma análise baseada em movimento são tratadas de uma forma computacionalmente bastante eficiente sem recorrer a transformações. Por exemplo, a relação entre o quantificador *todos* e o DP *nós alunos* em exemplos como (9), assim como outros casos de dependência de longa distância modelados como resultado de movimento em abordagens transformacionais, pode ser modelada de forma elegante na gramática de unificação por meio das chamadas categorias-barra (*slash-categories*), de que trataremos mais adiante.

Por conta disso, a LFG e a HPSG, enquanto constituem apenas nichos de resistência, na linguística teórica gerativa, ao pensamento transformacionalista predominante, na linguística computacional, junto com outros modelos baseados na unificação de traços, passam a preponderar (KAPLAN, 2004, p.84; LJUNGLÖF; WIRÉN, 2010, p.83-84).

Para o tratamento automático da subcategorização, não precisamos recorrer, necessariamente, à CSG, uma vez que esse fenômeno pode ser implementado computacionalmente numa CFG, como no exemplo (14).

(14) VP -> Vt NP | Vi

Vt -> "matou"

Vi -> "morreu"

Uma solução nos mesmos moldes pode dar conta da concordância no âmbito de uma CFG, criando subcategorias com base nos diferentes traços flexionais, como Dm e Df (determinante masculino e feminino, respectivamente) no exemplo (15).

(15) DP -> Dm NPm | Df NPf

NPm -> Nm

NPf -> Nf

Nf -> "bailarina" | "aluna"

Nm -> "palhaço" | "aluno"

Dm -> "o" | "este"

Df -> "a" | "esta"

No entanto, devido à proliferação de subcategorias, a estratégia exemplificada em (14) e (15) é muito custosa computacionalmente, em termos

de elaboração e manutenção de gramáticas com uma cobertura razoável, e o resultado, deselegante, impedindo expressar generalizações linguísticas (SAG; WASOW; BENDER, 2003, p.40; KAPLAN, 2004, p.81). Por exemplo, para modelar a flexão de número, precisaríamos duplicar a primeira regra da minigramática (15):

(16) DP -> Dms NPms | Dfs NPfs | Dmp NPmp | Dfp NPfp

Uma solução computacionalmente eficiente e linguisticamente elegante para o tratamento computacional da subcategorização e da concordância é aumentar as regras da CFG com estruturas de traços, que são pares da forma *atributo = valor*, onde o valor pode ser atômico ou constituído por uma outra estrutura de traços. Estruturas de traços são representadas, comumente, por meio de matrizes de atributos e valores (doravante AVM, do inglês *attribute-value matrix*), como em (17a) e (17b), que representam o gênero e o número dos vocábulos *menina* e *alunos*, respectivamente.

(17) a. [num='sg', gend='f']

b. [num='pl', gend='m']

Uma das utilidades principais da representação das propriedades de objetos linguísticos sob a forma de AVMs decorre da possibilidade que temos de especificar fenômenos como a concordância e a subcategorização, entre outros, em termos da unificação de traços. A unificação é uma operação matemática que permite combinar duas ou mais AVMs para formar uma única AVM, desde que os valores dos atributos de cada uma das AVMs individuais não sejam contraditórios entre si, como no caso de **estas menino*, em que os valores de gênero e número do demonstrativo e do substantivo são incompatíveis.

A LFG e a HPSG são formalismos gramaticais matematicamente muito complexos. Na primeira teoria, as AVMs são projetadas a partir de regras com metavariáveis e toda a gama de expressões regulares (FALK, 2001). Na última teoria, utilizam-se estruturas de traços "tipadas" (*typed*). (SAG; WASOW; BENDER, 2003). O NLTK permite familiarizar-se com os princípios básicos da gramática de unificação por meio da FCFG, um modelo relativamente simples, de estruturas de traços não tipadas, o qual não dispõe de metavariáveis e cujo único operador de expressões regulares é a disjunção lógica "|". Na FCFG, a CFG de (15), por exemplo, pode ser reformulada com mais elegância nos moldes de (18), que, ao contrário da primeira gramática, modela também os traços de número e pessoa do DP sem recorrer a uma proliferação de regras. Nessa nova gramática, a cada categoria se associa a AVM [agr=?a,bar=*n*], onde o atributo *agr* (de *agreement*, concordância em inglês) tem como valor uma variável arbitrariamente designada

por ?a e o atributo *bar* tem como valor um número natural, especificando o nível de projeção. O valor da AVM [agr=?a] do DP deve resultar da unificação das AVMs análogas do D e do NP. Caso a unificação seja possível, a concatenação das duas categorias é gramatical. Pelo contrário, caso a unificação fracasse, a concatenação de D e NP é agramatical.

(18) *D[agr=?a,bar=1] -> D[agr=?a,bar=0] N[agr=?a,bar=1]*

N[agr=?a,bar=1] -> N[agr=?a,bar=0]

N[agr=[num='sg', gend='m'],bar=0] -> 'palhaço' | 'aluno'

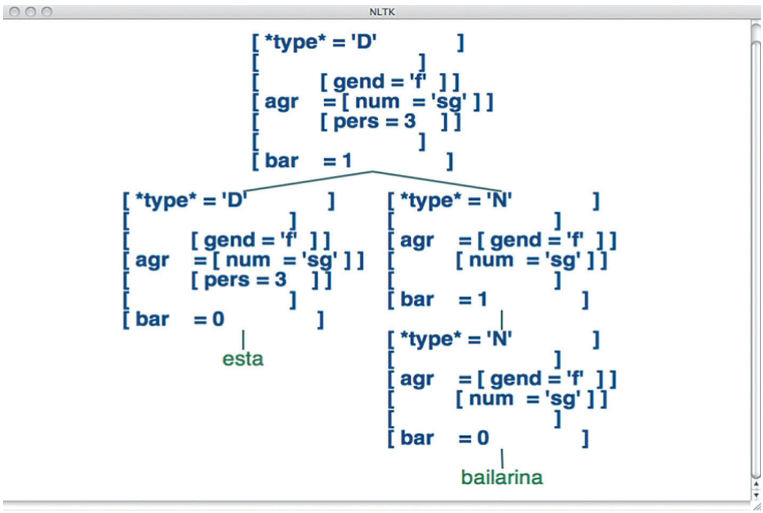
N[agr=[num='sg', gend='f'],bar=0] -> 'bailarina' | 'aluna'

D[agr=[num='sg', gend='m',pers=3],bar=0] -> 'o' | 'este'

D[agr=[num='sg', gend='f',pers=3],bar=0] -> 'a' | 'esta'

A Figura 1 apresenta o resultado da análise automática de um DP pelo *parser* construído por meio do NLTK a partir de (18). Nessa árvore, os diferentes nós não constituem categorias atômicas, como nas representações análogas da CFG. No formalismo FCFG, os nós são AVMs. As categorias sintagmáticas e lexicais são valores do atributo *type*.

Figura 1 – Representação arbórea com traços do DP esta bailarina.



Fonte: Elaboração própria.

O processamento automático da sintaxe no NLTK

O NLTK oferece módulos capazes de processar gramáticas em diversos formalismos e construir parsers por meio dos quais podemos analisar sentenças automaticamente, representando suas estruturas sob a forma de árvores. Nesta seção, exemplificamos isso por meio da construção de um parser tabular ascendente para um fragmento de gramática no formalismo CFG que modela a modificação adjetival em português conforme pressupostos tradicionais da Teoria X-barra. Veremos que a implementação computacional desse fragmento expõe uma dificuldade dessa abordagem para a qual não se tem atentado na literatura.

O Quadro 1 apresenta sinoticamente nas duas primeiras colunas os formalismos gramaticais mais importantes do NLTK e os principais tipos de parsers disponibilizados[8]. Os cinco primeiros parsers funcionam com gramáticas no formato clássico da CFG. O NLTK possibilita, também, a construção de parsers a partir de duas extensões da CFG: a gramática livre de contexto probabilística (PCFG) e a gramática livre de contexto baseada em estruturas de traços (FCFG). A terceira coluna especifica as abreviaturas pelas quais esses parsers são identificados na interface gráfica Donatus, de que trataremos na seção seguinte. A última coluna indica a existência ou não de interface gráfica do NLTK para cada tipo de parser.

Quadro 1 – Alguns dos formalismos e tipos de parser do NLTK.

Formalismo	**Tipo de parser**		**Símbolo no Donatus**	**Interface gráfica no NLTK**
CFG	recursivo-descendente		RD	sim
	shift-reduce		SR	sim
	tabular	estratégia descendente	TD	sim
		estratégia ascendente	BU	sim
		Earley	EP	não
PCFG	Viterbi		VP	não
FCFG	tabular baseado em traços		FG	não
Gramática Dependencial	*projective dependency*		DG	não

Fonte: Elaboração própria.

[8] Neste artigo, referimo-nos sempre à versão 2.0.1 do NLTK (BIRD; KLEIN; LOPER, 2012).

Para o processamento da sentença, tanto em nível sintático como semântico, há vários ambientes de desenvolvimento gratuitos atualmente disponíveis, como, por exemplo, o LKB (COPESTAKE, 2002), muito mais poderosos do que os correspondentes módulos do NLTK. O LKB, entre outras vantagens, não se limita à análise sintática e semântica automáticas, mas permite, igualmente, a geração automática de sentenças a partir de representações semânticas. De um ponto de vista didático, contudo, o NLTK se sobressai como a primeira opção a ser considerada em cursos introdutórios de sintaxe formal ou linguística computacional. Isso se deve, em primeiro lugar, ao fato de estar implementado em Python, uma linguagem especialmente acessível a não programadores. Em segundo lugar, pela ênfase que confere à linguística de corpus, o NLTK permite a integração entre o processamento de corpora e a análise automática de sentenças.

Para construir, a partir de uma dada gramática, um parser de um dos tipos listados no Quadro 1, o usuário do NLTK dispõe, em primeiro lugar, da interface de linha de comando de Python. Interfaces gráficas para os principais tipos de parser para gramáticas no formato CFG também estão disponíveis, como vemos no Quadro 1. Por meio dessas interfaces, é possível carregar uma gramática armazenada num arquivo e construir o correspondente parser, o qual pode ser aplicado na análise de uma sentença individual. Para além da facilidade de uso para não programadores, o principal atrativo dessas interfaces gráficas é a visualização passo a passo do funcionamento dos diferentes algoritmos de *parsing*. Para quem tem familiaridade com o essencial de Python, contudo, a interface de linha de comando é muito mais rápida, flexível e cômoda. Por meio de simples scripts, é possível, por exemplo, pré-processar o input utilizando estratégias customizadas de toquenização e analisar as sentenças de um texto inteiro conforme uma dada gramática e comparar o resultado com as correspondentes árvores de uma floresta sintática (*treebank*).

Vejamos como utilizar o NLTK por meio da interface de linha de comando de Python para a testagem de uma minigramática. Consideremos os sintagmas de (19), os três últimos dos quais, apesar de interpretáveis, nos parecem agramaticais por não obedecerem à ordem canônica dos adjetivos em português. Observe que (19h) seria gramatical em inglês.

(19) a. *o famoso velho vaso*

b. *o velho vaso branco*

c. *o velho vaso branco italiano*

d. *o famoso velho vaso branco italiano*

e. *o famoso velho vaso branco italiano furado*

*f. *o italiano branco furado vaso velho famoso*

*g. *o italiano branco furado velho famoso vaso*

*h. *o famoso velho furado branco italiano vaso*

Um número infinito de gramáticas no formato CFG permite gerar esses sintagmas. Com base em Carnie (2002), Klenk (2003) e Sag, Wasow e Bender (2003), construímos, com o propósito meramente didático, a minigramática (20), que implementa a análise tradicional da modificação adjetival no âmbito da Teoria X-barra. Nesse modelo, o AP é um modificador de NOM (i.e. N'), adjungindo-se tanto à esquerda quanto à direita, por meio das regras (20b) e (20c), tal como também postula Othero (2009, p.55) no quadro da hipótese DP. Desse modo, (20) não pode ser diretamente convertida numa DCG, pois contém a regra recursiva à esquerda (20c).

(20) a. NP -> D NOM

b. NOM -> AP NOM (doravante R)

c. NOM -> NOM AP (doravante T)

d. NOM -> N

e. AP -> A

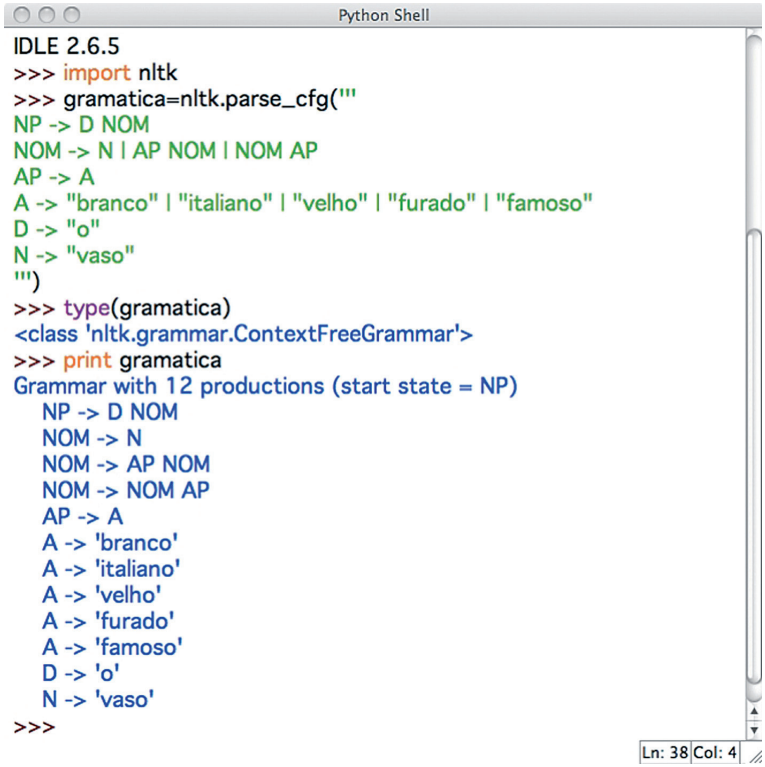
f. A -> «branco» | «italiano» | «velho» | «furado» | «famoso»

g. D -> «o»

h. N -> «vaso»

Para evidenciar a utilidade de um sistema como o NLTK no estudo da sintaxe, convidamos o leitor a calcular manualmente quantas análises são licenciadas pela gramática de (20) para cada um dos sintagmas de (19) e, em seguida, comparar seus resultados e o tempo que levou a obtê-los com os dados correspondentes obtidos por um parser tabular ascendente no NLTK. Para tanto, o primeiro passo é importar a própria biblioteca e, em seguida, transformar a gramática de (20), que constitui em Python um objeto do tipo *string* (cadeia), numa instância da classe nltk.grammar.ContextFreeGrammar. Esse procedimento é apresentado na Figura 2 abaixo.

Figura 2 – Construção de uma instância de CFG do NLTK no ambiente IDLE de Python.



Fonte: Elaboração própria.

Em (21) abaixo, construímos, inicialmente, um parser tabular ascendente a partir da instância referida pela variável *gramatica*. Em seguida, toquenizamos (19a) e aplicamos o parser sobre a lista de *tokens*, armazenando as análises na lista *arvores*.

(21)

>>> **p=nltk.ChartParser(gramatica,nltk.parse.BU_STRATEGY)**

>>> **s="o famoso velho vaso".split()**

>>> **arvores=p.nbest_parse(s)**

Agora, os comandos em (22) abaixo extraem o comprimento da lista de árvores e exibem o seu conteúdo, que vem a ser uma única árvore, primeiro sob a forma de parênteses rotulados e, depois, sob a forma de uma árvore (Figura 3).

(22)

>>> **len(arvores)**

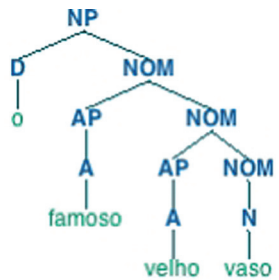
1

>>> **for arvore in arvores: print arvore**

(NP (D o) (NOM (AP (A famoso)) (NOM (AP (A velho)) (NOM (N vaso)))))

>>> **nltk.draw.draw_trees(*arvores)**

Figura 3 – Análise de um sintagma no NLTK conforme a gramática (20).



Fonte: Elaboração própria.

Como era de se esperar, o sintagma em questão tem apenas uma análise e, talvez, o leitor tenha chegado a resultado idêntico com relação aos demais sintagmas de (19). Mas vejamos se essa suposição confere com os resultados do algoritmo de *parsing*. Para verificar isso, inicialmente extraímos de um arquivo os sintagmas de (19) e armazenamos os sete últimos numa lista:

(23)

>>> **lista=open("sintagmas.txt").read().strip().split("\n")[1:]**

Em seguida, em (24), aplicamos o parser aos sintagmas da variável *lista* e exibimos a quantidade de árvores geradas para cada um dos sintagmas (19b-h).

(24)

>>> **t=[a for a in p.batch_nbest_parse([s.split() for s in lista])]**

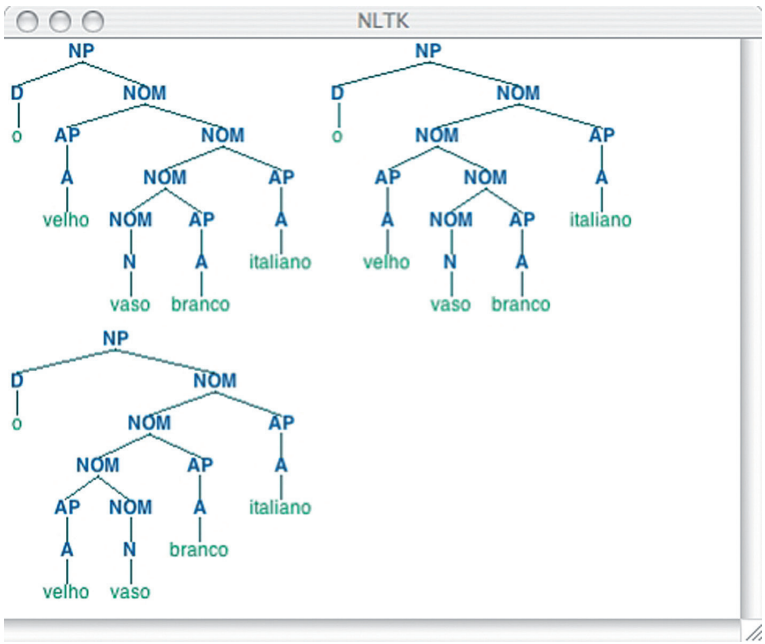
>>> **for letra,arvores in zip("bcdefgh",t):**

print "%s)%d\t" % (letra,len(arvores)),

b)2 c)3 d)6 e)10 f)10 g)1 h)1

Como vemos acima, (19b-e) têm, respectivamente, 2, 3, 6 e 10 análises! No caso dos sintagmas agramaticais, enquanto (19f) tem 10 análises, (19g) e (19h) apresentam apenas uma. Por não subclassificar os adjetivos, a minigramática de (20) hipergera, assim como hipergera gramática análoga do inglês (sem a regra T), produzindo estruturas como (19g) ao lado de (19h), pois não estabelece restrições quanto à ordem dos adjetivos.

Figura 4 – Ambiguidade de (19c) conforme a gramática de (20).



Fonte: Elaboração própria.

Se o leitor não chegou ao mesmo número de árvores para as construções (19b-e), o algoritmo estaria errado? Com certeza, não, como podemos constatar pela Figura 4 acima (gerada a partir do comando (25)), que exibe as três análises de (19c) produzidas pelo parser.

(25)

>>> **nltk.draw.draw_trees(*t[1])**

A discrepância entre os resultados do parser e os provavelmente obtidos pelo comum dos leitores deve-se a que humanos, de um modo geral, têm dificuldade para detectar ambiguidades estruturais de natureza puramente formal como essas, que não parecem repercutir na interpretação semântica nem ter qualquer consequência pragmático-discursiva. O que se exige para o cálculo da quantidade de árvores que podemos logicamente construir para os sintagmas de (19) conforme a gramática de (20) não é uma habilidade linguística, mas puramente matemática, como veremos abaixo.

É interessante comparar, em (24), as quantidades de análises dos sintagmas (19e-h), os quais aparentemente atribuem o mesmo conjunto de propriedades à entidade denotada por *o vaso*. No entanto, se cada uma das árvores geradas produzisse uma interpretação diferente, teríamos uma única interpretação para (19g) e (19h), mas 10 interpretações diferentes para (19e) e (19f). Observe que o fator determinante dessa ambiguidade é a ocorrência de adjetivos, no sintagma, tanto na posição pré-nominal quanto pós-nominal. Quando todos os adjetivos são pré-nominais, a ambiguidade desaparece, como vemos em (19a), (19g) e (19h). Isso significa que esse tipo de ambiguidade não existiria em línguas como o inglês, em que o adjetivo, exceto em casos lexicalizados como *court martial*, é apenas pré-nominal. A implementação computacional fornece, portanto, hipóteses para investigações psicolinguísticas experimentais, que procurariam determinar se falantes do inglês e do português de fato apresentam julgamentos semânticos diferentes quanto à ambiguidade dessas estruturas.

Como não percebemos diferenças interpretativas de (19c) relacionadas às árvores da Figura 4 (e o mesmo se aplica às análises de (19b-f)), parece-nos razoável considerá-las instâncias de *pseudoambiguidade* (*spurious ambiguity*)[9], considerada um problema a ser sanado especialmente no âmbito da Gramática Categorial, no qual decorre da ordem de aplicação das regras (RENZ, 1993, p.64), exatamente como nos exemplos (19b-f). Essas falsas ambiguidades não constituem, portanto, um defeito do algoritmo de *parsing*, que apenas constrói, mecanicamente, de forma rigorosa, cada uma das derivações possíveis dos sintagmas conforme (20). O leitor pode verificar na Figura 4 que cada expansão de símbolo de cada uma das três árvores é licenciada pela aplicação de uma das regras da gramática. Análises diferentes resultam porque o parser pode derivar a cadeia por meio de três sequências distintas de aplicação das regras R e T, a saber RTT, TRT e TTR. Essas são as três permutações com elementos repetidos (no caso, *anagramas*) matematicamente possíveis para RTT, número esse resultante da aplicação da fórmula (n+m)!/(n!m!) da análise combinatória (AHO;

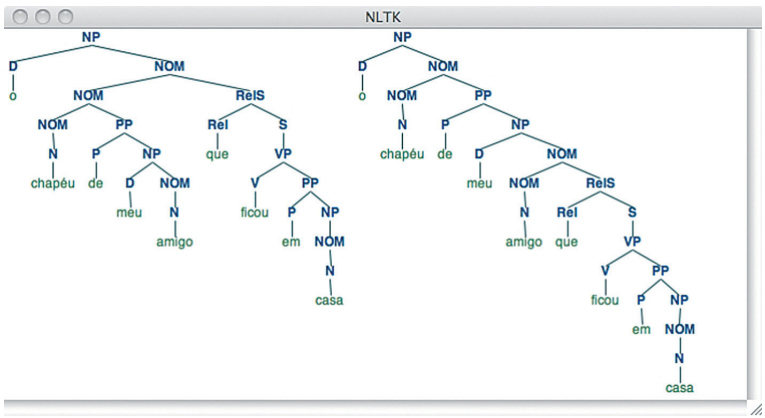
[9] Traduzimos esse termo técnico da literatura de sintaxe formal (SAG; WASOW; BENDER, 2003, p.388) por *pseudoambiguidade* para evitar a conotação pejorativa do adjetivo *espúrio*. Como sugeriu um parecerista, múltiplas análises sem contrapartidas semânticas de uma dada cadeia podem constituir uma espécie de subproduto necessário de uma determinada gramática que modela adequadamente outros fenômenos.

ULLMAN, 1994), onde m e n representam as repetições de R e T, correspondentes, respectivamente, às quantidades de adjetivos pré- e pós-nominais. No caso de (19e), o número de análises é a quantidade de anagramas de RRTTT, ou seja (2+3)!/(3!2!)=10.

A pseudoambiguidade dos exemplos de (19) decorre, portanto, da gramática (20), que, no contexto do *parsing* da CFG, não parece produzir, para essas expressões, análises condizentes com as intuições semânticas dos falantes, os quais, ao que tudo indica, não as consideram ambíguas como o fazem para sintagmas do tipo de (26), exemplo prototípico do que se convencionou designar *ambiguidade estrutural*, em que a cada árvore se associam interpretações distintas (RADFORD, 1988), como na Figura 5 abaixo.

(26) o chapéu de meu amigo que ficou em casa

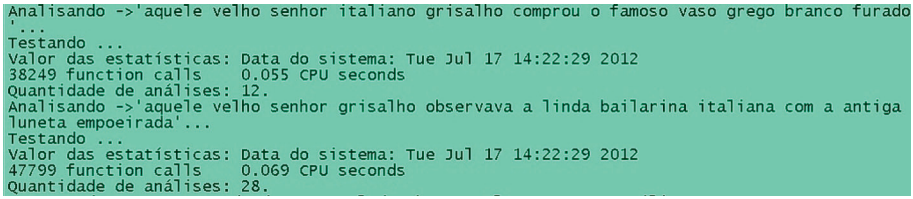
Figura 5 - Representação gráfica da ambiguidade estrutural legítima de (26).



Fonte: Elaboração própria.

A ambiguidade gerada pelas regras R e T pode vir a representar um problema para aplicações alimentadas pelo *parsing* sintátic, no qual uma desambiguação se faça necessária (LJUNGLÖF; WIRÉN, 2010, p.81). De fato, em uma gramática de maior cobertura essas falsas ambiguidades interagem entre si e com casos de ambiguidade estrutural legítima para produzir um número explosivo de análises, como vemos na Figura 6.

Figura 6 – Explosão no número de análises por meio da interação entre ambiguidades estruturais falsas e legítimas (output do Donatus para uma versão expandida de (20)).



Fonte: Elaboração própria.

Esse exemplo evidencia a utilidade da implementação computacional de modelos formais da linguagem humana, seja como instrumento de verificação da sua aplicabilidade na tecnologia da linguagem natural, seja como fonte de hipóteses para testar a sua plausibilidade psicológica. Naturalmente, um sintaticista dotado de especiais habilidades matemáticas chegaria aos mesmos resultados que o parser de (21) prescindindo do computador, computando manualmente os vários passos do algoritmo de *parsing*, mas isso, pelo menos na maioria dos casos, demandaria um certo esforço e um tempo razoável. No NLTK, tirando o tempo que leva a digitação de comandos como os de (21) – (24), a análise propriamente dita das construções de (19) leva menos de um segundo. No Donatus, essa análise é feita de forma ainda mais rápida.

A seguir, depois de explicar, na próxima seção, o funcionamento do Donatus, propomos, na quinta seção, uma abordagem computacional alternativa da modificação adjetival em português baseada no Programa Minimalista, abordagem essa que evita as pseudoambiguidades de exemplos como os de (19), ao mesmo tempo que faz jus às intuições que motivaram a categoria N' na Teoria X-barra tradicional.

A interface gráfica Donatus

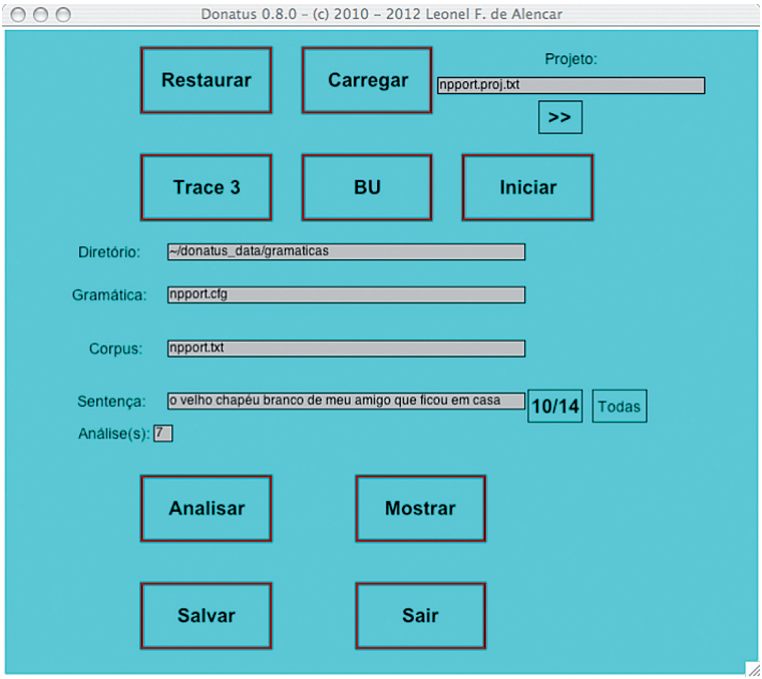
Apesar de sobressair, entre as linguagens de programação atuais, como extremamente amigável, sobretudo num contexto de aprendizagem da programação de computadores, Python não é uma "linguagem de brinquedo" (*toy language*) (ZELLE, 2004). Por outro lado, no NLTK, apenas parsers para gramáticas no formato da CFG dispõem de interfaces gráficas (Quadro 1). Isso significa que não é possível construir um parser de uma gramática no formalismo FCFG e aplicá-lo na análise sintática de textos sem dominar os princípios básicos e a sintaxe de comandos de Python, de que damos uma amostra acima.

Aprender programação constitui um pré-requisito na linguística computacional. No estudo da sintaxe formal, porém, isso pode ser evitado ou, pelo menos, adiado. Tendo em vista, primeiramente, um público-alvo constituído de alunos típicos de cursos de graduação em Letras e pós-graduação em Linguística brasileiros, desenvolvemos o Donatus[10], uma interface gráfica para alguns dos principais algoritmos de parsing do NLTK (Quadro 1). Veremos que essa interface é útil também a linguistas programadores, por automatizar rotinas na construção e testagem de gramáticas e permitir um acesso cômodo, por meio de poucos cliques, a recursos que não integram o repertório do NLTK.

Para exemplificar o funcionamento do Donatus, retomemos, inicialmente, a Figura 5, exemplo de ambiguidade legítima gerada por uma versão expandida de (20). Qual a quantidade de análises licenciada por essa gramática para uma variante de (26) com [velho chapéu branco] em vez de [chapéu]? O Donatus permite responder facilmente a essa pergunta. A interface do programa é constituída de botões clicáveis e campos de texto para fornecimento de input pelo usuário ou exibição de output (Figura 7 abaixo). A maneira mais prática de construir, nessa ferramenta, um parser para uma gramática e aplicá-lo na análise sintática é criando inicialmente um *arquivo de projeto*, especificando o nome dos arquivos da gramática e do corpus a ser analisado, o tipo de parser etc. Dessa forma, podemos criar um parser tabular ascendente (BU na Figura 7, conforme o Quadro 1) para uma versão da gramática (20) que contemple também (26) e aplicá-lo na análise de um corpus por meio de apenas três cliques, respectivamente nos botões **Carregar**, **Todas** e **Analisar**.

[10] O programa, cujo nome homenageia o gramático latino a quem se deve o termo *parse* (JUNGEN; LOHNSTEIN, 2006), está disponível para *download* a partir da URL <http://donatus.sourceforge.net/>.

Figura 7 – Tela do Donatus mostrando a quantidade de análises de exemplo análogo a (26) gerada por um parser a partir de uma versão ampliada de (20).



Fonte: Elaboração própria.

O botão **Todas** funciona como um comutador, passando a exibir, ao ser clicado, a cor azul escura. Quando ligado, são analisadas todas as construções do corpus; quando desligado (como na Figura 7), o parser é aplicado à construção exibida no campo **Sentença**. O campo **Análise(s)** exibe a quantidade de árvores atribuídas pelo algoritmo à última construção dada para análise, que totalizam 7 no exemplo em tela, evidenciando a interação entre ambiguidade estrutural legítima e pseudoambiguidade (Figura 7).

A análise em bloco de um grande conjunto de construções, armazenadas num arquivo de corpus, constitui uma vantagem do Donatus em comparação com as interfaces gráficas que o próprio NLTK oferece para a construção e manipulação de algoritmos de *parsing* da CFG. Nessas últimas, temos de analisar cada construção individualmente, num processo bastante moroso, exigindo a digitação de cada construção ou o uso dos comandos de copiar e colar. No Donatus, naturalmente, também podemos analisar construções individualmente, só que de maneira muito mais prática que nas interfaces gráficas do NLTK. Uma

vez carregado um arquivo de corpus, podemos percorrer de uma em uma a lista de construções a serem analisadas, clicando no botão que fica entre o campo de texto **Sentença** e o botão **Todas**.

Para cada construção analisada, são exibidas, entre outras, as seguintes informações na janela do Terminal: (i) quantidade de operações (*function calls*), (ii) tempo de execução em segundos (medido em "CPU seconds")[11] e (iv) quantidade de análises. Em (27), isso é exemplificado pela aplicação de (20) aos exemplos (19). As duas primeiras informações são fundamentais para avaliar a eficiência de gramáticas e algoritmos de *parsing*, permitindo determinar a modelação mais eficiente de um determinado fenômeno gramatical. Trata-se de recurso não disponível no NLTK cuja implementação exige conhecimentos avançados de Python.

(27)

[...]Analisando ->'o famoso velho vaso branco italiano furado'...

14396 function calls 0.032 CPU seconds

Quantidade de análises: 10. [...]

100.00% de construções (8 de um total de 8) com pelo menos uma análise.

4.25 análise(s) por construção.

Para visualização das análises atribuídas à última construção analisada, basta um clique no botão **Mostrar**. O Donatus exibe, então, numa janela adicional, até 8 das representações arbóreas geradas pelo parser. No Terminal são apresentadas as árvores na notação de parênteses rotulados.

Se o usuário precisar interromper o trabalho e fechar o Donatus, o programa dispõe da possibilidade de armazenar as especificações da sessão por meio do botão **Salvar** e retomá-las posteriormente com um clique no botão **Restaurar**.

A eliminação de pseudoambiguidades no *parsing* do DP

Vimos acima a dificuldade de calcular a quantidade de árvores que se podem atribuir a exemplos triviais do tipo de (19b-d) conforme uma gramática bastante simples, como (20). O parser tabular ascendente construído a partir dessa CFG, que implementa a abordagem tradicional da modificação adjetival na Teoria X-barra, gera um grande número de pseudoambiguidades para esses sintagmas.

[11] Medições realizadas num computador com processador Intel Core 2 Duo de 2.16 GHz e 2 GB de RAM, rodando o sistema operacional MAC OS 10.4.11.

A ambiguidade salta de 6 árvores em (19d) para 10 análises em (19e), por conta de um único adjetivo a mais em posição pós-nominal.

Como construir uma gramática linguisticamente fundamentada para as construções de (19) que não gere análises em excesso? Abordagens mais recentes do DP no âmbito do Programa Minimalista fornecem subsídios para resolver esse problema de uma maneira que nos parece elegante. Nesta seção, implementamos uma análise do DP em português nesses moldes no quadro da gramática de unificação, sem recorrer, portanto, a transformações sintáticas. Para tanto, utilizamos o formalismo FCFG do NLTK e o Donatus, que permite uma rápida testagem de uma gramática com base num grande número de exemplos bem como uma fácil visualização das árvores geradas.

Segundo Bernstein (2003, p.547-553), o contraste na posição do adjetivo entre línguas românicas e germânicas, em exemplos como (28) e (29), não se deve a uma geração dessa categoria em posições diferentes em relação ao N nos dois grupos de línguas. Em vez disso, o AP é gerado nesses casos apenas em adjunção à esquerda e o NP se move, no primeiro caso, para uma posição intermediária entre N e D, movimento esse ausente no segundo caso. Desse modo, essa abordagem, tal como a análise X-barra tradicional de (20), trata apenas da ordem do AP em relação ao NP, sem estipular restrições sobre a ordem de subclasses de adjetivos (determinando, por exemplo, que adjetivos de cor precedem adjetivos de origem em inglês) ou a posição pré-nominal ou pós-nominal de determinadas subclasses no caso de línguas como português e francês (que especificariam, por exemplo, pelo menos uma preferência para a posição pós-nominal de adjetivos de cor e de origem em português).

(28)	*a.*	*un chapeau noir*	*(francês)*
	b.	*um chapéu preto*	*(português)*
(29)	*a.*	*a black hat*	*(inglês)*
	b.	*ein schwarzer Hut*	*(alemão)*

Na esteira de Bernstein (2003), entre outros, postulamos uma categoria funcional Num para representar o número gramatical, cuja projeção máxima NumP é complemento da categoria D; o núcleo funcional Num tem, por sua vez, o NP como complemento, que corresponde ao NOM ou N' de (20) (RAPOSO, 1992, p.213-215)[12]. Nesse quadro, analisamos a posição pós-nominal do adjetivo

[12] Em Othero (2009, p.55-62), a categoria Num não representa a categoria funcional de número (que compreende os traços singular e plural, em português), mas numerais como *dois*, *três* etc.

em português, em exemplos do tipo de (28b), como resultado de movimento do NP para Num, por sobre o AP[13].

Em português, contudo, a exemplo de outras línguas românicas, o adjetivo também pode ocorrer pré-nominalmente, como [velho] em (30a). Analisamos esse exemplo como resultado do movimento do NP [velho chapéu] por sobre o AP [vermelho] (ver (30b)) .

(30) a. o velho chapéu vermelho

b. o velho chapéu vermelho ~~velho chapéu~~

Analogamente, entendemos que, em (31a) e (31b), o NP movido "arrasta" consigo dois APs, deixando *in situ*, em (31b), o AP [vermelho][14].

(31) a. o famoso velho chapéu ~~famoso velho chapéu~~

b. o famoso velho chapéu vermelho ~~famoso velho chapéu~~

O fragmento de gramática em (32) implementa essa análise do DP no formalismo FCFG.

```
(32) D[agr=?a,bar=1] -> D[agr=?a,bar=0] Num[agr=?a,bar=1]
     Num[agr=?a,bar=1] -> Num[agr=?a,bar=0] N[agr=?a,bar=1]/?x
     Num[agr=?a,bar=0] -> N[agr=?a,bar=1]
     N[agr=?a,bar=1] -> N[agr=?a,bar=0] | A[agr=?a,bar=1] N[agr=?a,bar=1]
     N[agr=?a,bar=1]/?x -> A[agr=?a,bar=1] N[agr=?a,bar=1]/?x
     N[bar=1]/N[bar=1] ->
     A[agr=?a,bar=1] -> A[agr=?a,bar=0]
```

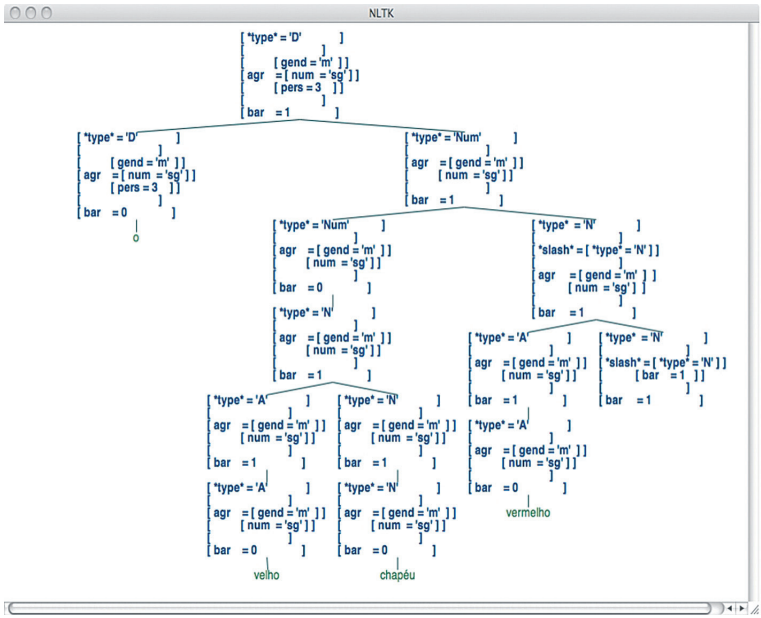
A FCFG permite expressar a noção de movimento da gramática transformacional, exemplificada nas análises (30b) e (31b), sem recorrer a transformações, graças às categorias-barra, que permitem modelar a relação de dependência entre o NP extraído e o seu vestígio, representados na Figura 8, respectivamente, por [velho chapéu] e N[bar=1]/N[bar=1] (i.e. NP/NP), categoria que se expande em (32) por uma cadeia vazia. Na notação do NLTK, Y/XP

[13] A rigor, o local de pouso de um XP movido não deve ser uma posição nuclear tal como Num na Figura 8, mas uma projeção máxima (possivelmente Spec, NumP). Trataremos dessa questão em um trabalho futuro.

[14] Em (31a), o movimento do NP não é necessário para derivar a posição pré-nominal dos APs. No entanto, parece-nos preferível um tratamento uniforme do NP em português, segundo o qual essa categoria é sempre extraída da sua posição-base. Essa é uma questão que deixamos para aprofundar em investigações futuras.

denota uma categoria Y que possui uma lacuna (*gap*) do tipo XP, a qual deve ser preenchida por uma categoria (*filler*) desse tipo num lugar mais alto na árvore (BIRD; KLEIN; LOPER, 2009, p.349-350).

Figura 8 – Análise de (30a) conforme (32)



Fonte: Elaboração própria.

Aplicada aos exemplos de (19), um parser construído pelo NLTK a partir de (32) produz apenas uma análise para cada uma dessas construções, contrariamente ao parser construído a partir da CFG (20). Esse último parser produz cada vez mais análises sem evidentes correlatos interpretativos à medida em que se aumenta o número de adjetivos pré- e pós-nominais no DP. Por exemplo, se incluímos apenas um adjetivo a mais no DP (19e) na posição pós-nominal (ver (33)), o parser da CFG de (20) passa a atribuir 15 análises à estrutura resultante[15]. No parser da FCFG de (32), porém, apenas uma única análise continua a ser gerada.

(33) o famoso velho vaso branco italiano furado sujo

É importante ressaltar que (32), a exemplo de (20), são apenas minigramáticas, construídas com o propósito de demonstrar a utilidade do *parsing* no estudo da

[15] Esse é o número de anagramas de RRTTTT, que se pode obter pela fórmula 6!/(2!4!). Analogamente, o acréscimo de mais um adjetivo em (33) na posição pré-nominal eleva o número de análises para 35.

sintaxe formal. Tal como (20), (32) hipergera, uma vez que também não modela as restrições de linearização dos adjetivos[16]. De fato, abstraindo dos traços de concordância, implementados em (32), as duas gramáticas são matematicamente equivalentes, na medida em que geram o mesmo conjunto de cadeias (*string language*) (MARTÍN-VIDE, 2004, p.160).

A coordenação simples em exemplos como (34) tem sido apresentada como argumento em prol da categoria NOM da teoria X-barra tradicional (RADFORD, 1988, p.174), permitindo expressá-la como instância da regra esquemática XP -> XP Conj XP (CARNIE, 2002, p.54).

(34) aquelas divertidas alunas e meninas bagunceiras

Como as duas propostas de modelação da modificação adjetival subjacentes, respectivamente, a (20) e a (32) se comportam diante do fenômeno da coordenação simples? É fato conhecido que esse tipo de estrutura pode resultar em casos de ambiguidade estrutural legítima, decorrente das diferentes possibilidades de escopo dos adjetivos (JURAFSKY; MARTIN, 2009, p.467). Será que a nossa proposta alternativa, por eliminar as pseudoambiguidades na análise de exemplos como (19), erroneamente geraria apenas uma análise para (34)? No contexto do *parsing*, para obter resposta a essa pergunta basta implementar as duas regras de (35) na gramática (32). Analogamente, a inclusão de (35b) em (20) (substituindo NP por NOM) permite que essa gramática gere (34).

(35) a. NumP -> NumP Conj NumP

b. NP -> NP Conj NP

Como se pode verificar manualmente ou construindo, por meio do Donatus, um parser para essa nova gramática, são geradas três árvores para (34). Que se possa tão facilmente implementar a coordenação simples no âmbito do DP em nossa abordagem alternativa não surpreende, uma vez que herda as virtudes da abordagem anterior, ao retomar a categoria NOM sob a forma de NP (= N'), necessária também para formular a pronominalização por meio de *one* em inglês (RADFORD, 1988, p.175). Nossa proposta evita as falsas ambiguidades constatadas no parser de (20) em exemplos como (19c) porque, em vez das duas regras R e T dessa gramática, inclui apenas a primeira, responsável por APs pré-nominais. APs pós-nominais são gerados em adjunção à esquerda a um *gap* do tipo NP, por meio da regra NP/NP -> AP NP/NP.

[16] Pretendemos implementar essas restrições num trabalho futuro, recorrendo ao formalismo da LFG, que permite expressar em regras separadas as relações de dominância e as relações de precedência entre constituintes (FALK, 2001). Para tanto, a FCFG do NLTK é muito limitada, uma vez que não dispõe desse recurso.

Por meio do Donatus, podemos facilmente calcular a quantidade de operações realizadas e o tempo consumido na análise de uma determinada cadeia por um determinado parser, tal como exemplificado em (27). Esse recurso, que, como vimos, inexiste no NLTK, permite comparar o desempenho de diferentes parsers. A análise de (19e) pelo parser da FCFG (32) revela-se bastante rápida, como vemos em (36).

(36) Analisando ->'o famoso velho vaso branco italiano furado'...

263290 function calls 0.383 CPU seconds

Quantidade de análises: 1.

Comparando (27) com (36), porém, verificamos que a primeira análise consumiu apenas 8.36% do tempo dessa última. Essa vantagem da abordagem de (20), contudo, deve-se, sobretudo, ao formalismo CFG, cujo *parsing*, por não ter de verificar traços, é mais rápido do que o da FCFG[17]. De fato, quando convertemos (20) para o formalismo FCFG, nos moldes de (37), a análise de (19e) passa a consumir 191131 *function calls* e 0.303 *CPU seconds*, respectivamente 72.59% e 79.11% dos valores correspondentes de (36).

(37) a. D[agr=?a,bar=1] -> D[agr=?a,bar=0] N[agr=?a,bar=1] (cf. (20a))

b. N[agr=?a,bar=1] -> A[agr=?a,bar=1] N[agr=?a,bar=1] (cf. (20b))

c. N[agr=?a,bar=1] -> N[agr=?a,bar=1] A[agr=?a,bar=1] (cf. (20c))

d. N[agr=?a,bar=1] -> N[agr=?a,bar=0] (cf. (20d)) etc.

Ao nosso ver, essa desvantagem de (32) em relação a (37) compensa pela não geração de pseudoambiguidades, cuja eliminação no âmbito do *parsing* da CFG e FCFG não constitui um processo trivial como se poderia pensar à primeira vista. A simples escolha arbitrária de uma das análises produzidas pelo parser resolveria o problema para os casos de (19) no contexto da minigramática (20) ou (37). Esse procedimento *ad hoc*, contudo, seria inviável numa gramática maior, pois eliminaria os casos de ambiguidade legítima como (26) e (34), os quais, como mostramos na Figura 6, pode interagir com os casos de pseudoambiguidade para gerar um número explosivo de análises para exemplos relativamente triviais. Essa quantidade excessiva de análises, por sua vez, representaria um problema para sistemas como tradutores automáticos baseados em interlíngua, em que o input da análise semântica é o output de um parser sintático, haja vista a necessidade de um resultado desambiguado (JURAFSKY; MARTIN, 2009).

[17] A verificação de traços, porém, é imprescindível para o tratamento de fenômenos como a concordância e a subcategorização. Isso evidencia que uma maior velocidade de *parsing* não é o único fator a ser considerado para determinar qual de duas abordagens é a mais eficiente.

Conclusões

Neste trabalho, defendemos o ponto de vista de que a CFG constitui um dos fundamentos da sintaxe formal. Mostramos que o enriquecimento das regras de uma CFG com estruturas de traços permite superar as deficiências desse formalismo que motivaram Chomsky a apelar para modelos mais poderosos, como a gramática transformacional. Ao possibilitar a construção de parsers tanto para a CFG quanto para extensões desse formalismo, como a FCFG, baseada na unificação de traços, o NLTK constitui valioso instrumento para o estudo da sintaxe sob uma perspectiva gerativa, pois permite extrair automaticamente todas as consequências de uma determinada análise de um dado fenômeno. Exemplificamos isso, primeiramente, por meio de um parser tabular ascendente para uma CFG que implementa a análise tradicional da modificação adjetival na Teoria X-barra clássica. A implementação computacional revela uma desvantagem dessa abordagem, decorrente do excesso de pseudoambiguidades geradas pelo parser. Esse problema não é evidenciado com tanta facilidade sem o recurso do computador. Apesar de ter sido desenvolvido com um propósito didático, o NLTK exige conhecimentos de programação em Python quando se trata de construir parsers para gramáticas mais complexas e aplicá-los em um grande número de exemplos. Como uma ponte que permite superar o fosso separando a maioria dos estudantes de Letras e de Linguística da sintaxe computacional, construímos o Donatus, uma interface gráfica amigável para as facilidades de *parsing* do NLTK, dotada de recursos não disponíveis nessa biblioteca, sendo, portanto, interessante também para programadores. Exemplificamos a ferramenta por meio de uma implementação não transformacional, no formalismo FCFG, de uma abordagem da modificação adjetival em português inspirada em proposta no quadro do Programa Minimalista. Esse parser elimina as pseudoambiguidades produzidas pela primeira abordagem, embora seja menos rápido do que um parser de uma FCFG que implementa a abordagem tradicional. Acreditamos que, num parser de maior cobertura, essa desvantagem seria compensada pela diminuição do custo de desambiguação, necessária em várias aplicações de PLN.

ALENCAR, L. F. de. Donatus: a user-friendly interface for the study of formal syntax using the Python NLTK Library. *Alfa*, São Paulo, v.56, n.2, p.523-555, 2012.

- *ABSTRACT: This paper firstly aims at showing the usefulness of CFG and FCFG in the study of formal syntax. Applying parsers based on these formalisms on the analysis of a corpus may reveal consequences from an approach which would otherwise pass by unnoticed. The Natural Language Toolkit (NLTK) comprises, among other facilities, generator tools for parsers in a variety of architectures. However, the non-trivial use of this library in automatic syntactic processing requires programming skills. In order to allow non-programmers access*

to parser implementation and testing, we developed Donatus, a user-friendly graphical interface to NLTK's parsing facilities with additional utilities that make it also useful to programmers. We explain the tool's functioning and demonstrate its relevance to formal syntactical investigation by means of a comparison between the computer implementations of two alternative approaches to adjectival modification in Portuguese. The first approach, based on traditional X-bar theory, generated a great number of false ambiguities. This problem was avoided by a parser based on an approach within the Minimalist Program. Without resorting to the computer, this difference between the two approaches would not be easily revealed.

- *KEYWORDS: Computational linguistics. Formal syntax. Generative grammar. X-bar theory. Context-free grammar. Unification grammar. Adjectival modification.*

REFERÊNCIAS

AHO, A. V.; ULLMAN, J. D. *Foundations of computer science*: C Edition. Nova Iorque: W.H. Freeman, 1994.

BACH, E. *Syntactic theory*. Nova Iorque: Holt, Rinehart and Winston, 1974.

BERNSTEIN, J. B. The DP Hypothesis: identifying clausal properties in the nominal domain. In: BALTIN, M.; COLLINS, C. (Org.). *The handbook of contemporary syntactic theory*. Malden: Blackwell, 2003. p.536-561.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural language toolkit*. Disponível em: <http://nltk.org/>. Acesso em: 17 jul. 2012

_____. *Natural language processing with Python*: analyzing text with the natural language Toolkit. Sebastopol: O'Reilly, 2009.

BLACKBURN, P.; BOS, J.; STRIEGNITZ. K. *Learn prolog now!* Londres: College Publications, 2006.

CARNIE, A. *Syntax*: a generative introduction. Malden: Blackwell, 2002.

CHAMETZKY, R. A. Phrase structure. In: HENDRICK, R. (Org.). *Minimalist syntax*. Malden: Blackwell, 2003. p.192-225.

CHOMSKY, N. *The minimalist program*. Cambridge: MIT Press, 1995.

_____. *Aspekte der syntax-theorie*. Tradução de Ewald Lang. Frankfurt: Suhrkamp, 1973.

_____. *Syntactic structures*. Mouton: The Hague, 1957.

COPESTAKE, A. *Implementing typed feature structure grammars*. Stanford: CSLI, 2002.

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

FUKUI, N. Phrase structure. In: BALTIN, M.; COLLINS, C. (Org.). *The handbook of contemporary syntactic theory*. Malden: Blackwell, 2003. p.536-561.

GREWENDORF, G. *Minimalistische syntax*. Tübingen: Francke, 2002.

JUNGEN, O.; LOHNSTEIN, H. *Einführung in die grammatiktheorie*. München: W. Fink, 2006.

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

KAPLAN, R. M. Syntax. In: MITKOV, R. (Org.). *The Oxford handbook of computational linguistics*. Oxford: OUP, 2004. p.70-90.

KLABUNDE, R. Automatentheorie und formale Sprachen. In: CARSTENSEN, K. U. et al. (Org.). *Computerlinguistik und sprachtechnologie*: eine einführung. 2.ed. Heidelberg: Elsevier, 2004. p.63-90.

KLENK, U. *Generative syntax*. Tübingen: Narr, 2003.

LJUNGLÖF, P.; WIRÉN, M. Syntactic parsing. In: INDURKHYA, N.; DAMERAU, F. J. (Org.). *Handbook of natural language processing*. 2.ed. Boca Raton: Chapman & Hall, 2010. p.59-91.

MARTÍN-VIDE, C. Formal grammars and languages. In: MITKOV, R. (Org.). *The Oxford handbook of computational linguistics*. Oxford: OUP, 2004. p.157-177.

MATEESCU, A.; SALOMAA, A. Formal languages: an introduction and a synopsis. In: ROZENBERG, G.; SALOMAA, A. (Org.). *Handbook of formal languages*. Berlin: Springer, 1997. v.1. p.1-39.

MIOTO, C.; SILVA, M. F.; LOPES, R. *Novo manual de sintaxe*. 2.ed. Florianópolis: Insular, 2005.

NIVRE, J. Statistical parsing. In: INDURKHYA, N.; DAMERAU, F. J. (Org.). *Handbook of natural language processing*. 2.ed. Boca Raton: Chapman & Hall, 2010. p.237-266.

OTHERO, G. A. *A gramática da frase em português*: algumas reflexões para a formalização da estrutura frasal em português. Porto Alegre: Ed. da PUCRS, 2009. Disponível em:<http://www.pucrs.br/edipucrs/gramaticadafrase.pdf> Acesso em: 02 ago. 2010.

_____. *Teoria X-barra*: descrição do português e aplicação computacional. São Paulo: Contexto, 2006.

PRATT-HARTMANN, I. Computational complexity in natural language. In: CLARK, A.; FOX, C.; LAPPIN, S. (Org.). *The handbook of computational linguistics and natural language processing*. Malden: Wiley & Blackwell, 2010. p.43-73.

RADFORD, A. *Transformational grammar*: a first course. Cambridge: CUP, 1988.

RAPOSO, E. P. *Teoria da gramática*: a faculdade da linguagem. Lisboa: Caminho, 1992.

RENZ, I. *Adverbiale im Deutschen*: ein vorschlag zu ihrer klassifikation und unifikationsbasierten Repräsentation. Tübingen: Niemeyer, 1993.

RODRIGUES, E. S.; AUGUSTO, M. R. A. Modelos formais de gramática: o programa minimalista vs. gramáticas baseadas em restrições: HPSG e LFG. *Matraga*, Rio de Janeiro, v.16, n. 24, p.133-149, 2009. Disponível em: <http://www.pgletras.uerj.br/matraga/matraga24/arqs/matraga24a06.pdf> Acesso em: 25 ago. 2011.

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

WAY, A. Machine translation. In: CLARK, A.; FOX, C.; LAPPIN, S. (Org.). *The handbook of computational linguistics and natural language processing*. Malden: Wiley & Blackwell, 2010. p.531-573.

WINTNER, S. Formal language theory. In: CLARK, A.; FOX, C.; LAPPIN, S. (Org.). *The handbook of computational linguistics and natural language processing*. Malden: Wiley & Blackwell, 2010. p.11-42.

ZELLE, J. M. *Python programming:* an introduction to computer science. Wilsonville: Franklin, Beedle & Associates, 2004.

Recebido em 26 de setembro de 2011.

Aprovado em 20 de agosto de 2012.