DJALMA PADOVANI

Um método adaptativo para análise sintática do Português Brasileiro

São Paulo
2022

DJALMA PADOVANI

Um método adaptativo para análise sintática do Português Brasileiro

Versão Corrigida

Tese apresentada à Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a obtenção do título de Doutor em Ciências.

São Paulo
2022

DJALMA PADOVANI

Um método adaptativo para análise sintática do Português Brasileiro

Versão Corrigida

Tese apresentada à Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a obtenção do título de Doutor em Ciências.

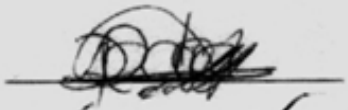
Área de concentração: Engenharia de Computação

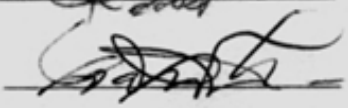
Orientador: Prof. Dr. João José Neto

São Paulo
2022

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 11 de abril de 2022

Assinatura do autor: 

Assinatura do orientador: 

Catalogação-na-publicação

Padovani, Djalma
Um método adaptativo para análise sintática do Português Brasileiro / D. Padovani -- versão corr. -- São Paulo, 2022.
216 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Processamento de linguagem natural 2.Análise morfossintática 3.Gramática transformacional 4.Autômatos finitos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

DEDICATÓRIA

Dedico este trabalho à minha esposa Lúcia, minha filha Mariana e aos meus pais Luiz e Marinha.

AGRADECIMENTOS

À minha querida esposa Lúcia, que sempre esteve ao meu lado em todos os momentos.

Á minha amada filha Mariana, que sempre me apoiou.

Aos meus pais Luiz e Marinha e aos meus irmãos Marisa, Ronaldo e Roberto, pelo incentivo aos estudos.

Aos professores Fernando Giorno, Reginaldo Arakaki, Edit Campos e Mário Miyake, que me ajudaram nos primeiros passos da vida acadêmica.

Ao professor João Jose Neto, que em todos esses anos de convivência, foi muito mais do que um orientador, contribuindo não apenas para meu crescimento científico e intelectual, mas também pessoal.

Aos professores da comissão julgadora Marcus Vinicius Midena Ramos, César Alberto Bravo Pariente, Ariane Machado Lima e Ricardo Luis de Azevedo da Rocha, pelas opiniões e comentários.

Aos amigos Newton Kiyotaka Miura e Paulo Roberto Cereda, companheiros de WTA e WAT, pelo apoio e troca de experiências.

À Escola Politécnica da USP, pela oportunidade de realização do curso de doutorado.

A todos que colaboraram direta ou indiretamente para a realização deste trabalho.

RESUMO

PADOVANI, DJALMA. **Um método adaptativo para análise sintática do Português Brasileiro** / D Padovani. São Paulo, 2022. Tese (Doutorado)-Escola Politécnica da Universidade de São Paulo.216p.

As línguas naturais caracterizam-se por sua riqueza semântica, léxica e sintática, permitindo a elaboração de textos complexos com alto grau de abstração como os vistos nas grandes obras da literatura, ou precisos e direcionados, como os encontrados em tratados acadêmicos e trabalhos científicos. Há um grande apelo para que as pessoas se comuniquem com as máquinas da mesma forma que fazem umas com as outras. No entanto, o processamento de linguagens naturais requer o desenvolvimento de programas capazes de determinar e interpretar a estrutura léxico-sintática e semântica das sentenças em vários níveis de detalhe. A análise sintática, também conhecida como *parsing*, é um dos principais componentes em várias aplicações de processamento de linguagem natural, porém se trata de uma tarefa complexa por causa das ambiguidades da linguagem, múltiplas interpretações de palavras, diferentes ordens possíveis de elementos e itens ausentes. Além disso, as línguas mais conhecidas se beneficiam de um número significativo de recursos computacionais, enquanto as demais, entre elas o Português, não dispõem de tantas ferramentas dessa natureza. Esta tese tem como objetivo apresentar um método para análise sintática do Português Brasileiro. O formalismo adaptativo foi escolhido como modelo teórico subjacente devido à sua potencial riqueza de representação e de manipulação, o que o torna consistente e flexível ao mesmo tempo, proporcionando o embasamento necessário para a construção do modelo computacional proposto, sem a necessidade de recorrer a técnicas auxiliares. O modelo proposto foi validado através de experimentos nos quais os resultados foram comparados aos obtidos pelos analisadores sintáticos do estado da arte para o Português, visando avaliar a sua eficiência nos diversos cenários de testes.

Palavras-Chaves: Processamento de Linguagem Natural. Analisadores Morfossintáticos. Gramáticas. Autômatos.

ABSTRACT

PADOVANI, DJALMA. **An adaptive method for syntactic analysis of Brazilian Portuguese** / D Padovani. São Paulo, 2022. Tese (Doutorado)-Escola Politécnica da Universidade de São Paulo.216p.

Natural languages are characterized by their semantic, lexical and syntactic richness, allowing the elaboration of complex texts with a high degree of abstraction, such as those seen in great works of literature, or precise and directed, as found in academic treatises and scientific works. There is a huge appeal for people to communicate with machines the same way they do with each other. However, natural language processing requires the development of programs capable of determining and interpreting the lexical-syntactic and semantic structure of sentences at various levels of detail. Parsing is one of the main components of many natural language processing applications, but it is a complex task because of language ambiguities, multiple word interpretations, different possible orders of elements and missing items. In addition, the most popular languages benefit from a significant number of computational resources, while the others, including Portuguese, do not have as many tools of this nature. This thesis aims to present a method for syntactic analysis of Brazilian Portuguese. The adaptive formalism was chosen as the underlying theoretical model because of its potential richness of representation and manipulation, which makes it consistent and flexible at the same time, providing the necessary foundation for the construction of the proposed computational model, without the need to resort to auxiliary techniques. The proposed model was validated through experiments in which the results were compared to those obtained by state-of-the-art syntactic analyzers for Portuguese, in order to evaluate the efficiency of the model in different test scenarios.

Keywords: Natural Language Processing. Parsers. Grammars. Automata.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

BERT	Bidirectional Encoder Representations from Transformers
CYK	Cocke Younger Kasami
GLC	Gramática Livre de Contexto
GPLC	Gramática Probabilística Livre de Contexto
GPLCL	Gramática Probabilística Livre de Contexto Lexicalizada
GLUE	General Language Understanding Evaluation
LAS	Labeled Attachment Score
LSTM	Long Short-Term Memory
MGL	Modelos Globais Lineares
PLN	Processamento da Linguagem Natural
PoS	Part-of-speech
SVM	Support Vector Machine
UAS	Unlabeled Attachment Score

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma introdução sobre a tese, a justificativa, os objetivos e a estrutura geral do trabalho. Procura-se com isso situar o leitor de forma que possa entender o contexto segundo o qual a pesquisa foi realizada.

1.1 Justificativa

As línguas naturais caracterizam-se por sua riqueza semântica, léxica e sintática, permitindo a elaboração de textos complexos com alto grau de abstração como os vistos nas grandes obras da literatura, ou precisos e direcionados como os encontrados em tratados acadêmicos e trabalhos científicos. As línguas naturais não ficam estanques, ao contrário, estão em constante evolução, agregando novos termos e estruturas e eliminando outros em um constante processo de adaptação às necessidades impostas pela realidade.

Há um grande apelo para que as pessoas se comuniquem com as máquinas da mesma forma que fazem umas com as outras. Muitas pessoas encontram dificuldades para usar dispositivos convencionais de interação com computadores, o que, em certa medida, restringe o uso de possibilidades linguísticas para que instruções possam ser analisadas, interpretadas e convertidas em comandos que a máquina possa executar. Tais limitações podem gerar desconforto e, em alguns casos, levar à rejeição; portanto, há uma grande demanda por computadores e sistemas que interpretem a linguagem natural.

Os primeiros trabalhos de pesquisa visando ao uso da linguagem natural como forma de interação entre homens e máquinas datam da década de 1940 (JURAFSKY; MARTIN, 2009) e, desde então, muitas abordagens e técnicas diferentes foram aplicadas e analisadas. A literatura descreve trabalhos que usam abordagens determinísticas, estatísticas ou heurísticas. Independentemente dos métodos usados, o processamento da linguagem natural está associado às operações de análise léxico-morfológica, sintática, semântica e pragmática (RICH; KNIGHT, 1990). A análise sintática, também conhecida como *parsing*, é um dos principais componentes de várias aplicações de processamento de linguagem natural, como tradutores, reconhecedores de voz e sistemas baseados em diálogo

(JURAFSKY; MARTIN, 2009). No entanto, analisar textos é uma tarefa complexa por causa das ambiguidades da linguagem, múltiplas interpretações de palavras, diferentes ordens possíveis de elementos e itens ausentes. Além disso, as línguas mais populares se beneficiam de um número significativo de recursos computacionais, enquanto as demais, entre elas o Português, não dispõem de tantas ferramentas dessa natureza (LIMA; COSTA-ABREU, 2020).

1.2 Objetivos

Esta tese tem como objetivo apresentar um método adaptativo para análise sintática do Português Brasileiro. O formalismo adaptativo foi escolhido como modelo teórico subjacente devido à sua riqueza de representação e de manipulação, o que o torna consistente e flexível ao mesmo tempo, proporcionando o embasamento necessário para a construção do modelo computacional proposto, sem a necessidade de recorrer a técnicas auxiliares. O modelo proposto foi validado através de experimentos nos quais os resultados foram comparados aos obtidos pelos analisadores sintáticos do estado da arte para o Português, visando avaliar a eficiência do modelo nos diversos cenários de testes. Os resultados foram apurados através do método PARSEVAL (BLACK et al., 1991), considerado padrão para apurar o grau de correção das árvores de derivações geradas por analisadores sintáticos. O presente trabalho é o resultado de um esforço de agregação de diversos conceitos e técnicas desenvolvidos por (IWAI, 2000), (MORAES, 2006), (PADOVANI; NETO, 2017) e (MIURA, 2019) na área de Linguística Computacional e Adaptatividade, com intuito de resolver problemas de Processamento de Linguagem Natural que emergem da pesquisa realizada nesta tese.

1.3 Estrutura do trabalho

O restante desta tese está organizado da seguinte forma: O capítulo 2 apresenta os conceitos utilizados no trabalho, descrevendo a análise sintática do ponto de vista linguístico teórico e computacional, além de gramáticas, algoritmos, técnicas de treinamento e testes, e os trabalhos desenvolvidos nesta área de pesquisa com tecnologia adaptativa. O capítulo 3 apresenta uma revisão bibliográfica dos trabalhos mais importantes relacionados à análise sintática, majoritariamente para a Língua

Inglesa, que representam o estado-da-arte neste tema, e uma pesquisa sistemática da literatura para Língua Portuguesa, na qual gramáticas computacionais e ferramentas de análise são amplamente analisadas, com extensão para trabalhos correlatos de processamento de linguagem natural, e uma analise comparativa ao final. O capítulo 4 apresenta a definição e a formalização do método proposto por esta tese. O capítulo 5 descreve as etapas preparatórias para a realização da análise sintática; o capítulo 6 apresenta o módulo de análise por construções estáticas; o capítulo 7 apresenta o módulo de análise por construções dinâmicas e o capítulo 8 o módulo de análise por combinação de construções estáticas e dinâmicas. O capítulo 9 apresenta um estudo da complexidade algorítmica dos algoritmos propostos e faz uma comparação com algoritmos clássicos. O capítulo 10 é reservado para apresentação dos experimentos realizados, assim como discussão dos resultados. O capítulo 11 apresenta as conclusões e direcionamentos para continuidade da pesquisa. Finalmente, as Referências Bibliográficas.

2 CONCEITOS

Este capítulo apresenta conceitos básicos de Linguística Teórica e Computacional, técnicas normalmente utilizadas em análise sintática, algoritmos clássicos e uma breve introdução do trabalho desenvolvido no Laboratório de Linguagens e Técnicas Adaptativas da Escola Politécnica da USP, no tema de processamento de linguagem natural. O objetivo é prover o leitor com a fundamentação teórica essencial para o entendimento das seções seguintes.

2.1 Introdução

A análise sintática é uma atividade multidisciplinar que envolve as áreas de Linguística, Ciência da Computação e Engenharia. Para que o tema seja mais bem entendido, é importante que conceitos básicos sejam apresentados, assim como técnicas normalmente usadas para resolver problemas do domínio sejam esclarecidas. Esta seção procura dar esta visão, apresentando conceitos de Linguística Teórica, Linguística Computacional, Análise Sintática e Análise Semântica, além de mostrar gramáticas e algoritmos clássicos que, em maior ou menor grau, são utilizados na resolução de problemas de *parsing*. Ao final, são apresentados trabalhos relacionados ao tema, desenvolvidos no Laboratório de Linguagens e Técnicas Adaptativas.

2.2 Linguística teórica

A linguística se consolidou e se desenvolveu como ciência ao longo do século XX. Nesse período, as pesquisas em análise sintática, que a princípio eram insignificantes, evoluíram e se tornaram uma referência efetiva para as demais. Analogamente à própria linguagem, a sintaxe das línguas naturais é um fenômeno extremamente complexo que permite o uso de diferentes abordagens e métodos para analisá-la, caracterizá-la e explicar seu funcionamento. Por um lado, a sintaxe faz parte da estrutura das linguagens, consistindo em formas e arranjos formais. Por outro lado, essas formas e arranjos implicam algo sobre o mundo e cumprem alguma função comunicativa dessa forma (OTHERO; KENEDY, 2015).

A Sintaxe Gerativa formulada por Noam Chomsky (LEES; CHOMSKY, 1957), é uma das abordagens mais influentes para a gramática das línguas humanas. Segundo

Chomsky, para todas as línguas é possível criar um número infinito de expressões linguísticas usando um número limitado de elementos constitutivos. Além disso, ele afirmou ser possível combinar itens lexicais, frases ou sentenças e ter uma sentença normal como resultado, desde que as regras de combinações sintáticas sejam respeitadas. Por exemplo, as regras do português determinam que os artigos sempre devam preceder os substantivos na formação dos sintagmas nominais. Se esta regra não for seguida, a estrutura é considerada incorreta, ou seja, não gramatical. Chomsky acrescenta que certas regras são particulares e dependentes de um idioma ou tipologia linguística, enquanto outras são universais e independentes de qualquer idioma específico. A Sintaxe Gerativa assume que a linguagem é o resultado de um processo computacional, onde o conjunto de computações que geram uma frase é chamado "derivação", e as frases e os sintagmas que resultam dessa computação são chamados "representação". Chomsky também definiu um conjunto de hierarquias para gramáticas (CHOMSKY, 1959). No conjunto mais restrito, existem gramáticas do tipo 3, também chamadas gramáticas regulares. A próxima é a gramática do tipo 2 ou livre de contexto. Acima destas, existem gramáticas do tipo 1 ou sensíveis ao contexto e, por fim, as gramáticas do tipo 0 ou irrestritas, cujas regras não estão sujeitas a quaisquer restrições (OTHERO; KENEDY, 2015).

Outra abordagem é apresentada pela Sintaxe Lexical, que parte da ideia de que a análise de seus elementos constitutivos, os itens lexicais, dará um melhor entendimento da formação das sentenças. A Sintaxe Lexical, também conhecida como gramática de restrição, é baseada em duas ideias centrais. A primeira ideia é que se uma estrutura obedece ou segue uma série de restrições impostas a todas as construções linguísticas bem formadas, então essa estrutura deve ser considerada gramatical. As teorias lexicais usam a noção de restrições impostas aos signos linguísticos em vez de usar derivações ou transformações como mecanismo de licenciamento para construções gramaticais. A segunda ideia é que a Sintaxe Lexical usa principalmente informações que se originam do léxico para licenciar construções sintáticas. Esse foco no léxico requer formalização detalhada de objetos lexicais para descrever os mecanismos que explicam como elementos como entradas lexicais, restrições linguísticas e regras morfológicas se combinam para

formar estruturas sintáticas. Existem diferentes formalismos para representar as propriedades lexicais e seus mecanismos de combinação. Entre eles, destacam-se: *Lexical-Functional Grammar* (BALTIN; BRESNAN, 1985)*, Generalized Phrase-Structure Grammar* (SOAMES et al., 1989)*, Tree Adjoining Grammar* (JOSHI, 1987) e *Head-Driven Phrase-Structure Grammar* (GUNJI; POLLARD; SAG, 1996)(OTHERO; KENEDY, 2015).

A Sintaxe Construcionista postula que as construções, ou pares de padrões linguísticos com significados, são blocos de construções fundamentais da linguagem humana. Qualquer padrão linguístico é considerado uma construção, desde que algum aspecto de sua forma ou de seu significado não possa ser previsto a partir de suas partes componentes, ou de outras construções que são reconhecidas como existentes. Neste tipo de sintaxe, cada enunciado é entendido como uma combinação de múltiplas construções diferentes, que juntas especificam seu significado e forma de maneira precisa. Ao invés de assumir uma divisão bem definida de léxico e sintaxe, a Sintaxe Construcionista considera todas as construções como parte de um contínuo léxico-sintático, em que palavras e construções sintáticas são pares de forma e significado e diferem apenas na complexidade simbólica interna (HOFFMANN; TROUSDALE, 2013). Entre os formalismos utilizados citam-se: *Embodied Construction Grammar* – ECG (BERGEN; CHANG, 2005), *Fluid Construction Grammar* FCG (STEELS, 2011, 2012) e, mais recentemente, *Template Construction Grammar* - TCG (BARRÈS; LEE, 2014) (MARQUES; BEULS, 2016b).

2.3 Linguística computacional

A linguística computacional é o campo de investigação cujo objetivo final seria o de dotar os computadores com competência linguística similar à dos seres humanos, construindo artefatos que permitam às máquinas realizar atividades complexas tais como tradução, perguntas e respostas, sumarização de textos, análise de sentimento e sistemas de diálogo. Os métodos usados em linguística computacional são derivados de outras áreas do conhecimento, como linguística teórica, ciências cognitivas e ciências computacionais. Os primeiros trabalhos em linguística computacional foram dedicados à tradução de textos e sistemas de perguntas e

respostas, geralmente limitados a domínios específicos. Na década de 1960, foram desenvolvidos os primeiros sistemas dedicados à compreensão da linguagem e agentes de diálogo. As técnicas e modelos teóricos variavam, mas podem ser resumidos em modelos que buscam emular um ser humano, combinando as entradas do usuário e recuperando as saídas com base em padrões de pré-construção ou representações de conhecimento. A grande dificuldade foi generalizar os resultados obtidos para além de domínios específicos. Uma grande mudança ocorreu a partir do final da década de 1980, com o uso de corpora, textos pré-anotados com informações linguísticas e técnicas estatísticas, permitindo o desenvolvimento de aplicações como reconhecedores de voz, *taggers* de classe gramatical, analisadores, tradutores e sistemas de perguntas e respostas. Embora ainda dominantes, abordagens baseadas em corpus e modelos estatísticos têm dado espaço para modelos híbridos, que integram esses conceitos a abordagens que usam lógica formal, para obter um melhor entendimento e comportamento mais inteligente em sistemas de diálogo e compreensão da linguagem (SCHUBERT, 2020).

A sintaxe computacional é a parte da linguística computacional que se preocupa com a representação estrutural da gramática, criando meios para os computadores lidarem com relações como ordem, dependência, concordância e regência de palavras. A Teoria Gerativa de Chomsky teve grande influência na formação dos primeiros modelos sintáticos computacionais, desenvolvidos para o propósito de tradução de texto. Porém, ela se mostrou insuficiente para lidar com fenômenos relacionados ao nível lexical, como ambiguidade semântica, categórica e configuracional. Gramáticas de Unificação (SHIEBER, 1986), com forte orientação lexical, foram propostas para preencher essa lacuna. Elas usavam pares de atributo-valor para descrever o domínio da informação e regras indutivas para representar concatenações e associações. Embora essas gramáticas fornecessem o formalismo necessário para a análise, elas acabaram sobrecarregando o trabalho lexicográfico e se tornaram inviáveis como uma alternativa robusta para o processamento automático de línguas humanas (OTHERO; KENEDY, 2015).

A maior disponibilidade de dados em meio eletrônico a baixo custo favoreceu o desenvolvimento de abordagens estatísticas que buscavam induzir o conhecimento

linguístico ao invés de descrevê-lo manualmente. Como resultado, vários corpora de árvore sintática, os chamados *treebanks*, foram produzidos, tendo como referência o *Penn Treebank* da Universidade da Pensilvânia (OTHERO; KENEDY, 2015). Um fragmento da *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) é apresentado na Figura 1.

```
( (S (NP-SBJ (NP Pierre Vinken)
             ,
             (ADJP (NP 61 years)
               old)
             ,)
     (VP will
         (VP join
             (NP the board)
             (PP-CLR as
             (NP a nonexecutive director))
         (NP-TMP Nov. 29)))
     .))
( (S (NP-SBJ Mr. Vinken)
     (VP is
         (NP-PRD (NP chairman)
         (PP of
             (NP (NP Elsevier N.V.)
             ,
             (NP the Dutch publishing group)))))
     .))
```

Figura 1 – Fragmento do *Penn Treebank* da Universidade da Pensilvânia

Fonte: Adaptado de *NLTK (BIRD; KLEIN; LOPER, 2016)*

2.4 Análise sintática

O objetivo básico da análise sintática ou *parsing* é estabelecer as relações sintáticas entre as palavras da linguagem natural (CLARK; FOX; LAPPIN, 2010). A análise sintática é uma tarefa de vital importância no Processamento de Linguagem Natural (PLN), por causa de seu papel na determinação da estrutura de expressões linguísticas de forma que se possam extrair os seus significados. Além disso, a análise sintática demonstrou trazer vantagens para outras tarefas do PLN, tais como extração de relações, etiquetação de papéis semânticos e detecção de paráfrases (GILDEA; PALMER, 2001; CALLISON-BURCH, 2008; SOCHER et al., 2013). As linguagens naturais podem ser analisadas por abordagens orientadas a gramática

ou dados. Cada uma tem suas limitações. Analisadores guiados apenas por regras gramaticais geralmente perdem robustez, pois não há regras gramaticais suficientes para que analisem todas as possíveis cadeias de entrada. Além disso, uma determinada cadeia de entrada pode não fazer parte da linguagem natural, como ocorre, por exemplo, quando uma palavra é escrita ou pronunciada incorretamente ou quando algum elemento essencial é omitido da frase. Alguns pesquisadores argumentam que o problema de robustez em abordagens baseadas em regras gramaticais pode ser resolvido com o relaxamento das restrições gramaticais impostas aos analisadores sintáticos. No entanto, o relaxamento das regras gramaticais pode fazer com que os analisadores produzam várias análises para uma determinada cadeia de entrada, consumindo muito tempo até concluírem a análise, resultando em ineficiência computacional. Além disso, ter múltiplas análises candidatas para uma determinada cadeia de entrada pode aumentar o risco de o analisador selecionar uma alternativa incorreta como resultado final, reduzindo assim a precisão (JAF; RAMSAY, 2016).

Por outro lado, os analisadores sintáticos baseados em dados usam um mecanismo indutivo para gerar análises de cadeias de entrada. A maioria dos analisadores baseados em dados é capaz de atribuir pelo menos uma análise à cadeia de entrada e, portanto, são muito robustos. No entanto, a extrema robustez dos analisadores baseados em dados pode gerar análises gramaticalmente incorretas, prejudicando a precisão em comparação com os analisadores baseados em gramática. Além disso, a desambiguação pode representar um grande problema para analisadores baseados em dados, pois a alta robustez é alcançada através do relaxamento extremo das regras gramaticais. O problema de desambiguação pode ser parcialmente resolvido com algoritmos de aprendizado de máquina que criam um mecanismo de pontuação, associando uma pontuação a cada análise e, em seguida, aplicando um critério de otimização. Quanto ao problema de eficiência, argumenta-se que as abordagens baseadas em dados são superiores às abordagens baseadas em gramática; no entanto, isso geralmente prejudica a sua precisão (KAPLAN et al., 2004; SAMUELSSON, 2007; JAF; RAMSAY, 2016).

2.4.1. Análise sintática por constituintes

A análise sintática por constituintes é a tarefa de reconhecer e atribuir uma estrutura constituinte a uma frase. Uma estrutura constituinte é composta por constituintes, grupo de palavras que se comportam como uma única unidade, e suas relações. Estruturas constituintes são comumente modeladas por gramáticas livres de contexto, que compreendem um conjunto de regras ou produções expressas sobre um conjunto de símbolos não terminais e um conjunto de símbolos terminais (JURAFSKY; MARTIN, 2009). A principal característica da análise de constituintes é a visualização confortável da estrutura de uma frase em termos da relação de constituintes. Um exemplo de árvore de constituintes é apresentado na Figura 2.

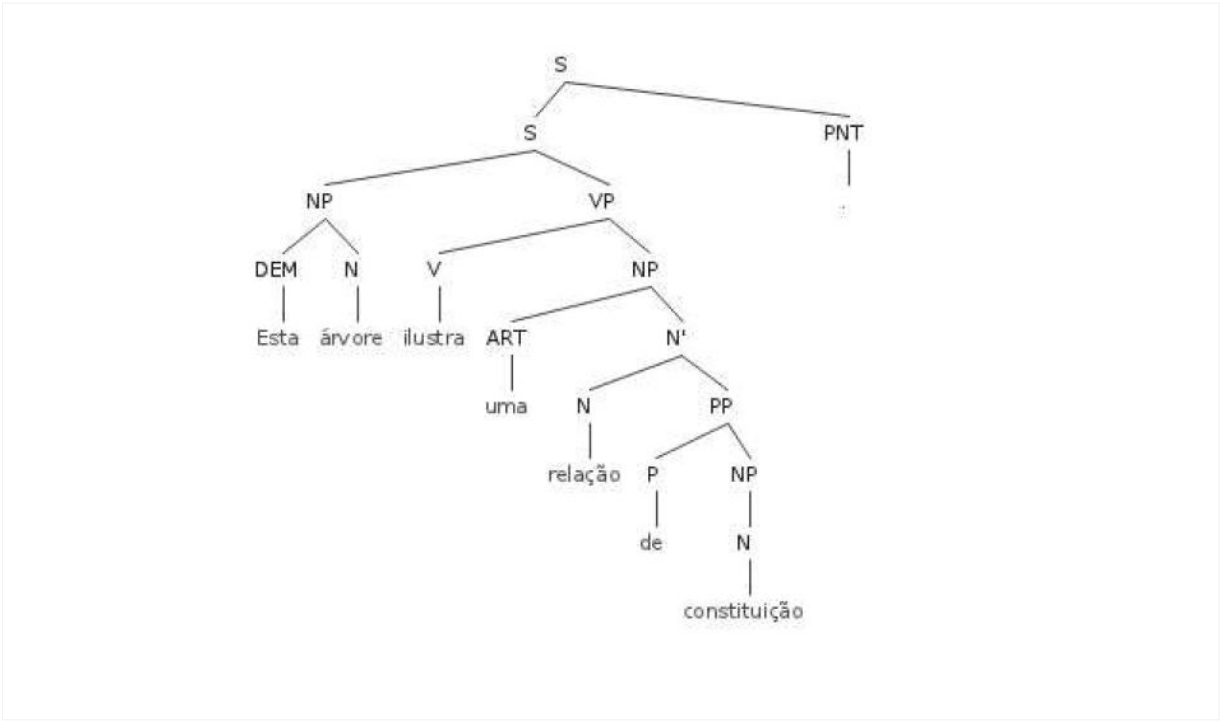


Figura 2 – Relação de constituição.
Fonte: Adaptado de (LXCENTER, 2021)

No exemplo apresentado na Figura 2, as palavras “Esta” e “árvore” são classificadas morfologicamente como pronome demonstrativo e substantivo (DEM e N). Estes dois elementos estão ligados por arcos ao sintagma nominal NP, indicando que são constituintes deste sintagma. Os símbolos não terminais diretamente ligados aos

símbolos terminais são também chamados de símbolos pré-terminais. No caso, DEM e N são símbolos pré-terminais de “Esta” e “árvore”, respectivamente.

Embora seja possível gerar sentenças sintaticamente válidas, usando regras de produção de maneira neutra, nem todas as construções sintaticamente corretas possuem significado semântico. Chomsky usou a frase “Ideias verdes incolores dormem furiosamente” como um exemplo para mostrar que as regras de produção são capazes de gerar frases sintaticamente corretas, mas semanticamente sem sentido (OTHERO; KENEDY, 2015). Outro aspecto importante das regras de produção é que elas podem ser utilizadas para analisar sentenças de cima para baixo, um processo conhecido como *top-down*, ou, alternativamente, de baixo para cima, denominado *bottom-up*. No processo de cima para baixo, a frase é dividida em constituintes, conforme a lista de regras é seguida. Na abordagem ascendente, a frase é gerada de baixo para cima por um processo de junção que agrega pequenos constituintes para criar constituintes maiores até atingir o maior, ou seja, a frase.

2.4.2. Análise sintática por dependência

Existem teorias gramaticais que não se baseiam em termos constituintes, mas, sim, na relação entre as palavras das sentenças. Tais teorias são conhecidas como gramáticas de dependência (JURAFSKY; MARTIN, 2009). Análises sintáticas feitas com gramáticas de dependência geram árvores de dependência. Um exemplo de árvore de dependência gerada por esta gramática é apresentado na Figura 3.

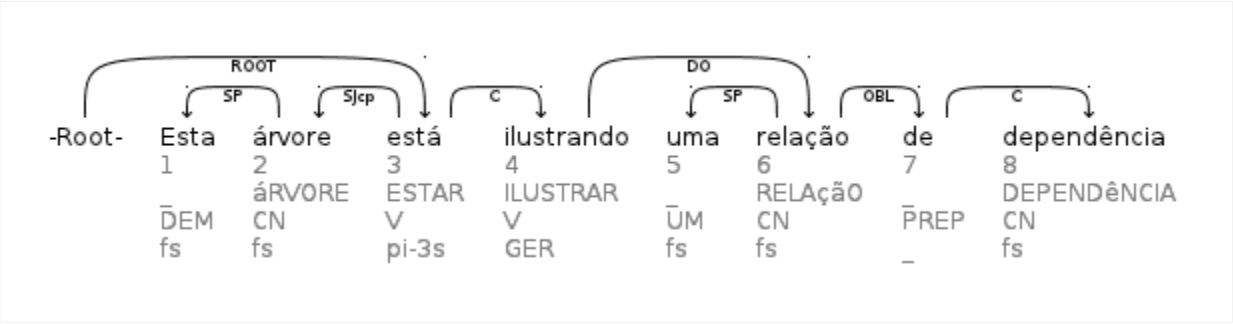


Figura 3 – Relação de dependência.

Fonte: Adaptado de (LXCENTER, 2021).

A estrutura básica de uma relação de dependência é composta por um elemento principal denominado *Head* e um elemento que se associa a ele, denominado dependente. Os arcos das árvores de dependência indicam os elementos dependentes, enquanto as etiquetas dos arcos indicam o tipo de dependência. Relações de dependência podem ser classificadas de acordo com o papel do termo dependente em relação ao principal. Sujeito, objeto direto, objeto indireto, e complemento nominal são exemplos destas classificações. Por exemplo, na Figura 3, o verbo “está” é o elemento *head* da sentença, o que é indicado pela etiqueta *root*. Ele está ligado ao verbo “ilustrando”, que tem o papel de complemento, indicado pela etiqueta c, e ao substantivo “árvore” que tem o papel de sujeito, indicado pela etiqueta Sjcp. A Figura 3 apresenta, ainda, informações adicionais relacionadas ao léxico, que não fazem parte diretamente da análise de dependências, mas que auxiliam a identificá-las. Por exemplo, a Figura 3 mostra o modo infinitivo da palavra “ilustrando” (“ilustrar”) e indica que “ilustrando” é um verbo que está na forma nominal gerúndio. O projeto *Universal Dependencies* (NIVRE et al., 2017) fornece um inventário amplo de classificações de relações para uso computacional (JURAFSKY; MARTIN, 2009).

2.5 Análise semântica

Analisadores semânticos mapeiam frases em representações de significado formal (MOONEY, 2007). Eles geralmente são usados para a resolução de problemas em domínios específicos, tais como sistemas de perguntas e respostas, onde as perguntas escritas em linguagem natural são convertidas em instruções de consultas a bancos de dados através de algoritmos de aprendizado supervisionados ou não supervisionados (MARQUES; BEULS, 2016a).

Existem diversas formas de representar o significado de um texto. As mais conhecidas são Lógica de Primeira Ordem, *Abstract Meaning Representation* (AMR) e *Frame-Based* ou *Slot-Filler Representation*. A Figura 4 apresenta exemplos de representações semânticas usando Lógica de Primeira Ordem e AMR. Independentemente da notação utilizada, representações de significado formal consistem de um conjunto de símbolos ou vocabulário representacional que, quando apropriadamente arranjados, correspondem a objetos, propriedades de objetos e

relacionamentos entre objetos de um determinado estado de coisas que se deseja analisar (JURAFSKY; MARTIN, 2009).

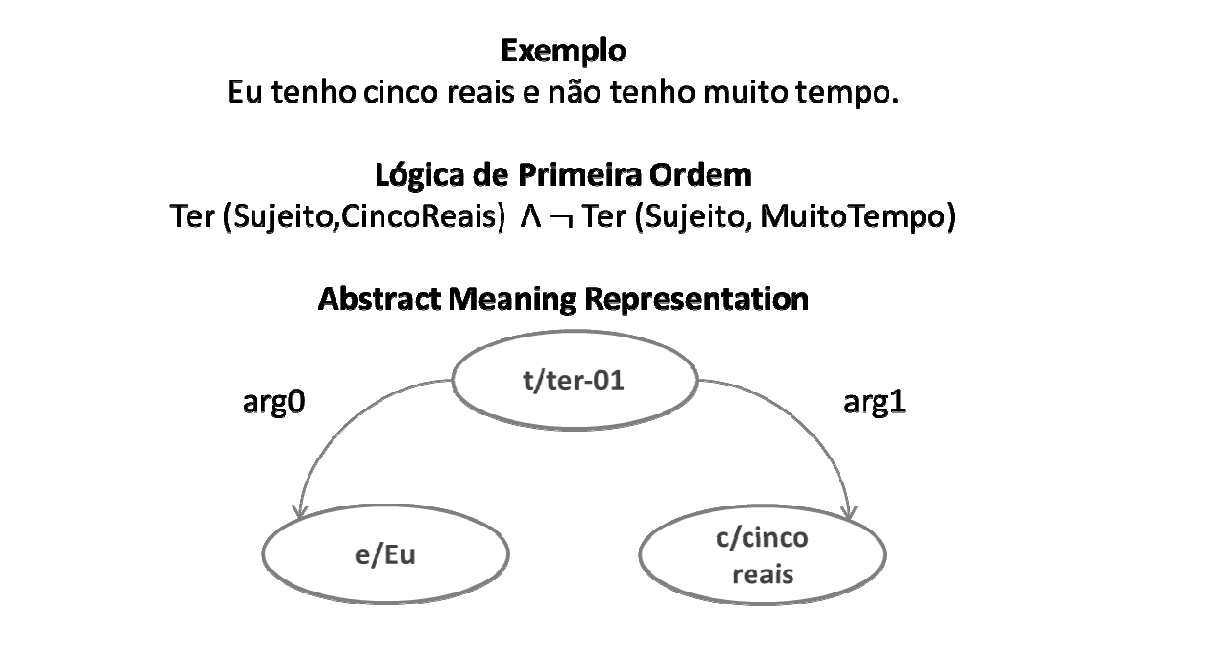


Figura 4 – Representações Semânticas.
Fonte: Adaptado de (JURAFSKY; MARTIN, 2009)

2.6 Gramáticas

Esta seção apresenta conceitos básicos de Gramáticas Livres de Contexto e Gramáticas Livres de Contexto Probabilísticas.

2.6.1. Gramáticas livres de contexto

Uma Gramática Livre de Contexto pode ser definida como uma 4-tupla:

$$G = (N, T, R, S)$$

onde:

N é o conjunto de símbolos não terminais

T é o conjunto de símbolos terminais

R é o conjunto de regras de produção

S é o símbolo inicial

2.6.2. Gramáticas livres de contexto probabilísticas

A ideia chave das Gramáticas Livres de Contexto Probabilísticas é estender a definição de Gramática Livre de Contexto (GLC), associando uma distribuição de probabilidade sobre possíveis derivações. Uma Gramática Livre de Contexto Probabilística estende cada regra em R com uma probabilidade condicional p (COLLINS, 2011):

$$\propto \rightarrow \beta\ [\mathrm{p}]$$

O símbolo p apresenta a probabilidade de uma expansão do não terminal α em β, à esquerda. Se considerarmos todas as expansões possíveis de um não terminal, a soma de suas probabilidades deve ser 1.

$$\sum_{\propto \rightarrow \beta \in \mathrm{R}:\ \propto = \mathrm{X}} \mathrm{p}(\propto \rightarrow \beta\ ) = 1\ \forall\ \mathrm{X} \in \mathrm{N}$$

As GLCP fornecem meios para a desambiguação de regras de produção através da apuração das probabilidades relativas a uma sentença e suas árvores de derivação. A probabilidade de uma análise específica T para uma sentença S é definida como o produto das probabilidades de todas as regras r usadas para expandir cada nó n na árvore de análise:

$$\mathrm{P}(\mathrm{T}, \mathrm{S}) = \prod_{\mathrm{n} \in \mathrm{T}} \mathrm{p}\big(\mathrm{r}(\mathrm{n})\big)$$

Um algoritmo de desambiguação deve escolher a análise T dentre o conjunto de árvores de análise $\tau(S)$ que maximiza a probabilidade $\mathrm{p}(\mathrm{T})$, formalmente:

$$\mathrm{T}(\mathrm{S}) = \underset{\mathrm{T} \in \tau(\mathrm{S})}{\operatorname{argmax}\, \mathrm{p}(\mathrm{T})}$$

2.7 Algoritmos

Esta seção apresenta algoritmos clássicos usados para analisar sentenças por meio de Gramáticas Livres de Contexto e Gramáticas Livres de Contexto Probabilísticas, a saber: algoritmo CYK, algoritmo Earley e algoritmo Shift-Reduce.

2.7.1. Algoritmo CYK

O algoritmo de Cocke-Younger-Kasami (SAKAI, 1962), conhecido também pela sua sigla CYK ou CKY, é um algoritmo de análise de linguagens livres de contexto, batizado com o nome de seus inventores, John Cocke, Daniel Younger e Tadao Kasami. Emprega análise *bottom-up* e programação dinâmica. A versão padrão do CYK opera somente com gramáticas livres de contexto descritas na forma normal de Chomsky (CHOMSKY, 1959). Este algoritmo faz o reconhecimento das sentenças através de uma matriz na qual são listados todos os símbolos não terminais que geram cada subsequência da sentença analisada. Após avaliar as subsequências de tamanho 1, o algoritmo avalia as subsequências de tamanho 2, depois as de tamanho 3 e assim sucessivamente. Para subsequências de tamanho 2 ou maiores, o algoritmo considera cada possível partição da subsequência em duas partes e verifica se existe uma produção $A \rightarrow B\ C$ tal que B corresponda à primeira parte da sequência e C corresponda à segunda. Caso o algoritmo encontre essa correspondência, ele registra o símbolo A correspondente a uma subsequência inteira. A sentença é reconhecida pela gramática se, ao final do processo, a subsequência contendo a sentença inteira corresponder ao símbolo inicial S. A análise sintática é gerada como um subproduto da matriz de reconhecimento (YOUNGER, 1967).

2.7.2. Algoritmo EARLEY

O algoritmo EARLEY (EARLEY, 1970) é um tipo de *parser top-down* que também usa programação dinâmica para analisar gramáticas livres de contexto, porém, diferentemente do algoritmo CYK, as gramáticas não precisam estar na forma normal de Chomsky. O algoritmo EARLEY, assim como o CYK, batizado com o nome de seu inventor Jay Earley, é atraente porque pode analisar todas as linguagens livres de contexto. O algoritmo EARLEY opera construindo conjuntos de estados a partir da cadeia de entrada. Cada estado é uma tupla $(X \rightarrow \alpha \bullet \beta, j)$, composta por uma regra de produção $(X \rightarrow \alpha\ \beta)$; uma marcação no lado direito da regra de produção, indicando quanto dessa regra já foi considerada; um ponteiro para um conjunto anterior. A marcação da posição no lado direito da regra é denotada por um ponto (•), e j é um ponteiro para o conjunto S(j). O conjunto de

estados definido na posição de entrada k é chamado S(k). O algoritmo se inicia com a regra de nível mais alto em S(0). Os itens de S(k) e S(k+1) são criados a partir de três operações aplicadas aos itens de S(k):

Predição: Para cada item em S(k) da forma (X → α • Y β, j) (onde j é a posição original), inclua (Y → • γ, k) em S(k) para toda a produção da gramática com Y no lado esquerdo da regra (Y → γ).

Varredura: Se a é o próximo símbolo da cadeia, para todo item em S(k) da forma (X → α • a β, j), inclua (X → α a • β, j) em S(k+1).

Conclusão: Para todo item em S(k) da forma (Y → γ •, j), busque todos os itens em S(j) da forma (X → α • Y β, i) e inclua (X → α Y • β, i) em S(k).

O algoritmo aceita a cadeia se (X → γ •, 0) terminar em S(n), onde (X → γ) é a regra do nível mais alto e n é o comprimento da cadeia, caso contrário, o algoritmo a rejeita.

A versão original do algoritmo de EARLEY também incluía o uso de *look-ahead* (leitura do elemento subsequente ao elemento na posição de entrada k) para processar determinados tipos de gramáticas não ambíguas em tempo linear, o que acabou sendo excluído da maior parte das implementações devido ao pouco ganho prático (EARLEY, 1970; AYCOCK; HORSPOOL, 2002).

2.7.3. Algoritmo SHIFT-REDUCE

Algoritmos SHIFT-REDUCE são usados por uma classe de *parsers* que analisam as cadeias de entrada de forma determinística, em uma única varredura, que é feita de baixo para cima (*bottom-up parsing*), com apoio de uma tabela de decisões. São normalmente empregados em linguagens de programação e outras notações formalmente definidas por uma gramática. A cada etapa do processamento, a cadeia de entrada é dividida em um símbolo objeto de análise, um símbolo de *lookahead* e os símbolos que compõem a parte restante da cadeia, ainda a ser analisada. O algoritmo funciona através de uma combinação de operações de deslocamento (SHIFT) e de redução (REDUCE). Na operação de deslocamento, o algoritmo avança na leitura da cadeia de entrada, movimentando o símbolo lido para uma pilha de análise. Na operação de redução, o algoritmo seleciona uma regra gramatical,

determinada a partir de uma tabela que contém todas as combinações sintaticamente válidas dos símbolos da pilha e do símbolo de *lookahead,* substituindo os símbolos da pilha pelo não terminal à esquerda da regra gramatical selecionada. O algoritmo SHIFT-REDUCE é executado em tempo linear com relação ao tamanho da cadeia e essa eficiência está relacionada ao fato de ele trabalhar com gramáticas não ambíguas. Gramáticas ambíguas são problemáticas para esse tipo de analisador porque elas podem gerar conflitos entre as operações de deslocamento e redução, que ocorrem quando parte da cadeia é elegível à redução e também ao deslocamento. Além disso, elas podem gerar mais de uma regra de produção para a mesma sequência de símbolos analisados, fazendo com que o analisador não consiga determinar com qual delas ele deve realizar a redução (AHO; ULLMAN, 1972).

2.8 Tecnologia adaptativa

Esta seção apresenta as principais aplicações conhecidas de técnicas adaptativas em linguística computacional.

2.8.1. Autômatos adaptativos

Um autômato adaptativo é uma máquina de estados que modifica sucessivamente sua estrutura de acordo com a aplicação de ações adaptativas associadas às regras de transição realizadas pelo autômato (NETO, 1993). Desta forma, transições podem ser eliminadas ou incorporadas ao autômato como resultado de cada uma das transições executadas durante a análise de entrada. Os autômatos adaptativos são reconhecedores de linguagens sensíveis ao contexto. Constituem um modelo de máquina de estados finitos com características dinâmicas, que permitem que sua estrutura e comportamento sejam alterados sempre que necessário, em particular quando são identificadas ocorrências de novas dependências de contexto nas entradas analisadas. Podem operar como autômatos finitos para o reconhecimento de linguagens regulares; como autômatos de pilha estruturados, para aceitar linguagens livres de contexto; e como máquinas equivalentes à máquina de Turing, utilizando o recurso das ações adaptativas, para tratar linguagens sensíveis ao contexto.

2.8.2. Dispositivos adaptativos

O modelo dos autômatos adaptativos, inicialmente proposto como uma nova abordagem para o desenvolvimento de compiladores, foi expandido para abranger o conceito de dispositivos adaptativos (NETO, 2001). Segundo esta formulação, o dispositivo adaptativo é formado por um dispositivo convencional não adaptativo e por um conjunto de funções adaptativas responsáveis pela automodificação do mesmo. O dispositivo convencional pode ser uma gramática, um autômato ou qualquer outro dispositivo que respeite um conjunto finito de regras estáticas. Este dispositivo é definido por meio de uma coleção de regras, geralmente na forma de cláusulas *if-then*, que testam sua situação corrente em uma configuração específica e levam o dispositivo para sua próxima situação. Se nenhuma regra for aplicável, uma condição de erro será relatada e a operação do dispositivo será descontinuada. Se houver uma regra única aplicável à situação corrente, a próxima situação do dispositivo será determinada pela regra em questão. Se mais de uma regra aderir à situação corrente do dispositivo, todas as novas possíveis situações serão tratadas em paralelo e o dispositivo exibirá uma operação não determinística.

2.8.3. Gramáticas livres de contexto adaptativas

Gramática adaptativa é um formalismo proposto por Iwai (2000) no qual regras gramaticais são criadas dinamicamente a partir do processamento da cadeia de entrada e de informações adicionais sobre o contexto em que são encontradas. A autora define uma gramática adaptativa G como sendo uma tripla ordenada:

$$\mathrm{G} = (\mathrm{G}^0, \mathrm{T}, \mathrm{R}^0)$$

onde:

G^0 é uma gramática inicial, definida da seguinte forma:

$$\mathrm{G}^0 = (\mathrm{VN}^0, \mathrm{VT}, \mathrm{VC}, \mathrm{PL}^0, \mathrm{PD}^0, \mathrm{S})$$

onde:

VN^0 é um conjunto finito não vazio de símbolos não terminais

VT é um conjunto finito não vazio de símbolos terminais, sendo $\mathrm{VN}^0 \cap \mathrm{VT} = \varnothing$

VC é um conjunto finito de símbolos de contexto

PL^0 é o conjunto de regras de produção aplicáveis em situações livres de contexto

PD^0 é o conjunto de regras de produção aplicáveis em situações dependentes de contexto

$S \in VN^0$ é o símbolo inicial da gramática

As regras de produção são expressões dos seguintes tipos:

Tipo 1: Regras pertencentes ao conjunto PL^i, onde i é um número inteiro maior ou igual a zero, representando o número de alterações adaptativas já ocorridas com relação à gramática inicial:

$$N \rightarrow \{ A \} \alpha$$

onde:

$\alpha \in (VT \cup VN)^*$

$N \in VN$

$\{ A \}$ refere-se a uma ação adaptativa opcional associada à regra de produção

Tipo 2: Regras pertencentes ao conjunto PL^i, onde i é um número inteiro maior ou igual a zero, representando o número de alterações adaptativas já ocorridas com relação à gramática inicial:

$$N \rightarrow \varnothing$$

onde:

$\varnothing$ é um meta símbolo indicando conjunto vazio.

Esta produção indica que, embora o símbolo não terminal N esteja definido, deriva um conjunto vazio, ou seja, não há substituição prevista para o não terminal. Isto significa que, se esta regra for aplicada em alguma derivação, a gramática não gerará qualquer sentença. Esta regra é utilizada para o caso em que na gramática existam regras que referenciem não terminais que deverão ser dinamicamente definidos, como resultado da aplicação de alguma ação adaptativa.

Tipo 3: Regras pertencentes ao conjunto PD^i, onde i é um número inteiro maior ou igual a zero, representando o número de alterações adaptativas ocorridas com relação à gramática inicial:

$$\alpha N \leftarrow \{ A \} \beta M$$

onde:

$\alpha \in VC \cup \{\varepsilon\}$ e $\beta \in VC$

N e $M \in VN^i$

$\{ A \}$ é uma ação adaptativa opcional associada à regra de produção

A seta ao contrário indica que β está sendo injetada na cadeia de entrada, inserindo informação de contexto por meio da troca de αN por βM.

As regras do Tipo 3 também podem apresentar a seguinte definição:

$$\alpha N \rightarrow \{ A \} \beta M$$

onde:

$\alpha \in VC$, $\beta \in VT \cup \{\varepsilon\}$

N e $M \in VN^i$

$\{ A \}$ é uma ação adaptativa opcional associada à regra de produção.

Neste caso, a seta está na direção usual, mas apresenta, no seu lado esquerdo, um símbolo de contexto seguido de um símbolo não terminal, indicando que αN está sendo substituído por βM e incluindo o símbolo de contexto β na cadeia de saída.

T é um conjunto finito, possivelmente vazio, de funções adaptativas.

R^0 é uma relação do tipo (r, A), onde $r \in (PL^0 \cup PD^0)$ e $A \in T$, $R^0 \subseteq P^0 \times (T \cup \{\varepsilon\})$. Para cada regra de produção r há uma relação R^0 que a associa a uma ação adaptativa A.

2.8.4. Dependência de contexto

Moraes (2006) descreve mecanismos para incorporar informações de contexto no processamento de linguagem natural usando tecnologia adaptativa. A autora cita, como exemplo, a alteração dinâmica dos atributos associados aos símbolos não

terminais da gramática. Para executar essa tarefa, um símbolo de contexto, representando um determinado atributo, pode figurar isoladamente na cadeia de entrada e ser memorizado como argumento de uma ação adaptativa, permitindo que essa informação seja recuperada e utilizada. Como exemplo, considerando o trecho de gramática que descreve simplificadamente o sintagma nominal NP, cujo núcleo é um substantivo, temos:

$\mathrm{NP}_0 \rightarrow \{\ \mathrm{C(NP\_Vazio)}\ \}\ \mathrm{Y1}\ \mathrm{NP}_1$

$\sigma\_\mathrm{sa}\ \mathrm{NP}_1 \rightarrow \mathrm{NP}_0$

$\alpha\_\mathrm{subst\_pm}\ \mathrm{NP}_1 \rightarrow \{\ \mathrm{C(NP\_PM)}\ \}\ \mathrm{NP}_2$

$\alpha\_\mathrm{subst\_pf}\ \mathrm{NP}_2 \rightarrow \{\ \mathrm{C(NP\_PF)}\ \}\ \mathrm{NP}_2$

$\alpha\_\mathrm{subst\_sm}\ \mathrm{NP}_1 \rightarrow \{\ \mathrm{C(NP\_SM)}\ \}\ \mathrm{NP}_2$

$\alpha\_\mathrm{subst\_sf}\ \mathrm{NP}_1 \rightarrow \{\ \mathrm{C(NP\_SF)}\ \}\ \mathrm{NP}_2$

$\alpha\_\mathrm{x}\ \mathrm{NP}_1 \leftarrow \{\ \mathrm{C(NP\_Vazio)}\ \}\ \alpha\_\mathrm{x}\ \mathrm{NP}_2$

$\mathrm{NP}_2 \rightarrow \mathrm{Y}_2\ \mathrm{NP}_3$

$\sigma\_\mathrm{sa}\ \mathrm{NP}_3 \rightarrow \mathrm{NP}_2$

onde:

Y_1, Y_2: transdutores capazes de extrair da sentença os símbolos terminais da gramática e de substituí-los por símbolos de contexto;

σ_sa: símbolo de contexto indicando sintagma adjetivo que antecede um substantivo;

$\alpha\_\mathrm{subst\_pm}$: símbolo de contexto indicando substantivo plural masculino;

$\alpha\_\mathrm{subst\_pf}$: símbolo de contexto indicando substantivo plural feminino;

α_x: símbolo de contexto indicando que o símbolo na cadeia de entrada não é um substantivo;

NP_0, NP_1, NP_2, NP_3: símbolos não terminais que indicam sintagmas nominais;

$\mathrm{NP\_PM}$: sintagma nominal plural, masculino;

$\mathrm{NP\_PF}$: sintagma nominal plural, feminino;

NP_SM: sintagma nominal singular, masculino;

NP_SF: sintagma nominal singular, feminino;

NP_Vazio: sintagma nominal vazio;

50_Tipo_Atual: símbolo não terminal indicando o tipo da categoria gramatical que está sendo derivada.

C: função adaptativa que caracteriza a categoria gramatical com atributos das categorias lexicais. É definida da seguinte forma:

$$C(\text{Tipo}) = \{\text{X: - } [50\_\text{Tipo\_Atual} \rightarrow \text{X}] + [50\_\text{Tipo\_Atual} \rightarrow \text{Tipo}] \}$$

A ação adaptativa C é executada para alterar a regra que define o tipo da categoria gramatical NP. A definição de C supõe a utilização de um não terminal 50_Tipo_Atual para identificar o estado que indica o tipo da construção sintática que está sendo derivada, no caso, o sintagma nominal NP . Os atributos dos símbolos pertencentes às categorias lexicais são, então, transferidos para a categoria gramatical da qual ele é o núcleo, detectando, também, situações em que ocorre a categoria vazia. O nome 50_Tipo_Atual é derivado de um exemplo de declarações de identificadores com tipos associados apresentado por Neto (1993).

2.8.5. Analisadores dependentes de contexto

Miura (2019) apresenta um método para gerar analisadores sintáticos dependentes de contexto. O método proposto pelo autor decodifica as informações organizadas e armazenadas de uma sentença, gerando um grafo que simultaneamente descreve sua estrutura sintática, dada por uma gramática, e as relações de dependência entre seus elementos constituintes. A decodificação é realizada por um analisador gerado automaticamente a partir da especificação de suas regras, permitindo a sua alteração de forma incremental. A análise sintática de uma sentença é obtida após a execução das seguintes etapas de processamento:

a) Manipulação de gramática – Nesta etapa, o analisador converte a descrição textual de uma gramática livre de contexto na notação Wirth modificada para outra representação na forma de uma estrutura de dados

que acrescenta informações adicionais de rótulos para a geração de um transdutor sintático (NETO, 1993);

b) Geração do transdutor sintático – Nesta etapa, o analisador recebe como entrada a sequência de *tokens* originada na etapa anterior e gera a descrição de um autômato de pilha estruturado correspondente à linguagem livre de contexto descrita pela gramática em Wirth;

c) Análise sintática – Nesta etapa o analisador realiza a análise sintática livre de contexto do texto em linguagem natural, usando a gramática processada nas etapas anteriores;

d) Levantamento de padrões de relações de dependências – Nesta etapa, são identificadas as relações de dependência entre os elementos da árvore sintática estrutural gerada na etapa anterior;

e) *Parser* – Nesta etapa, o analisador sintático gera as árvores de dependência, usando as informações obtidas da etapa anterior.

2.9 Avaliação

Esta seção apresenta as principais técnicas usadas para avaliação de sistemas para processamento de linguagem natural, incluindo analisadores sintáticos.

2.9.1. Introdução

Os sistemas de PLN não são entidades monolíticas, mas, sim, conjuntos de componentes organizados em um pipeline de processamento. Os analisadores sintáticos dependem das etiquetas de marcação adotadas, as quais, por sua vez, dependem da *tokenização*. É possível avaliar cada componente individualmente ou considerar vários componentes de uma só vez. As avaliações de ponta a ponta permitem mensurar, de forma mais significativa, a eficácia do sistema como um todo em situações reais, enquanto as avaliações individuais permitem medir o desempenho de cada componente e avaliar a influência dos erros apresentados em um componente no desempenho dos demais (CLARK; FOX; LAPPIN, 2010).

2.9.2. Treinamento e Testes

Tipicamente, o desenvolvimento e a avaliação de um sistema de PLN envolvem a separação dos dados disponíveis nos seguintes subconjuntos:

a) Dados de treinamento: Esse termo refere-se a conjuntos de dados nos quais os itens de entrada são emparelhados com as saídas desejadas, geralmente como resultado de anotação manual, de forma que o sistema possa ser treinado, antes de sua avaliação ou uso;

b) Dados de desenvolvimento: Normalmente, um ou mais subconjuntos de dados são mantidos para avaliação, à medida que o sistema vai sendo desenvolvido;

c) Dados de teste: Este termo descreve os dados que serão usados para avaliar o desempenho do sistema, após o desenvolvimento.

A disjunção dos subconjuntos é fundamental para que o sistema que está sendo avaliado não receba informações contidas nos dados de treinamento. Em sua forma mais pura, isso significa que os dados de teste devem permanecer inteiramente intocados e isolados do pesquisador até que o sistema esteja finalizado e pronto

para avaliação. É importante observar que as avaliações podem ser excessivamente otimistas, mesmo quando os dados de teste são mantidos separados dos dados usados no desenvolvimento do sistema. É o caso de assumir que os sistemas serão avaliados com os mesmos tipos de dados que foram usados durante o desenvolvimento do sistema, por exemplo, mesmo gênero e tópico. Caso essa suposição não seja cumprida, o desempenho do sistema será impactado (CLARK; FOX; LAPPIN, 2010; ESCUDERO; MARQUEZ; RIGAU, 2000; GILDEA, 2001).

De maneira geral, reserva-se o máximo de dados possível para treinamento e desenvolvimento, deixando o restante para testes. Por exemplo, 70% dos dados para treinamento, 20% para desenvolvimento e 10% para testes. No entanto, a alocação de 70 por cento dos dados para treinamento pode não representar um conjunto suficiente para métodos de aprendizado automático, se apenas uma pequena quantidade de dados anotados estiver disponível.

A solução mais comum para este tipo de problema é a validação cruzada *k-fold*. Esta técnica consiste em usar todo o conjunto de dados disponíveis, dividindo-os em k subconjuntos, de forma que o treinamento seja feito em *k*-1 subconjuntos e o teste feito no subconjunto remanescente. O processo é repetido alternando os *k*-1 subconjuntos de treino e o subconjunto de testes, de forma a garantir que todos os subconjuntos participem tanto do treino como do teste, mas nunca ao mesmo tempo. Os resultados finais são apresentados na forma de média, desvio-padrão, intervalo de confiança e teste de significância, padronizando a comparação de algoritmos e sistemas alternativos (CLARK; FOX; LAPPIN, 2010).

Uma variante da validação cruzada *k-fold*, chamada *repeated random subsampling* ou validação cruzada de Monte Carlo, consiste em dividir aleatoriamente os dados em um conjunto de treino e outro de teste, repetindo o processo k vezes e gerando os resultados na forma de média, desvio-padrão, intervalo de confiança e teste de significância, assim como na validação cruzada *k-fold*. A vantagem deste método é que é possível escolher independentemente o tamanho do conjunto de treino e de teste e o número de vezes em que os testes serão realizados. A desvantagem é que algumas observações podem nunca ser selecionadas no conjunto de teste,

enquanto outras podem ser selecionadas mais de uma vez (KUHN; JOHNSON, 2013).

O método *holdout*, no qual os dados são divididos entre treino e teste uma única vez, embora comumente utilizado, não é considerado tão confiável quanto outros os métodos, visto que uma única divisão pode fazer com que determinadas sentenças apareçam apenas no arquivo de treino ou apenas no arquivo de teste, podendo gerar distorções nos resultados (CLARK; FOX; LAPPIN, 2010).

2.9.3. Métricas

A métrica mais usada para avaliação de analisadores sintáticos é o PARSEVAL (BLACK et al., 1991), que compara os termos constituintes gerados pelo analisador com termos constituintes de textos pré-anotados, revisados, considerados corretos e conhecidos como padrão ouro (*gold standard*). O PARSEVAL usa a estrutura de parênteses gerada pelo analisador sintático para identificar os termos constituintes e compará-los ao padrão ouro. A métrica original do PARSEVAL não considerava a etiqueta dos constituintes, que foi incluída posteriormente. A avaliação é feita através de medidas de precisão e cobertura e do indicador F-score, definidos da seguinte forma:

$$\mathrm{P} = \frac{\mathrm{CC}}{\mathrm{CS}}$$

onde:

P = precisão;

CC = Quantidade de termos constituintes etiquetados pelo analisador sintático em concordância com os termos constituintes etiquetados do padrão ouro

CS = Quantidade de termos constituintes etiquetados pelo analisador sintático

$$\mathrm{C} = \frac{\mathrm{CC}}{\mathrm{CO}}$$

onde:

C = cobertura

CC = Quantidade de termos constituintes etiquetados pelo analisador sintático em concordância com os termos constituintes etiquetados do padrão ouro

CO = Quantidade de termos constituintes etiquetados do padrão ouro

$$F = \frac{2 * P * C}{P + C}$$

onde:

F = F-score

P = Precisão

C = Cobertura

Além de precisão, cobertura e F-score, o PARSEVAL utiliza uma medida chamada *crossing brackets (cruzamento de parênteses),* que é definida da seguinte forma:

$$CB = \frac{cb}{cs} * 100$$

onde:

CB = % *crossing brackets*

cb = quantidade de constituintes que cruzam o padrão ouro

cs = quantidade de termos constituintes etiquetados gerados pelo analisador sintático

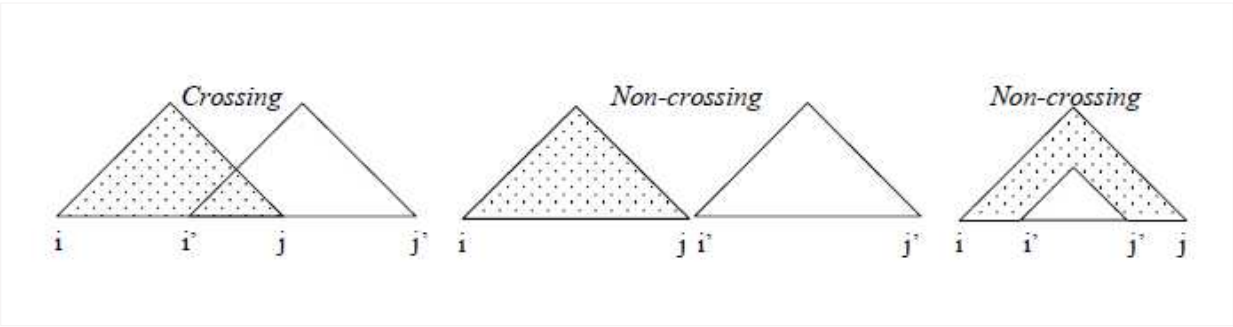


Figura 5 – Parênteses cruzando e não cruzando o padrão ouro.

Fonte: Adaptado de Kakkonen (2007)

A Figura 5 ilustra a ideia da medida *crossing brackets*. As fronteiras [i,j] representam os termos constituintes do padrão ouro, enquanto as fronteiras [i',j'] representam os termos constituintes gerados pelo analisador. Os pares [i,j] e [i',j'] são definidos como *crossing brackets* quando $i < i' \leq j < j'$. Quanto menor o índice *crossbrackets*, melhor é o resultado da análise. O exemplo abaixo apresenta as métricas do PARSEVAL etiquetado para a sentença "Hoje há novo concerto no Porto.", inicialmente preparada com o desdobramento da preposição "no" em "em_" "o".

Tabela 1. Exemplo da análise Parseval

Texto	Parser			Padrão Ouro		
	Início	Fim	Etiqueta	Início	Fim	Etiqueta
Hoje	1	1	ADV	1	1	ADV
há	2	2	V	2	2	V
Hoje há	1	2	V_	-	-	-
novo	3	3	A	3	3	A
Hoje há novo	1	3	VP	-	-	-
Hoje há novo	1	3	S	-	-	-
concerto	4	4	N	4	4	N
novo concerto	-	-	-	3	4	N_
novo concerto	-	-	-	3	4	NP
há novo concerto	-	-	-	2	4	VP
em_	5	5	P	5	5	P
o	6	6	ART	6	6	ART
Porto	7	7	N	7	7	N
o Porto	6	7	NP	6	7	NP
em_ o Porto	5	7	PP	5	7	PP
há novo concerto em_ o Porto	-	-	-	2	7	VP
concerto em_ o Porto	4	7	N_	-	-	-
concerto em_ o Porto	4	7	NP	-	-	-
concerto em_ o Porto	4	7	S	-	-	-
Hoje há novo concerto em_ o Porto	1	7	S	1	7	VP
.	8	8	PNT	8	8	PNT
Hoje há novo concerto em_ o Porto.	1	8	S	1	8	S

Fonte: Autor

A sentença 1 foi gerada pelo *parser*, enquanto a sentença 2 foi obtida do padrão ouro.

1) (S (S (S (VP (V_ (ADV Hoje) (V há)) (A novo))) (S (NP (N_ (N concerto) (PP (P em_) (NP (ART o) (N Porto))))))) (PNT .))
2) (S (VP (ADV Hoje) (VP (VP (V há) (NP (N_ (A novo) (N concerto)))) (PP (P em_) (NP (ART o) (N Porto))))) (PNT .))

A Tabela 1 apresenta os constituintes gerados pelo *parser* e os constituintes equivalentes do padrão ouro. Os constituintes corretos gerados pelo parser são aqueles nos quais as colunas início, fim e etiqueta são as mesmas do padrão ouro. *Crossing brackets* são os constituintes do *parser* com posição inicial ou final entre a posição inicial e final de algum constituinte do padrão ouro, no caso, as linhas correspondentes aos textos “Hoje há novo” e “há novo concerto”. As métricas deste exemplo são as seguintes:

Precisão = $\frac{\mathrm{CC}}{\mathrm{CS}}$ = 11/18 = 61,11%

Cobertura = $\frac{\mathrm{CC}}{\mathrm{CO}}$ = 11/16 = 68,75%

F-score = $\frac{2*\mathrm{P}*\mathrm{C}}{\mathrm{P}+\mathrm{C}}$ = 2* (61,11*68,75) / (61,11+68,75) = 64,71%

Crossing Brackets = $\frac{\mathrm{cb}}{\mathrm{cs}} * 100$ = 2/18*100 = 11,11%

3 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta inicialmente uma revisão bibliográfica dos trabalhos mais importantes relacionados à análise sintática, majoritariamente para a Língua Inglesa, que representam o estado-da-arte neste tema. Em seguida, apresenta-se um resumo de trinta anos da pesquisa para Língua Portuguesa, realizada a partir dos artigos disponíveis na Web of Science Core Collection (CLARIVATE, 2020), complementados pelos trabalhos de Julia, Seabra e Siqueira (1995), Bick (2000), Menezes e Neto (2000) e Padilha e Vicari (2000), que foram selecionados manualmente de Ladeira (2010) e Sardinha (2005). Por fim, é feita uma análise comparativa, procurando identificar lacunas entre o estado-da-arte e a pesquisa desenvolvida para a Lingua Portuguesa, visando identificar ideias, técnicas e estratégias que possam auxiliar no desenvolvimento de novos trabalhos para Língua Portuguesa.

3.1 Pesquisa geral

Esta seção apresenta uma revisão bibliográfica dos trabalhos mais importantes relacionados à análise sintática, que, de maneira geral, foram desenvolvidos para a Língua Inglesa.

3.1.1. Introdução

Steedman (1994) afirma que os analisadores sintáticos são constituídos por três componentes. O primeiro é a gramática, que especifica as regras que devem ser seguidas para combinar constituintes. O segundo componente é um algoritmo não determinístico que constrói a representação da estrutura da frase usando as regras gramaticais. Este algoritmo determina se as regras gramaticais são usadas de cima para baixo, de baixo para cima ou de uma maneira diferente. Também define em que ordem as palavras da frase são examinadas: da esquerda para a direita ou usando uma abordagem diferente. O terceiro componente é o oráculo, um mecanismo para resolver ambiguidades locais, decidindo qual regra gramatical deve ser escolhida quando o algoritmo encontra mais de uma opção.

A definição precisa apresentada por Steedman pode levar à conclusão de que as técnicas de *parsing* são limitadas ou, de alguma maneira, estejam totalmente

exploradas. No entanto, o que se observa na literatura é uma ampla gama de estratégias e abordagens que exploram ao máximo as diferentes formas de resolver o problema, algumas, inclusive, modificando ou mesmo desconstruindo os componentes descritos pelo autor. Para que o assunto seja tratado de maneira didática, esta seção está dividida em duas partes: a primeira apresenta as diferentes abordagens para a realização de análise sintática, percorrendo métodos determinísticos e probabilísticos, e relacionando desde pesquisas pioneiras até as técnicas de *word embedding* e *deep learning* que alteraram radicalmente a maneira como o *parsing* é realizado; a segunda parte está dedicada à pesquisa para o Português, na qual são apresentados 35 artigos selecionados da Web of Science (CLARIVATE, 2020), juntamente com artigos selecionados dos trabalhos de Ladeira (2010) e Sardinha (2005).

3.1.2. Métodos determinísticos

Marcus (1979) investiga a hipótese de que a linguagem natural pode ser analisada sintaticamente usando um mecanismo determinístico que lê a cadeia de entrada da esquerda para a direita sem suporte de paralelismo ou *backup*. Ele sugere um analisador gramatical que não processa sentenças que violem qualquer uma das duas restrições das regras gramaticais propostas por Chomsky: move-np, posição que o sintagma nominal pode ocupar na sentença; e move wh-phrase, mecanismo da sintaxe que permite expressar questões. O interpretador resolve ambiguidades estruturais locais com regras especialmente escritas para esse propósito, sem a necessidade de análise não determinística. Marcus também demonstra que a hipótese determinística necessita da interação entre semântica e sintática para testar a qualidade semântica de diferentes possibilidades estruturais para uma entrada.

Karlsson (1990) e Karlsson et al. (1996) propõem um formalismo chamado *Constraint Grammar* para analisar sentenças por meio de regras morfológicas e sintáticas. O formalismo foi desenvolvido a partir da análise extensiva de corpora, resultando em um conjunto de regras declarativas, utilizadas como restrições para descartar opções inválidas. O analisador segue um mecanismo no qual inicialmente é gerado o maior número possível de etiquetas morfossintáticas para, em seguida,

descartar etiquetas inválidas com base nas regras descritas pelas restrições declarativas. Analisadores desenvolvidos a partir das gramáticas de restrição atingem F-scores de aproximadamente 95% para rótulos de função sintática (GRAMMAR, 2020).

Brill (1993) introduz um formalismo chamado Gramática Transformacional, que analisa textos em árvores binárias sem identificação de não terminal. A gramática é aprendida por meio de um algoritmo de indução que, partindo de uma configuração inicial da estrutura da frase, altera sucessivamente a posição dos colchetes, comparando os resultados com um corpus de treinamento, com o objetivo de reduzir o total de erros. As regras para mudança de colchetes são descritas em modelos pré-definidos, que também levam em consideração a marcação de classes gramaticais. O autor testou o analisador no *Air Travel Information System* (ATIS) (HEMPHILL; GODFREY; DODDINGTON, 1990) e no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), usando a porcentagem de constituintes não cruzados como medida de precisão, alcançando 91,12% e 91,6%, respectivamente.

3.1.3. Analisadores parciais

Os analisadores tradicionais têm que lidar com muitas dificuldades para recuperar todas as informações da análise sintática: textos irrestritos geralmente contêm erros, a gramática e o léxico são incompletos e a tarefa de desambiguação geralmente é desafiadora. Os analisadores parciais visam recuperar de forma eficiente e confiável parte, mas não toda a informação da análise sintática. Embora mais simples, as análises produzidas por esta técnica podem ser usadas em tarefas como recuperação de informação, extração de informação e preparação de analisadores sintáticos mais complexos (ABNEY, 1997).

Voutilainen e Heikkila (1994) apresentam ENGCG, um analisador parcial que atribui função sintática e etiquetas de classe gramatical a cada palavra. Ambas são identificadas por meio da aplicação de regras de correspondência de padrões que eliminam etiquetas incorretas. Voutilainen (1995a) apresenta o NPTool, uma ferramenta que faz análises morfossintáticas para extrair sintagmas nominais de textos em inglês. Suas descrições morfológicas e sintáticas são baseadas em regras linguísticas codificadas manualmente e empregam um mecanismo para eliminar

etiquetas incorretas. Schwarz (1990) propõe Copsy, um analisador sintático de dependência que identifica e normaliza termos de várias palavras para recuperação de informação, usando regras baseadas em conhecimento empírico e algoritmo de análise orientado a padrões para identificar sintagmas nominais.

3.1.4. Métodos estatísticos

Gramáticas ponderadas e métodos de indução são recursos utilizados nas mais diferentes tarefas relacionadas à resolução de problemas de linguagem, como, por exemplo, aprendizagem, mudança, geração e compreensão. Gramáticas ponderadas são obtidas adicionando probabilidades às regras de produção de gramáticas convencionais (não probabilísticas). É possível imaginar problemas relacionados à linguagem como uma população de gramáticas que representam diferentes grupos de falantes ou mesmo um conjunto de hipóteses que os falantes da língua devem considerar para fazer a escolha certa. Outras propriedades da linguagem, como aprendizagem gradual de regras, mudança gradual de linguagem e continuum de dialeto, fazem mais sentido se usadas em conjunto com gramáticas ponderadas (ABNEY, 1997).

Gramática Probabilística Livre de Contexto (GPLC) é o formalismo gramatical ponderado mais comum para análise sintática de constituintes. Ela pode ser usada para determinar a árvore de derivação mais provável ou a probabilidade de uma árvore de derivação específica de uma frase. No entanto, a GPLC utiliza premissas que podem levar a resultados incorretos. O primeiro deles é a independência das regras gramaticais. De acordo com essa premissa, os nós não terminais se expandem independentemente uns dos outros. Mas uma análise detalhada da sintaxe mostra que a escolha de como um nó se expande pode depender de sua posição relativa na árvore de derivação. A outra premissa é a falta de sensibilidade das regras gramaticais às palavras, que só aparecem nas regras que envolvem pré-terminais, quando, na verdade, também são importantes em outras situações, como, por exemplo, para resolução de ambiguidades (JURAFSKY; MARTIN, 2009).

Collins (1997, 2003) apresenta um método estatístico generativo denominado Gramática Probabilística Livre de Contexto Lexicalizada (GPLCL), que inclui o léxico como parte integrante de símbolos não terminais. O autor descreve o método

por meio de três modelos. No primeiro, ele detalha a técnica de lexicalização da GPLC, destacando a possibilidade de utilização de parâmetros para indicar dependência entre pares de *head words* e uma métrica de distância para indicar preferência de fixação da subárvore. No segundo, ele estende o modelo para distinguir entre complementos e adjuntos, incluindo também subcategorização. No terceiro, Collins dá ao movimento wh um tratamento probabilístico. Os modelos foram avaliados no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), apresentando precisão e cobertura de 88,1 e 87,5%, respectivamente.

Charniak (2000) apresenta outro tipo de *parser* generativo e lexicalizado, que utiliza um modelo inspirado em máxima entropia, permitindo testar diferentes eventos condicionados logicamente. O analisador foi testado no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), atingindo F-score de 91,1% em um conjunto de teste de sentenças contendo até 40 palavras e F-score de 89,5% em um conjunto de teste de sentenças contendo até 100 palavras. Segundo o autor, estes resultados representam uma taxa de redução de erros de 13%, quando comparados aos obtidos por Collins (1997).

Klein e Manning (2003a) apresentam um método para melhorar o desempenho da GPLC não lexicalizada, usando anotações linguísticas aplicadas às etiquetas convencionais. Seu método subcategoriza as etiquetas não terminais com anotação parental, a fim de obter informações de contexto, e utiliza etiquetas com informação lexical das *head-words* para lidar com a dispersão de regras. Além disso, eles dividem as etiquetas de classes gramaticais com relação à *head-word*, visando capturar o comportamento específico das palavras. Os autores avaliaram sua proposta no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), obtendo F-score de 86,36%. Eles também observam que, além dos resultados objetivos, a GPLC não lexicalizada traz outros benefícios, como compactação, facilidade de interpretação e de análise.

Petrov et al. (2006) chamam a atenção para a falta de informação lexical na GPLC convencional e propõem um método alternativo à lexicalização. A proposta deles aprimora as etiquetas não terminais com subcategorizações, criadas a partir de um processo de aprendizado de divisão e aglutinação de uma gramática X-bar básica

(JACKENDOFF, 1977), trazendo mais granularidade aos não terminais, a fim de representar as condições geralmente usadas para desambiguação sintática. Os autores testaram sua abordagem no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), alcançando F-score de 90,2%.

Taskar et al. (2004) observam a eficiência de métodos discriminativos aplicados em categorização de texto e desambiguação do sentido das palavras, e propõem uma abordagem para usá-los também em análise sintática. Seu método emprega o conceito de programação dinâmica para representar de maneira compacta um grande número de análises em uma tabela. Essas análises são usadas para criar um conjunto de *features* que representam características das regras de produção, informações lexicais e informações de contexto, com o objetivo de treinar um algoritmo de classificação de margem máxima (SVM). Os autores testaram seu método em um subconjunto de sentenças de até 15 palavras do *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), atingindo F-score de 89,12%.

Finkel, Kleeman e Manning (2008) propõem um modelo discriminativo baseado em programação dinâmica e *conditional random fields* (CRF). Sua proposta permite a utilização de um amplo conjunto de *features*, criado a partir das regras gramaticais, índices indicando pontos de divisão de sentenças e características das palavras, dando flexibilidade para lidar com situações em que as regras de produção ou palavras são desconhecidas. Eles criaram melhorias, tais como pré-filtro de tabelas, paralelização e otimização estocástica, visando evitar que o modelo entrasse em complexidade exponencial, ao mesmo tempo garantindo a possibilidade de processar sentenças de qualquer comprimento. Eles testaram seu modelo no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), alcançando F-score de 90,9%, em um conjunto de sentenças de até 15 palavras, e F-score de 89,0% em um conjunto de sentenças de até 40 palavras.

3.1.5. Redes neurais

Elman (1991) propõe um modelo de rede neural para verificar se modelos conexionistas são capazes de representar estruturas compostas geradas por uma gramática. Ele conclui que as características de seu modelo – representação

distribuída da estrutura interna, sensibilidade ao contexto e ligação das palavras com a representação – pode indicar uma nova forma de representação mental.

Henderson (2003) introduz um analisador híbrido composto por dois componentes. O primeiro componente é uma rede neural treinada e testada no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), que é usada para estimar os parâmetros do segundo componente, um *Left-Corner Parser* que tem o papel de selecionar a análise mais provável. O modelo apresentado atingiu F-score de 88,8%.

Henderson (2004) apresenta três diferentes métodos para treinar uma rede neural usada para estimar as probabilidades de um *parser* estatístico. O primeiro método é generativo, o segundo é discriminativo e o terceiro é uma combinação na qual o modelo probabilístico é generativo, mas o critério de treinamento é discriminativo. Os modelos foram testados no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e o último superou os dois anteriores, alcançando F-score de 90,1%.

Titov e Henderson (2006) observam que o desempenho dos analisadores pode ser prejudicado quando um modelo treinado em um domínio é executado em outro, no qual a quantidade de dados é pequena. Para superar essa limitação, eles propõem uma rede neural treinada no domínio de origem ou nos domínios de origem e destino, combinada com um classificador *Support Vector Machine* (SVM), treinado apenas no domínio de destino. A rede neural é um analisador sintático que gera análises candidatas a partir de um conjunto de *features*. Os parâmetros desta rede são, então, usados para criar o *kernel* utilizado pelo classificador SVM para reclassificar as análises. Eles avaliaram seu método treinando-o apenas no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e no *Penn Treebank* complementado com partes do *Brown* corpus (FRANCIS; KUCERA, 1964). O modelo foi testado apenas no *Brown* corpus, atingindo F-score de 83,2%, quando treinado apenas no *Penn Treebank* e F-score de 87,0%, quando treinado com ambos os corpora.

Carreras, Collins e Koo (2008) chamam a atenção para o problema de alta complexidade computacional de modelos globais lineares (MGL), o que torna uma inferência exata virtualmente intratável. Os autores propõem um mecanismo de

análise para superar essa limitação, porém mantendo a flexibilidade das *features* dos modelos globais lineares. De acordo com a proposta, um algoritmo *perceptron* é combinado com programação dinâmica para recuperar constituintes. As *features* são modeladas de acordo com o formalismo da *Tree Adjoining Grammar* (TAG), podendo incorporar GPLC, bigramas, trigramas e *features* de superfície (características facilmente identificáveis nas palavras ou nas regras de produção). A complexidade computacional é gerenciada por um analisador de dependência de baixa ordem, que atua em uma etapa de preparação, reduzindo o espaço de busca do analisador MGL. Os autores testaram sua proposta no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), alcançando F-score de 91,1%.

3.1.6. Word embedding e deep learning

Word embeddings são vetores de distribuição que procuram capturar o contexto em que uma palavra é inserida a partir da hipótese distributiva, segundo a qual “palavras com significados semelhantes tendem a ocorrer em contextos semelhantes”. Os vetores de distribuição são obtidos otimizando um objetivo auxiliar em um grande corpus não rotulado, tal como prever uma palavra com base em sua vizinhança ou prever a vizinhança de uma palavra. A aplicação de *word embeddings* provou ser muito eficiente em tarefas centrais do PLN e geralmente é usada como a camada de processamento inicial em modelos de *deep learning* (YOUNG et al., 2018).

Deep learning é um termo relacionado a redes neurais que usam várias camadas para aprender a representação hierárquica de dados. Diversas tarefas de PLN, tais como análise sintática, tradução automática e sumarização, melhoraram após a introdução de técnicas de *deep learning*. Diferentes classes de redes neurais podem ser criadas com *deep learning,* cada uma delas apropriada para determinadas tarefas. Redes neurais convolucionais (RNC) permitem a extração de *features* de sentenças de entrada, sendo adequadas para representações semânticas. Redes neurais recorrentes (RNR) podem processar as informações sequencialmente, usando cálculos e resultados de etapas anteriores como parâmetros de entrada, e são ideais para modelagem de linguagem, tradução automática e reconhecimento de fala. Redes neurais recursivas (RNRv) são capazes de representar estruturas hierárquicas, sendo adequadas para análise de constituintes (YOUNG et al., 2018).

Costa et al. (2003) avaliam a aplicação do conceito de árvores incrementais na análise sintática, visando emular a forma como os seres humanos realizam essa tarefa. Eles propõem um modelo no qual as sentenças são analisadas da esquerda para a direita e cada *token* é vinculado a uma árvore incremental, de forma que os *tokens* já processados ofereçam o contexto usado na escolha das árvores incrementais vinculadas ao próximo *token*. Como pode haver mais de uma escolha, o modelo usa uma rede neural recursiva (RNRv), treinada com as estruturas sintáticas do *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), para obter a alternativa mais provável. Os autores fizeram experimentos em um conjunto de 2.000 sentenças e observaram que o modelo classificou consistentemente a resposta correta nas primeiras posições de uma lista de candidatas. Porém, não são apresentados indicadores quantitativos.

Menchetti et al. (2005) comparam o uso de redes neurais convolucionais (RNC) e redes neurais recursivas (RNRv) para mapear estruturas discretas em espaços vetoriais usados em algoritmos de aprendizagem. Eles avaliam esses métodos para resolver dois problemas aplicados ao PLN. O primeiro, denominado *first pass attachment,* consiste em resolver ambiguidades sintáticas locais para escolher a melhor árvore de análise incremental. O segundo, chamado *re-ranking,* consiste na escolha da melhor árvore sintática em um conjunto de árvores candidatas geradas por um analisador estocástico. Os autores testaram essas técnicas no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e concluíram que as redes recursivas (RNRv) superam as convolucionais (RNC) em ambos os casos.

Collobert (2011) propõe um algoritmo discriminativo para análise sintática de linguagem natural baseado em uma “rede convolucional de transformadores de grafos (*Graph Transformer Network*)”. O modelo usa a representação de palavras proposta por Collobert e Weston (2008) e estrutura o analisador sintático como mecanismo de etiquetação recursiva, assumindo que uma árvore sintática é constituída de uma pilha de níveis e que a inferência de um determinado nível deve considerar os níveis anteriores. O autor testou o modelo no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), atingindo F-score de 89,1% em sentenças de até 15 palavras, F-score de 89,7% em sentenças de até 40 palavras e F-score de 89,7% em sentenças de até 100 palavras.

Collobert et al. (2011) propõem uma arquitetura de rede neural unificada e um algoritmo de aprendizagem que pode ser aplicado a diferentes tipos de tarefas de PLN, tais como etiquetação de classe gramatical, reconhecimento de nomes de entidades e etiquetação de função semântica. A rede neural aprende representações internas com base em informações de grandes conjuntos de dados não rotulados, evitando o uso de engenharia específica para essas tarefas. Com relação à análise, seu sistema de rotulagem de função semântica não usa árvores de análise, mas, ao invés disso, tenta capturar informações sintáticas como um efeito colateral das transformações de operador de frase, um conceito baseado em *Operator Grammars* (HARRIS, 1968). Eles testaram o modelo no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e relataram F-score de 92,25%.

Socher et al. (2013) afirmam que os conjuntos de categorias usadas pela análise de linguagem natural são poucos e discretos, e que não são capazes de capturar a riqueza sintática e semântica de sintagmas linguísticos. As tentativas de melhorar a representação com lexicalização e subcategorização não resolveram o problema e trouxeram outros como aumento do número de dimensões e dispersão. Os autores introduzem um novo formalismo, chamado *Compositional Vector Grammar* (CVG), que combina GPLC com uma rede neural recursiva para aprender representações vetoriais usadas para inferir estruturas sintáticas. Eles testaram o modelo no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e relataram F-score de 90,4%.

Chen e Manning (2014) apresentam uma rede neural que classifica ações *shift-reduce* para um analisador de dependência baseado em transições. A rede neural trabalha com vetores de representações densas, em vez de indicadores discretos, a fim de reduzir o problema de incompletude dos dados, a dependência de recursos criados manualmente e os custos computacionais para processamento de *features*. Seu modelo extrai palavras, etiquetas de classes gramaticais e rótulos de dependência, computando o vetor de representação na camada oculta e selecionando aquele que apresenta a pontuação mais alta, que é então convertido na ação *shift-reduce* correspondente. Eles testaram seu analisador no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), alcançando UAS (*Unlabeled Attachment Score*) de 91,8% e LAS (*Labeled Attachment Score*) de

89,6%. Os autores também testaram o modelo no *Penn Chinese Treebank* (MARTAPA, 2005), alcançando UAS de 83,9% e LAS de 82,4%.

Dyer et al. (2015) apresentam uma nova maneira de representar e controlar o estado corrente de analisadores de dependência baseados em transição, usando pilhas LSTM (*Long Short-Term Memory*) para capturar a cadeia de entrada, o histórico completo de ações tomadas pelo analisador e o conteúdo completo de fragmentos parciais de árvore. A principal vantagem é que o *parser* tem total conhecimento de seu estado corrente para fazer as previsões, ao invés da visão limitada das abordagens convencionais. O modelo foi testado no *Stanford Dependency Treebank* (MARNEFFE; MACCARTNEY; MANNING, 2006), obtendo UAS de 93,1% e LAS de 90,9%, e no *Penn Chinese Treebank* (MARTAPA, 2005), obtendo UAS de 87,2% e LAS de 85,7%.

Zhou et al. (2015) apresentam um modelo de predição para analisadores de dependência baseados em transição que usa uma rede neural para pontuar a sequência completa de decisões tomadas pelo analisador, visando obter a máxima verossimilhança da frase (*Maximum-Likelihood Estimation*). A rede neural é otimizada através de *contrastive learning* (HINTON, 2002; CUN; HUANG, 2005; LIANG; JORDAN, 2008; VICKREY; LIN; KOLLER, 2010; LIU; SUN, 2015), técnica em que os dados observados têm sua probabilidade aumentada e os dados não observados têm sua probabilidade diminuída, combinada com *beam-search* (ZHANG; CLARK, 2011), método que amplia o espaço de busca. O modelo foi testado no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), atingindo UAS de 93,2% e LAS de 92,2%.

3.1.7. Abordagens mistas

Zhu et al. (2013) observam que as regras unárias aumentam o número de ações de *shift-reduce* para gerar todas as possíveis saídas para a mesma árvore de constituintes de entrada. A variação no número de ações pode afetar o desempenho dos métodos em que o treinamento trabalha junto com busca, como *global discriminative training* e *beam-search*. Os autores propõem uma forma de evitar esse problema, estendendo o framework *shift-reduce* padrão com uma ação adicional, chamada IDLE, para preencher as sequências de transição, reduzindo a

diferença de seus tamanhos. Os autores testaram seu método em *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e *Penn Chinese Treebank* (MARTAPA, 2005), atingindo F-score de 91,3% e 85,6%, respectivamente.

Weiss et al. (2015) propõem um modelo de rede neural para um analisador de dependência baseado em transição inspirado no classificador de Chen e Manning (2014). Seu modelo é estruturado em uma camada de entrada, uma de *embedding*, duas camadas ocultas interconectadas, uma camada *softmax* e um *perceptron*. Da camada de entrada, eles extraem *features* de palavras, etiquetas de classes gramaticais e rótulos de arco e os organizam em três matrizes esparsas diferentes, que são transformadas em representações vetoriais densas pela camada de *embedding*. As camadas ocultas, compostas por unidades lineares retificadas, processam esses vetores e interagem com a camada *softmax* para fornecer os vetores de probabilidade correspondentes. A camada *perceptron* recebe os vetores de probabilidade e os converte em ações de *shift-reduce*, usando *beam-search* para pesquisa estruturada. O modelo foi testado no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) e obteve 93,99% de UAS e 92,05% de LAS. Depois de incorporar dados não rotulados, eles melhoraram a precisão para 94,26% de UAS e 92,41% de LAS .

Vinyals et al. (2015) avaliam redes neurais *sequence-to-sequence* (SUTSKEVER; VINYALS; LE, 2014) e técnicas de *attentio*n (BAHDANAU; CHO; BENGIO, 2014) como abordagens alternativas para analisadores de constituintes, com o objetivo de resolver problemas de dependência de domínio, reduzir a complexidade e aumentar a eficiência computacional. Os autores linearizaram as árvores de constituição para que fossem tratadas como um problema de *sequence-to-sequence* e aplicaram o mecanismo de *attention* para analisar frases longas, realizando experimentos em conjuntos de dados anotados manual e automaticamente. O modelo foi treinado em um conjunto de dados anotado automaticamente por outros dois outros analisadores e testado na seção 23 do *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), obtendo um F-score de 92,1%.

Jaf e Ramsay (2016) fornecem uma visão geral da estrutura da Língua Árabe e apresentam um método de análise que combina abordagens baseadas em dados

com técnicas baseadas em gramáticas para obter melhores resultados. Sua proposta estende o algoritmo *shift-reduce* (AHO; ULLMAN, 1972) com um conjunto de perguntas e respostas e um conjunto de regras de dependência gramatical, ambos extraídos automaticamente de um *treebank*, visando restringir os resultados. O analisador foi testado no *Penn Arabic Treebank* (MAAMOURI; BIES, 2004) e os autores alegaram resultados promissores, considerando precisão e tempo para processamento dos textos.

Yang et al. (2017) notam que os diversos tipos de analisadores sintáticos geralmente produzem erros diferentes, mas complementares, sugerindo que a combinação de analisadores pode obter melhores resultados. No entanto, as técnicas de combinação também enfrentam a dificuldade de obter análises corretas, seja por causa da ausência de árvores de derivação corretas disponíveis ou pela falta de elementos constituintes corretos. Os autores então propõem um modelo de rede neural *sequence-to-sequence*, inspirado em Vinyals et al. (2015), que é treinado com uma lista de árvores de análise linearizadas como entrada e a análise correta como saída. Eles também aplicam LSTM (*Long-Short Term Memory*) e mecanismos de *attention* para selecionar informações de contexto de longa distância e filtrar as partes mais relevantes das árvores candidatas. O modelo foi testado no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), obtendo um F-score de 90,86%, superando os analisadores de *Berlekey* (PETROV et al., 2006), Zpar (ZHU et al., 2013) e *ReParsing* (SAGAE; LAVIE, 2006).

Chen et al. (2017) propõem aumentar a precisão de analisadores de constituintes do tipo *shift-reduce*, usando informações de fronteira para prever as ações a serem executadas pelo analisador. Os autores apresentam um primeiro modelo que usa informações de fronteira para verificar se uma palavra pertence a um constituinte válido, e um segundo modelo que leva em consideração as informações da árvore sintática para resolver problemas de desambiguação de nós não terminais. Os autores treinaram o modelo usando um extenso conjunto de dados rotulados por um analisador sintático já existente e testaram-no no *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), alcançando 90,8% F-score e no *Penn Chinese Treebank* (MARTAPA, 2005), alcançando 84,8% F-score.

Yuan, Jiang e Tu (2019) afirmam que analisadores de dependência convencionais podem propagar erros porque, como eles tomam suas decisões com base em dados coletados de sentenças de entrada, se ocorrer um erro, ele será propagado para as previsões subsequentes, afetando os resultados gerais, especialmente se o erro ocorrer nos estágios iniciais. Os autores propõem um modelo para superar esse problema, usando uma abordagem bidirecional, inspirada em tradução automática neural (LIU et al., 2016). O modelo consiste de um analisador do tipo *left-to-right*, treinado de forma convencional, um analisador do tipo *right-to-left*, treinado com palavras em ordem reversa e três algoritmos para decodificação: o primeiro baseado na função de pontuação conjunta padrão, o segundo baseado em *dual decomposition* (RUSH; COLLINS, 2012), e o terceiro baseado em *dynamic oracle* (GOLDBERG; NIVRE, 2012). Os autores testaram o modelo em diferentes *treebanks*. No *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), eles alcançaram UAS de 94,60% e LAS de 94,02%.

Jaf e Calder (2019) propõem um analisador do tipo *shift-reduce* que pode ser usado para diferentes línguas. A ideia principal é aproveitar os recursos disponíveis para alguns idiomas para analisar outros que não compartilham dos mesmos meios. A arquitetura do analisador consiste em uma camada de codificação, que converte palavras e etiquetas de classes gramaticais (POS) em *embeddings*, uma camada BiLSTM (*Bidirectional Long Short Term Memory*), que gera *feature embeddings* e uma camada de *perceptrons* (MLP), que infere as ações e etiquetas de análise a partir *feature embeddings*. Os autores tornaram o analisador multilíngue treinando-o primeiro com um corpus de um idioma linguisticamente semelhante e, em seguida, com um corpus do idioma desejado. O analisador foi testado no *Universal Dependencies Treebank* (NIVRE et al., 2017), melhorando o UAS em 8% e o LAS em 6,94%, quando pré-treinado em inglês para análise cazaque, e melhorando o UAS em 6,56% e o LAS em 20,68%, quando pré-treinado em Farsi para analisar *Kurmanji*.

3.1.8. BERT

Devlin et al. (2018) apresentam BERT (Bidirectional Encoder Representations from Transformers), uma variante bidirecional de redes *Transformer* (VASWANI et al.,

2017), treinada para prever uma palavra mascarada em seu contexto e classificar se duas frases são consecutivas ou não. Um modelo BERT pré-treinado pode ser ajustado com apenas uma camada de saída adicional para criar modelos para uma ampla uma série de tarefas de PLN, tais como perguntas e respostas e inferência de linguagem, sem modificações substanciais de arquitetura. Os autores apresentam resultados que estabelecem novo estado da arte para onze tarefas de PLN, incluindo 80,5% de acurácia no General *Language Understanding Evaluation* (GLUE), 93.2% de F-score no *Stanford Question Answering Dataset* (SQuAD v1.1) (RAJPURKAR et al., 2016), 83,1% de F-score no (SQuAD v2.0), e 83,7% de acurácia no corpus *Multi-Genre Natural Language Inference* (MultiNLI) (WILLIAMS; NANGIA; BOWMAN, 2018).

Jawahar, Sagot e Seddah (2019) investigam se o BERT pode ser usado para capturar informação sobre as estruturas das sentenças. Os autores realizam uma série de experimentos visando identificar informações de superfície, sintáticas e semânticas em um modelo BERT de 12 camadas. Informações de superfície são apuradas usando como métricas o comprimento e a presença de palavras na sentença; informações sintáticas são obtidas através dos indicadores de sensibilidade à ordem das palavras, profundidade da árvore sintática e sequência de constituintes na árvore sintática; e informações semânticas são apuradas através do tempo dos verbos, número do sujeito e do objeto direto, sensibilidade à substituição aleatória de substantivos por verbos e troca aleatória de orações oracionais coordenadas. Os resultados mostram que o BERT captura informações de superfície nas camadas inferiores (1 a 4), informações linguísticas nas camadas intermediárias (5 a 9) e características semânticas nas camadas superiores da rede neural (9 a 12). Os autores concluem, ainda, que o BERT obtém informação linguística de uma forma composicional que imita estruturas semelhantes a árvores.

He e Choi (2020) apresentam modelos para etiquetagem morfológica (PoS), análise sintática e análise semântica, usando BERT. Para cada tarefa, os autores primeiro replicam e simplificam a abordagem atual de última geração para, em seguida, avaliar abordagens usando *embeddings* gerados por BERT. Os autores modificam o modelo proposto por Devlin et al. (2018) separando a arquitetura BERT das camadas de saídas e usam *embeddings* BERT pré-treinados como camadas de

entrada para tarefas específicas. O objetivo da mudança foi reduzir o custo computacional. Com relação à análise sintática, eles utilizam uma versão simplificada do *biaffine parser* (DOZAT; MANNING, 2016), na qual os *embeddings* das palavras da versão original são substituídos por *embeddings* de lemas. O vetor de *features* é formado pela concatenação dos lemas pré-treinados em BERT, com a etiqueta morfológica (PoS) BERT aprendida durante o treino do modelo de etiquetagem, e a representação da última camada BERT dos lemas pré-treinados. Os vetores de *features* alimentam uma rede Bi-LSTM, gerando estados posteriormente usados por dois perceptrons multicamada para gerar a informação de *heads* e dependentes, e outros dois perceptrons multicamada usados para gerar as informações dos arcos de dependências. Os resultados apresentados mostram ganhos de até aproximadamente 2,31% em UAS e 2, 40 % em LAS.

3.2 Pesquisa em Língua Portuguesa

Esta seção resume trinta anos da pesquisa para Língua Portuguesa. Na introdução, apresenta-se o protocolo utilizado para selecionar os trabalhos; em seguida, encontram-se as categorias nas quais os artigos foram classificados e comentados.

3.2.1. Introdução

Em abril de 2020, foram pesquisados todos os artigos disponíveis na *Web of Science Core Collection* (CLARIVATE, 2020), filtrando os termos *Portuguese* e *Parsing* em qualquer tópico. Este critério de busca recuperou cinquenta e nove artigos, publicados entre os anos de 2001 e 2018. Nenhuma publicação foi encontrada antes de 2001 nem após 2018. Além disso, foram filtrados artigos das categorias Inteligência Artificial, Métodos Teóricos, Aplicações Interdisciplinares, Sistemas de Informação, Cibernética, Arquitetura de Hardware e Engenharia de Software para obter publicações mais relacionadas à linguística computacional, que é a abordagem usada nesta tese. Com estes filtros, o número de artigos caiu para trinta e cinco. Além destes, foram incluídos os trabalhos de Julia, Seabra e Siqueira (1995), Bick (2000), Menezes e Neto (2000) e Padilha e Vicari (2000), selecionados de Ladeira (2010) e Sardinha (2005). Ao final, os trabalhos foram agrupados nas categorias analisadores sintáticos, analisadores morfológicos, identificação de

expressões compostas, analisadores semânticos, resolução de anáforas, extração de informações, perguntas e respostas, tradução automática, classificação de textos, corpus e gramática e ferramentas.

3.2.2. Analisadores sintáticos

Os autores Julia, Seabra e Siqueira (1995) apresentam um *parser* para a Língua Portuguesa que visa auxiliar linguistas a extrair representações semânticas a partir de requisitos de software escritos em linguagem natural. O analisador examina as sentenças à luz de regras sintáticas, semelhantes às definidas por Chomsky (LEES; CHOMSKY, 1957), e produz representações semânticas a partir da análise sintática e de um conjunto de heurísticas. Os autores ilustram o modelo com alguns exemplos, mas não apresentam resultados quantitativos.

Bick (2000) apresenta o PALAVRAS, um *parser* para a Língua Portuguesa que analisa sentenças com base em regras definidas em *Constraint Grammar* (KARLSSON, 1990). O PALAVRAS funciona de forma incremental, primeiro encontrando as etiquetas morfológicas e, em seguida, identificando suas funções sintáticas. O analisador elimina a ambiguidade de interpretações morfológicas aplicando regras que usam condições contextuais para restringir as classificações possíveis, selecionando o rótulo considerado mais apropriado. As funções sintáticas são identificadas pelo emprego de regras produtivas e restritivas, a primeira mapeando rótulos ambíguos e a última rejeitando rótulos inválidos em função do contexto. O *parser* conclui a análise convertendo a estrutura plana original da *Constraint Grammar* para o formato de árvore de derivação. Bick testou o analisador em textos irrestritos e desconhecidos, relatando precisão superior a 99% para marcação morfológica e 97% para classificação de função de sintaxe.

Bonfante e Nunes (2002) apresentam um analisador lexicalizado, no qual a palavra principal orienta a análise sintática e as demais palavras se comportam como seus modificadores, influenciando a aplicação das regras da gramática. O analisador constrói a estrutura sintática das sentenças por meio de um processo ascendente guiado pela probabilidade de a palavra principal se conectar com seu modificador para formar um sintagma. O conjunto de dados de treinamento é composto por sentenças do corpus NILC (COMPUTACIONAL, 1996), anotadas sintaticamente

pelo *parser* PALAVRAS (BICK, 2000). Os autores não apresentam resultados quantitativos, mas afirmam que a pré-identificação de sintagmas nominais traz benefícios qualitativos, pois facilita o trabalho do *parser* para eliminar combinações pouco prováveis de palavras e também pode ser utilizada para outras aplicações, como recuperação de informação.

Wing e Baldridge (2006) propõem um *parser* generativo dirigido por *head words* combinando o *treebank* Floresta Sintá(c)tica (AFONSO et al., 2002) com o motor do *parser* multilíngue Bikel (BIKEL, 2002, 2004). Os autores converteram o formato de anotação do Floresta para o padrão do *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), criando uma versão específica para o português, que incluía marcação explícita de *head words*, representação aprimorada de conjunções e cláusulas relativas distintas. Eles também adaptaram o *parser* de Bikel para analisar textos em português, criando heurísticas de descoberta de *head words*, *features* morfológicas, marcação de argumentos e não argumentos e opções de calibragem. O analisador foi testado em um subconjunto do Floresta e os resultados foram calculados de acordo com o F-score e métricas de precisão de dependência. Foram testadas diferentes configurações de etiquetas e parametrizações do analisador, sendo que a melhor atingiu F-score de 63,3%.

Silva et al. (2010) propõem adaptar para a Língua Portuguesa analisadores originalmente desenvolvidos para o inglês, visando beneficiar-se de sua estabilidade e alto desempenho, bem como aproveitar os pacotes de extensão que os adaptam para outras línguas. Os autores avaliaram os *parsers* de Bikel (BIKEL, 2002), Stanford (KLEIN; MANNING, 2003b) e Berkeley (PETROV et al., 2006) aplicados ao CINTIL *Treebank* (COSTA; BRANCO, 2010), que segue o mesmo padrão de rotulagem do *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993). Os resultados mostraram F-score de 84,97% para o *parser* de *Bikel*, F-score de 88,07% para o *parser* de Stanford e F-score de 89,33% para o *parser* de Berkeley. Esses desempenhos são comparáveis aos obtidos originalmente para o idioma inglês. Os autores explicaram que o *parser* de Berkeley foi o analisador que obteve os melhores resultados, possivelmente pelo fato de ser o menos dependente das regras da linguagem para encontrar *head words*. Eles também implantaram outros recursos linguísticos, tais como lematização, reconhecimento de nomes de

entidades e reconhecimento semântico de nomes de entidade, buscando melhorias no desempenho. Os analisadores foram submetidos a uma nova bateria de testes, indicando ganhos em todos os casos, com os F-scores de Berkeley variando de 89,55% a 90,34%. Finalmente, os autores testaram a influência das etiquetas morfológicas (POS) na análise sintática, usando um etiquetador hipotético ideal, e o analisador de Berkeley alcançou F-score de 95,61%.

Agustini, Gamallo, e Lopes (2002) propõem um método não supervisionado para aprender subcategorizações com o objetivo de melhorar o desempenho de análise. Seu método identifica classes semânticas que são usadas para restringir combinações sintáticas. As informações aprendidas com o método são usadas para resolver ambiguidades de ligação de nós não terminais: NP-PP, VP-NP e VP-PP. Eles testaram o seu modelo em um pequeno corpus português de casos jurídicos (*Portuguese General Attorney Opinions*), no qual alcançaram em média 94,61% de precisão e 19,94% de *cobertura*. Embora a precisão tenha mostrado resultados promissores, a cobertura foi muito baixa, o que foi justificado pelos autores como consequência do tamanho reduzido do corpus.

Martins, Nunes e Hasegawa (2003) apresentam o Curupira, um analisador de propósito geral que fornece todas as análises sintáticas possíveis para frases do português brasileiro. O Curupira analisa as sentenças de cima para baixo, aplicando regras gramaticais priorizadas e um léxico compilado de acordo com o mecanismo descrito por Kowaltowski et al. (1988). O analisador não faz nenhuma desambiguação sintática ou validação semântica, e as análises que ele gera não apresentam nenhuma ordem de preferência. Os autores testaram o *parser* em um conjunto de 297 frases, principalmente manchetes de páginas da web dos mais importantes jornais brasileiros. O Curupira trouxe a análise mais esperada, em primeiro lugar, para 38% das sentenças e, ao menos uma estrutura sintática possível, para 75% das sentenças, sendo que em 17,57% delas a estrutura esperada não estava presente. O Curupira não conseguiu apresentar nenhum resultado para 25% das sentenças, a maioria delas envolvendo uso metalinguístico do português brasileiro e inversões sintáticas.

Gamallo, Lopes e Agustini (2005) propõem um método para resolver ambiguidades de ligação de nós não terminais aprendendo as posições sintáticas que as palavras podem ocupar em uma frase. O método leva em consideração não apenas a distribuição de palavras nas frases, mas também o papel que a semântica tem de impor restrições às posições sintáticas. Os autores trabalham com o conceito de palavra principal (cabeça) e palavras dependentes e assumem que as restrições semânticas são aplicadas não apenas pela palavra principal à dependente, mas, ao contrário, pela palavra dependente para selecionar as palavras-cabeça específicas. O método foi testado em dois corpora diferentes, comparando a resolução de anexação de três diferentes tipos de sequências de sintagmas, chegando a 4 pontos a mais em precisão e cobertura quando comparados com métodos baseados em similaridade de palavras.

Zeng et al. (2014) dizem que a precisão dos métodos de análise baseados em GPLC pode ser melhorada por meio da decomposição da anotação das árvores de análise. Além disso, observam que *Latent Variable Grammars* (MATSUZAKI; MIYAO; TSUJII, 2005), um formalismo que estende as anotações com variáveis latentes, vem produzindo resultados positivos. Porém, essa técnica não se adéqua a textos com palavras desconhecidas, pois não há garantia de que elas façam parte do modelo lexical adotado. Os autores propõem um método de similaridade baseado em grafos para descobrir o conhecimento lexical de extensos conjuntos de dados não rotulados, incorporando-os ao conjunto de dados rotulados. Subcategorias e probabilidades lexicais são inferidas, assumindo que palavras diferentes em contextos sintáticos semelhantes apresentem probabilidades próximas. Eles avaliaram sua proposta com o analisador de *Berkeley* (PETROV et al., 2006) e o Bosque, um subconjunto revisado do Floresta Sintá(c)tica (AFONSO et al., 2002), aprimorado com os dados não rotulados do Europarl Parallel Corpus (KOEHN, 2005), obtendo F-score de 84,24%.

Mielens, Sun e Baldridge (2015) propõem uma estratégia para inserir dependências em textos parcialmente anotados usando *Graph Fragment Language* (SCHNEIDER et al., 2013), de modo que um analisador de dependência padrão possa ser treinado usando todas as anotações. As anotações completas que saem do processo são usadas como entrada para treinar o Turbo *Parser* (MARTINS et al., 2010). Os

autores avaliaram a aplicabilidade de sua abordagem nos idiomas inglês, chinês, português e quiniaruanda. Em português, eles realizaram os testes em um subconjunto de dados derivado do Bosque (AFONSO et al., 2002), concluindo que seu procedimento melhora significativamente o desempenho em relação ao descarte das anotações parciais, requerendo apenas um pequeno conjunto de sentenças anotadas.

Padovani e Neto (2017) apresentam o Linguístico, um *parser* projetado para alterar dinamicamente seu comportamento, modificando sua configuração em resposta a eventos identificados durante a execução. O Linguístico incorpora módulos complementares para dividir textos em sentenças e sentenças em *tokens*. Além disso, um etiquetador morfológico, baseado em autômatos adaptativos (NETO, 1994), fornece as etiquetas de classes gramaticais que o analisador sintático usa para análise. Os autores fizeram testes preliminares no *treebank* CINTIL (COSTA; BRANCO, 2010), alcançando F-score de 88,74%.

3.2.3. Analisadores morfológicos

Menezes e Neto (2000) propõem um identificador morfológico que infere informações linguísticas sobre o léxico e seu contexto, e retém esse conhecimento usando Autômatos Adaptativos (NETO, 1994), para que possa ser recuperado para marcação ou classificação. Durante a fase de treinamento, um banco de dados de palavras conhecidas é armazenado em uma estrutura baseada em árvore, onde cada nó representa uma letra e as folhas representam as etiquetas ordenadas por sua frequência. Prefixos e sufixos são armazenados separadamente, para que não haja repetição. O classificador inicialmente seleciona etiquetas de palavras conhecidas, depois aplica dinamicamente heurísticas em sufixos de palavras para inferir etiquetas de palavras desconhecidas e, finalmente, refina os resultados, de acordo com o contexto. Os autores testaram seu método no corpus Tycho Brahe (GALVES, 2018), alcançando uma precisão de 91,01%.

Padilha e Vicari (2000) propõem um modelo teórico, utilizando máquinas de estados finitos para a análise da morfologia do português. Os autores afirmam que as máquinas de estados finitos são adequadas para o processamento morfológico da Língua Portuguesa, mas constataram sua limitação para tratar novas construções,

uma vez que não são capazes de diferenciar mapeamentos ambíguos e não dispõem de algoritmos de aprendizagem para novas transformações.

Marques e Lopes (2005) afirmam que o principal problema dos etiquetadores morfossintáticos é o esforço necessário para adaptar esses sistemas a novas linguagens, tipos ou gêneros de texto, pois eles requerem grandes quantidades de textos etiquetados para aprendizagem ou construção de sistemas complexos de regras de desambiguação. Os autores propõem superar este problema introduzindo um modelo no qual um sistema de análise lexical e uma rede neural cooperam para treinar marcadores neurais. A precisão média do modelo é de 91% quando ele é treinado com um corpus de 5400 palavras e 96% quando treinado com um corpus de 18.865 palavras marcadas.

Quaresma et al. (2014) observam que aplicações de análise de sentimento classificam sentimentos em conjuntos de emoções positivas, negativas e neutras, e propõem um etiquetador que aplica conceitos de Modalidade (SALKIE, 1988) para ter uma classificação mais refinada, marcando verbos ambíguos de alta frequência da Língua Portuguesa. O etiquetador usa os rótulos morfossintáticos atribuídos pelo analisador PALAVRAS (BICK, 2000), enriquecidos com um esquema de anotação específico, e aplica o algoritmo *Support Vector Machine* (SVM) para inferir os valores modais dos verbos. Os autores testaram sua proposta em uma amostra de 160 mil *tokens*, melhorando o F-score em valores modais de 35,3% para 81,1%.

Fernandes, Rodrigues e Milidiú (2014) dizem que os ganhos de desempenho em etiquetadores de classes gramaticais ainda são importantes porque as saídas que eles geram são usadas em tarefas mais complexas, como análise sintática, rotulação de função semântica e extração de informações. Em seguida, eles propõem um etiquetador baseado no *perceptron* estruturado de Collins (2002) para mapear o par formado pelo *token* e pela sequência de etiquetas nos parâmetros do *perceptron*. Eles testaram o etiquetador nas versões original e revisada do corpus MacMorpho (ALUÍSIO et al., 2003; FONSECA; ROSA, 2013) alcançando F-score de 97,18% no primeiro e 93,67% no segundo.

3.2.4. Identificação de expressões compostas

Ramisch et al. (2010) chamam a atenção para a importância de identificar expressões compostas como preparação para tarefas complexas de PLN, como análises sintática e semântica. Os autores propõem uma estratégia híbrida que combina medidas de associação estatísticas aplicadas a bigramas e trigramas, com alinhamento de palavras entre a Língua Inglesa e o Português, produzindo um modelo ponderado que visa melhorar o desempenho como um todo. A combinação dos dois métodos foi feita através da aplicação de uma Rede Bayesiana, que foi treinada com variáveis criadas a partir dos resultados de cada método. Eles testaram seu modelo no *Corpus of Pediatrics* (COULTHARD, 2005), alcançando F-score de cerca de 50%, contra 20,59% de medidas de associação estatísticas e 1,78% de métodos baseados em alinhamento individualmente.

Baptista, Mamede e Gomes (2010) destacam a importância dos verbos auxiliares na Língua Portuguesa e descrevem construções de tais verbos no português europeu com o objetivo de fornecer árvores de derivação corretas para tarefas subsequentes de Processamento de Línguagem Natural. As construções propostas foram testadas em um pequeno corpus de 20.296 palavras, alcançando precisão de 99,4% e *cobertura* de 95,5%.

Baptista, Mamede e Markov (2014) descrevem um método para integrar expressões idiomáticas em um *parser* de dependência. Seu método aplica regras de consulta sobre dependências morfossintáticas, identificadas pelo analisador em estágios anteriores, criando, ao final, uma dependência específica denominada FIXED, que vincula o verbo aos seus argumentos. Os autores avaliaram seu método testando a identificação de expressões idiomáticas verbais relacionadas a substantivos de partes do corpo em um subconjunto do Cetem Publico (“What is CETEMPublico - Linguateca”, 2020) e alcançaram F-score de 87%.

Zilio et al. (2016) dizem que as expressões de palavras compostas têm características linguísticas ou estatísticas que as tornam difíceis de serem encontradas. Existem duas abordagens bem conhecidas sobre como identificá-las. A primeira conta com técnicas de análise e são mais adequadas para lidar com dependências de longa distância, estruturas internas e apassivação. A segunda

abordagem é baseada em estatísticas e traz melhores resultados ao analisar características estatísticas das expressões. Os autores comparam os dois métodos e sugerem que uma abordagem combinada pode trazer melhores resultados.

Correia, Baptista e Mamede (2016) observam que padrões de ocorrência conjunta permitem uma melhor compreensão do uso e do significado das palavras, e apresentam o *Syntax Deep Explorer*, um sistema que identifica ocorrências conjuntas aplicando dependências sintáticas em corpora. O modelo resume as ocorrências encontradas, apresentando-as aos usuários através de uma interface de programa de computador. Os autores testaram o sistema no Cetem Publico ("What is CETEMPublico - Linguateca", 2020) e relataram resultados positivos tanto na precisão quanto no tempo de resposta.

3.2.5. Analisadores semânticos

Rosa (2007) descreve o theta-PRED, um sistema que gera grades temáticas de sentenças em português. As grades temáticas são estruturas que representam a relação semântica entre as palavras da frase. O sistema utiliza a gramática de Richard Montague, segundo a qual o significado da sentença é uma função dos significados de suas partes e seu modo de combinação sintática (DOWTY; WALL; PETERS, 1981). O theta-PRED é composto por um módulo simbólico que pré-analisa as sentenças de acordo com a gramática montagoviana, gerando representações lógicas que são usadas para treinar um módulo conexionista, usado para a inferência de papéis que resultam na grade temática. O autor testou o módulo conexionista em um conjunto de 120 palavras, relatando uma precisão de 94%.

Maziero, Hirst e Pardo (2016) introduzem o assunto da Análise do Discurso, um processo para reconhecimento automático das relações de coordenação e subordinação, chamando a atenção para duas limitações importantes da abordagem da análise simbólica: a dependência de padrões desenvolvidos manualmente por especialistas e a falta de generalização devido à dependência dos padrões aos domínios em que foram desenvolvidos. Os autores apresentam versões adaptadas para o português de dois trabalhos originalmente desenvolvidos para o inglês que usam técnicas de aprendizado de máquina baseadas em *features* para superar essas limitações (SORICUT; MARCU, 2003; HERNAULT et al., 2010). Eles testaram

seus modelos em quatro corpora combinados, alcançando F-score de 35% e 52% e ganhos de 26% e 22% respectivamente.

Marques e Beuls (2016b) explicam o problema da definição da colocação dos pronomes clíticos nas Línguas Latinas, dando especial atenção à complexidade do Português Europeu, que não define a posição dos clíticos em termos de verbos, mas sim no que diz respeito ao contexto em que os verbos são inseridos. Os autores propõem um modelo computacional inicial, focado em pronomes proclíticos, que utiliza *Fluid Construction Grammar* (STEELS, 2011, 2012), não apenas para identificar a posição correta dos clíticos, mas também para gerar sentenças sem incorrer em excessos. Testes preliminares indicaram que a geração excessiva mais comum estava na ordem das palavras e que o modelo era capaz, em alguns casos, de verificar sentenças não gramaticais e sugerir a ordem correta.

Anchiêta e Pardo (2018) explicam o papel desempenhado pelos analisadores semânticos e listam alguns padrões usados para representação de significado, chamando atenção especial para o padrão *Abstract Meaning Representation* (AMR), devido à sua estrutura simplificada que gera representações de fácil leitura e avaliação. Os autores também observam que as técnicas de análise semântica mais comuns são baseadas em algoritmos de aprendizado de máquina e corpora anotados, recursos que não estão disponíveis para a Língua Portuguesa, e propõem um analisador semântico baseado em regras para superar essa limitação. Sua proposta compreende um conjunto de regras para a criação de representações AMR baseadas em análises sintáticas geradas pelo *parser* PALAVRAS (BICK, 2000) e rótulos de argumento-predicado provenientes do etiquetador de papéis semânticos Brazilis (HARTMANN; DURAN; ALUÍSIO, 2016). Eles testaram seu analisador no corpus *Little Prince* (ANCHIÊTA; PARDO, 2019), comumente usado para este tipo de avaliação, e alcançaram F-score Smatch (CAI; KNIGHT, 2013) de 53,5%, em comparação com 58% dos analisadores AMR desenvolvidos para o inglês.

3.2.6. Resolução de anáforas

Palomar et al. (2001) apresentam um algoritmo para resolver anáforas, um problema linguístico que envolve encontrar uma expressão referenciada por um sintagma nominal ou pronominal. Seu algoritmo visa originalmente encontrar sintagmas

nominais referenciados por pronomes em textos espanhóis, podendo ser adaptado para o português, entre outras línguas. Os autores identificaram as características dos pronomes, tais como concordância morfológica e condições sintáticas, e as replicaram no algoritmo, criando regras que eliminam hipóteses não válidas. Eles também definiram um conjunto de heurísticas que ordenam as hipóteses restantes visando selecionar a melhor. O algoritmo foi avaliado em uma amostra do corpus *Blue Book* (LEÓN; SERRANO, 1995), alcançando uma precisão de 76,8%.

3.2.7. Extração de informações

Bick (2003) apresenta um sistema de extração de informações de textos em português que interage com usuários por intermédio de perguntas e respostas. O sistema usa regras linguísticas definidas pelo formalismo *Constraint Grammar* (VANHALTEREN, 1996) para atribuir etiquetas morfossintáticas e semânticas aos constituintes das perguntas inseridas pelos usuários. Essas etiquetas são usadas para encontrar os documentos corretos, que, uma vez selecionados, são formatados e apresentados como resposta ao usuário. Além da correspondência exata, o sistema pode expandir os resultados da pesquisa afrouxando as regras sintáticas para permitir que o usuário possa refinar suas perguntas a fim de encontrar as respostas que deseja.

Gamallo, Lopes e Agustini (2005) destacam a importância das técnicas de extração de informações para gerar representações semânticas. Os autores propõem um método baseado em regras para criar representações semânticas no formato de proposições básicas, triplas que denotam uma relação semântica entre um verbo e dois argumentos. O método proposto extrai as proposições básicas a partir de análises de dependências formatadas no padrão CoNLL-X (BUCHHOLZ; MARSI, 2006; NIVRE et al., 2007), sem necessidade de usar dados para treinamento. Os autores testaram seu método usando 103 sentenças do corpus CorpusEco (ZAVAGLIA, 2006), analisadas com o *parser* DepPattern 3.0 (OTERO; GONZÁLEZ LÓPEZ, 2011) e etiquetadas com o *tagger* Freeling 3.0 (PADRO; STANILOVSKY, 2012). Ao final, 190 triplas foram extraídas, alcançando precisão de 53%.

3.2.8. Perguntas e respostas

Quaresma e Pimenta Rodrigues (2003) introduzem um sistema de diálogo que permite aos usuários interagir com a base de dados do Procurador-Geral da República através de uma interface web e de um mecanismo de interpretação de linguagem natural. Esse mecanismo converte os textos informados pelos usuários em uma lógica declarativa de predicados que serve para recuperar os documentos que são apresentados na interface web. Os autores desenvolveram um pequeno protótipo que usaram para fazer testes preliminares, mas não apresentaram resultados quantitativos.

Marrafa et al. (2004) descreve o INQUER, um sistema de perguntas e respostas que permite aos usuários se comunicar com a base de dados do WordNet.pt (FELLBAUM, 1998), utilizando a Língua Portuguesa sem restrições. Seu sistema possui um módulo para realizar análises sintáticas e semânticas, gerando uma representação estruturada dos textos de entrada que é utilizada para extrair e inferir informações da base de dados. Por fim, o INQUER converte as respostas em formato de linguagem natural, usando um algoritmo que combina os resultados do banco de dados com a representação estruturada da pergunta.

3.2.9. Tradução automática

Armentano-Oller et al. (2006) apresentam um sistema de tradução automática entre o português e o espanhol europeu, desenvolvido com a ferramenta Apertium (APERTIUM, 2020), uma biblioteca de código livre especializada em tradução automática. Eles fornecem uma visão geral da Arquitetura Apertium, enfatizando o uso de técnicas de transferência superficial, juntamente com explicações sobre cada componente da biblioteca. Ao final, os autores apresentam o desempenho obtido nos testes preliminares, indicando resultados promissores tanto na cobertura do texto dos indicadores quanto nas taxas de erro das palavras de tradução.

Oliveira et al. (2007) observam que as pesquisas em recuperação de informações em diferentes línguas têm atraído atenção devido à crescente disponibilidade de documentos multilíngues, mas também notam que essa linha de investigação sofre com as questões de ambiguidade inerentes à tradução de línguas naturais. Os

autores propõem um método para traduzir consultas entre o chinês e o português analisando textos em ambas as línguas. O método proposto utiliza *Constraint Synchronous Grammar* (WONG; DONG; HU, 2006), um tipo de Gramática Livre de Contexto cujas regras de produção são descritas em termos de padrões do idioma de origem, padrões do idioma de destino, e os critérios para convertê-los. Os autores também demonstram como seu método remove ambiguidades em diferentes etapas da tradução, mas não apresentam experimentos com resultados quantitativos.

Oliveira, Wong e Hong (2010) observam o baixo desempenho dos tradutores baseados em regras para tratar frases longas. Os autores explicam que tais tradutores enfrentam dificuldades para selecionar a árvore de análise correta entre as diferentes possibilidades geradas durante o processo de tradução. Eles propõem um método para melhorar o desempenho de tais tradutores, dividindo o comprimento das sentenças e usando *Constraint Synchronous Grammar* (WONG; DONG; HU, 2006) para analisar simultaneamente a linguagem de origem e de destino visando reduzir ambiguidades. Os autores testaram sua proposta em um conjunto de teste de 2070 frases, selecionadas aleatoriamente no sítio do Governo de Macau (“Macao Special Administrative Region Government Portal”, 2010), com uma extensão média de 19 palavras por frase, e observaram uma redução média de 45,72% no tempo processamento e também uma melhor qualidade de tradução.

3.2.10. Classificação de textos

Gonçalves e Quaresma (2008) observam os benefícios de técnicas de classificação de textos para organizar e filtrar informações de uma grande quantidade de documentos, e investigam o desempenho de estruturas linguísticas em algoritmos de classificação de textos. Os autores realizaram diversos testes, avaliando classificadores treinados com variáveis criadas a partir de informações morfológicas, morfossintáticas e sintáticas. Eles usaram o algoritmo *Support Vector Machine* (SVM) para classificação de textos e um subconjunto de artigos do corpus Cetem Publico (“What is CETEMPublico - Linguateca”, 2020) para avaliar cada cenário de teste. Os resultados indicaram desempenhos semelhantes entre os classificadores

morfológicos e morfossintáticos, e desempenho inferior do classificador sintático, que ficou abaixo dos outros dois.

3.2.11. Corpus e gramática

Boos et al. (2014) chamam a atenção para a importância de grandes corpora para o PLN e apresentam o brWaC, um método de três etapas para a construção de um corpus de bilhões de palavras a partir de textos em português do Brasil coletados em páginas da web. Inicialmente, uma lista de palavras de média frequência é enviada para um mecanismo de busca, que retorna textos com as palavras desejadas. Em seguida, os textos são limpos, os registros duplicados são removidos e as palavras são anotadas com etiquetas de classes gramaticais. Finalmente, uma etapa de controle de qualidade garante que os documentos foram escritos em português do Brasil. Os autores aplicaram seu método para coletar mais de 1.300.000 documentos, correspondendo a 3 bilhões de palavras, e relataram ter etiquetado 8% delas.

Costa, Branco (2010) introduzem o LXGram, uma gramática de propósito geral para a Língua Portuguesa, que foi implementada seguindo o modelo *Head-Driven Phrase Grammar* (HPSG) (GUNJI; POLLARD; SAG, 1996), usando generalizações linguísticas codificadas manualmente e um modelo estocástico para desambiguação de análises. LXGram suporta o português europeu e brasileiro, e uma ampla gama de fenômenos linguísticos, como dependências de longa distância, coordenação e subordinação. Os autores testaram a gramática em um subconjunto da Wikipedia em português, junto com os corpora Cetem Publico (“What is CETEMPublico - Linguateca”, 2020) e Cetem Folha (CETEM, 2020), obtendo mais de 30% de cobertura em textos de jornais, e de um subconjunto dessas frases, observaram que 60% foram representados corretamente e 40% foram desambiguados corretamente.

Costa e Kepler (2014) observam que métodos supervisionados de indução gramatical apresentam bom desempenho, mas são dependentes de corpora anotados, que demandam muito trabalho e requerem conhecimentos linguísticos para serem construídos. Por outro lado, os métodos não supervisionados, embora muito desejados, ainda não se igualam ao desempenho das abordagens supervisionadas. Os autores apresentam um modelo semissupervisionado para

preencher esta lacuna, baseado no modelo *Constituent Context Model* (KLEIN; MANNING, 2001), que é capaz de usar fragmentos supervisionados durante a fase de aprendizagem. Eles testaram o modelo no corpus Tycho Brahe (GALVES, 2018), alcançando F-score de 64,32%, em comparação com 50,50% dos métodos não supervisionados usados como referência.

3.2.12. Ferramentas

Alencar et al. (2014) observam que transdutores de estado finito são a forma preferencial de implementação de analisadores morfológicos devido à sua compactação e rápido tempo de processamento. No entanto, os autores percebem a carência de transdutores lexicais para a Língua Portuguesa distribuídos através de licenciamento *Open Source* e apresentam uma biblioteca para preencher essa lacuna. Sua biblioteca, chamada JMorpher, desenvolvida em Java, tem o objetivo de ser portável para qualquer dispositivo móvel embasado no sistema operacional Android. Os autores avaliaram o JMorpher no corpus MacMorpho (ALUÍSIO et al., 2003), observando resultados equivalentes aos de bibliotecas usadas como referência com relação à correção, mas identificaram possibilidades de melhorias com relação ao desempenho no Android, principalmente quando comparado com os resultados obtidos em computadores pessoais.

Gamallo et al. (2018) apresentam LinguaKit, um conjunto de ferramentas destinadas a apoiar a realização de tarefas de análise linguística e extração de informações. O Linguakit é composto por quatro módulos, cada um deles dedicado a tarefas específicas, abrangendo análise básica, análise profunda, extração e aplicações. O módulo de análise básica é organizado em funções direcionadas às etapas iniciais do PLN, tais como segmentação de frase e separação em *tokens*. O módulo de análise profunda compreende funções para análise morfossintática, tais como lematização, identificação de classes gramaticais e análise sintática baseada em dependência. O módulo de extração possui recursos para identificar expressões compostas, minerar opiniões de texto e extrair relações semânticas. O módulo final é dedicado a aplicações, tais como sumarização, anotação semântica e identificação de linguagem. Os autores também detalham a arquitetura do sistema, dando especial atenção à integração com infraestrutura de *big data*, direcionada para o

desenvolvimento de aplicações escaláveis. Finalmente, eles descrevem sistemas semelhantes como *Stanford Core NLP* (MANNING et al., 2015), *Freeling* (PADRO; STANILOVSKY, 2012), OpenNLP (APACHE, 2016), NLTK (BIRD; KLEIN; LOPER, 2016) e spaCY (SPACY, 2017), explicando os diferenciais do Linguakit.

3.3 Análise comparativa

Observa-se que as primeiras pesquisas para o Português estavam muito concentradas em técnicas baseadas em regras, como é o caso de Julia, Seabra e Siqueira (1995), e especialmente por Bick (2000), cujo *parser* PALAVRAS foi utilizado por diversos outros trabalhos tais como Bonfante e Nunes (2002), Quaresma et al. (2014) e Anchiêta e Pardo (2018). Por outro lado, técnicas de análise parcial, como as propostas por Vanhalteren (1996), Voutilainen (1995b) e Schwarz (1990) não foram encontradas em pesquisas para a Língua Portuguesa.

Analisadores dependentes de regras criadas manualmente muitas vezes falham em entregar uma análise porque não encontram uma palavra no léxico ou uma regra na gramática. Essas limitações levaram a abordagens estocásticas que fornecem pelo menos uma análise subótima (SILVA et al., 2010). Bonfante e Nunes (2002) introduziram o conceito de *parser* lexicalizado para o português, baseando-se em modelos lexicalizados generativos como Collins (1997) e Charniak (2000), mas não apresentaram resultados quantitativos em seu trabalho.

A resolução de ambiguidades de ligação utilizando o conceito de *head-word* e palavras dependentes também pode ser encontrada na pesquisa para Língua Portuguesa em Gamallo, Lopes e Agustini (2005). Alternativas à lexicalização, como a subcategorização proposta por Klein e Manning (2003a) e Petrov et al. (2006) são encontradas em Agustini, Gamallo e Lopes (2002) e Zeng et al. (2014). No entanto, nada é encontrado nas investigações sobre o uso de modelos discriminativos, como apresentado por Taskar et al. (2004) e Finkel, Kleeman e Manning (2008).

As redes neurais já existem há muito tempo, inicialmente aplicadas em gramáticas ilustrativas como Elman (1991), estenderam sua cobertura para cenários reais, como os métodos híbridos apresentados por Henderson (2003, 2004), ou em tarefas relacionadas, tais como a portabilidade de domínio, no modelo de Titov e Henderson

(2006). Um aspecto importante dos métodos discriminativos, como as redes neurais, são os recursos computacionais necessários para treinar os modelos. Carreras, Collins e Koo (2008) fornecem uma solução alternativa para este problema, usando um modelo de dependência para restringir o espaço de busca manipulado pela rede neural. Não foram encontrados trabalhos com abordagens semelhantes em pesquisas para a Língua Portuguesa.

Pacotes para converter analisadores de última geração para outras linguagens se tornaram comuns. Uma abordagem multilíngue é apresentada por Jaf e Calder (2019), que adaptou um analisador *shift-reduce*, pré-treinado em inglês, para analisar o cazaque e pré-treinado em farsi para analisar kurmanji. Wing e Baldridge (2006) criaram uma versão em português de *Penn Treebank* (MARCUS; SANTORINI; MARCINKIEWICZ, 1993), modificando o formato de anotação de Floresta Sintá (c) tica (AFONSO et al., 2002), e adaptaram o *parser* de Bikel (BIKEL, 2002) para interpretar a Língua Portuguesa, concluindo que o uso de analisadores prontos é uma opção realista. Outro trabalho seguindo essa abordagem é apresentado por Silva et al. (2010) que adaptaram Berkeley (PETROV et al., 2006), Bikel (BIKEL, 2002) e Stanford (KLEIN; MANNING, 2003b) para o português, obtendo resultados semelhantes aos relatados para o inglês. No entanto, o trabalho necessário para adaptar os padrões de rotulagem ou para incorporar regras específicas do idioma pode inviabilizar os pacotes multilíngues.

Os conceitos de *word embedding* e *deep learning* mudaram a maneira de realizar as tarefas do PLN de muitas maneiras (YOUNG et al., 2018). No que diz respeito à análise, métodos que usam RNN e RNRv, como Socher et al. (2013), trouxeram alternativas à subcategorização e lexicalização, e outras, como Menchetti et al. (2005), deram uma nova perspectiva com relação às técnicas de reclassificação. Percebem-se também novas formas de análise, como as árvores incrementais de Costa et al. (2003), e a pilha de etiquetadores de Collobert, (2011). Em outro artigo, COLLOBERT et al. (2011) basicamente propõem uma nova maneira de fazer PLN, coletando informações sintáticas como um efeito colateral de sua análise semântica. Representações densas e redes neurais também foram usadas em novas abordagens para analisadores de dependência, como em Chen; Manning (2014),

Dyer et al. (2015) e Zhou et al. (2015). Nenhuma dessas ideias foi encontrada em pesquisas para a Língua Portuguesa.

Muitos são os trabalhos que combinam diferentes técnicas, com o objetivo de melhorar a eficiência ou preencher lacunas específicas no campo da análise sintática. Novas perspectivas para *frameworks* estabelecidos, como o método para lidar com regras unárias no analisador *shift-reduce* proposto por Zhu et al. (2013), ou o modelo de rede neural para inferir as ações de deslocamento e redução do analisador baseado em transição proposto por Weiss et al. (2015), ou mesmo o uso de informações de fronteira para eliminar a ambiguidade de ligação de sintagmas, conforme proposto por Chen et al.(2017), são encontrados junto com abordagens mais transformadoras, que, por exemplo, alteram as características básicas da análise sintática, conforme proposto pelo modelo de Vinyals et al. (2015), que transforma análises constituintes em estruturas lineares para serem treinadas em uma rede neural de sequência a sequência; ou Yang et al. (2017), que combinam saídas de diferentes analisadores com o objetivo de melhorar o desempenho. Jaf e Ramsay (2016) também usam uma abordagem mista, combinando técnicas orientadas a dados com informações gramaticais em *parser* para a Língua Árabe. Yuan, Jiang e Tu (2019) vão ainda mais longe, afirmando que os erros em analisadores de dependência são inerentes à leitura de entrada da esquerda para a direita e propõem superar este problema usando uma abordagem bidirecional.

Abordagens mistas também são encontradas nas pesquisas para a Língua Portuguesa, mas em número reduzido. É o caso de Martins, Nunes e Hasegawa (2003) e Padovani e Neto (2017). O primeiro, com o objetivo de apresentar todas as análises de sentenças, sem filtrar opções semânticas ou sintáticas inválidas; e o segundo enfocando o uso de um formalismo dinâmico capaz de modificar sua configuração interna em resposta a eventos externos.

Por outro lado, a maior parte dos trabalhos de pesquisa encontrados para a Língua Portuguesa não é propriamente a respeito de análise sintática, mas sim, de um vasto leque de trabalhos afins, englobando classificação de dados, como etiquetadores morfológicos e identificação de expressões multipalavra; análise semântica, como analisadores semânticos e resolução de anáfora; e aplicativos de

PLN gerais, como extração de informações, recuperação de informações, resposta a perguntas, tradução automática e classificação de texto.

Além disso, percebe-se que os autores desenvolvem suas propostas em diferentes corpora, com diferentes conjuntos de etiquetas, o que dificulta a comparação direta dos resultados. Com algumas exceções, pesquisas em inglês utilizam o *Penn Treebank* e o mesmo padrão foi seguido para chinês e árabe. A padronização pode ajudar a aprimorar a pesquisa e o trabalho de Branco et al. (2010, 2014) e Costa e Branco (2010), visando criar um corpus para o português no estilo *Penn Treebank*, desempenha um papel importante nessa direção. Ainda com relação à metodologia, observa-se que, de maneira geral, tanto nas pesquisas para o português como para as demais, não existe um padrão de testes amplamente utilizado. Foram observadas variações em diversos aspectos, tais como técnica de divisão do corpus entre treino e teste, número de repetições dos testes e tamanho da sentenças consideradas na avaliação. Tais diferenças dificultam a comparação direta dos trabalhos e requerem atenção adicional para interpretação dos resultados.

Por fim, um dos principais desafios para línguas como o português é a falta de recursos para apoiar a investigação, o que tem sido solucionado com bibliotecas e sistemas de código aberto, tais como os apresentados por Gamallo et al. (2018).

4 MÉTODO ADAPTATIVO PARA ANÁLISE SINTÁTICA

Este capítulo apresenta o método proposto nesta tese para a análise sintática do Português Brasileiro. Inicialmente, é feita uma introdução, apresentando o modelo linguístico utilizado pelo analisador, em seguida, é definido o modelo computacional e, por fim, é apresentada a formalização do analisador sintático.

4.1 Introdução

Uma vertente recente ainda pouco explorada em Linguística Computacional é a Gramática de Construções. Esta pesquisa não considera o léxico e a sintaxe como módulos rigidamente separados, mas, sim, formando um contínuo de construções que vão desde um item lexical, tal como a palavra “janela”, passando pelas classificações morfológicas, tais como a classificação “adjetivo”, e chegando até construções abstratas, como é o caso da construção transitiva. Neste tipo de sintaxe, cada enunciado é entendido como uma combinação de múltiplas construções diferentes que, juntas, especificam seu significado e forma precisos. Por exemplo, as sentenças “Ela dançou samba” ou “Ele perdeu a cabeça” são instanciações da construção [SN1 V SN2]. Os sintagmas nominais SN1 e SN2 são argumentos do verbo e exercem o papel de sujeito e objeto direto respectivamente. Trata-se de uma visão não derivacional, que utiliza a gramática com base em esquemas abstratos gerais, e não com base em símbolos e regras de derivação, como ocorre com os modelos gerativos (FERRARI, 2011).

A Gramática de Construções adota um modelo que pode ser entendido como um léxico expandido, conseguindo representar um número infinito de sentenças por meio da combinação de diferentes construções gramaticais. Por exemplo, a frase “O que você emprestou ao João?” resulta da combinação da construção interrogativa – QU; a construção bitransitiva SUJEITO + OBJETO DIRETO + OBJETO INDIRETO; e as construções lexicais, ou seja, as palavras que compõem a sentença. Nesta gramática, as frases malformadas são explicadas através do conflito da combinação de elementos construtivos. Por exemplo, na sentença “Correr emprestou dinheiro para João” há uma incompatibilidade entre o verbo correr e o espaço destinado ao sujeito da construção bitransitiva, que deve ser um sintagma nominal. De acordo com os linguistas construcionistas, o conhecimento linguístico de um falante é

formado pela rede de construções gramaticais a que ele tem acesso. Construções gramaticais são as interconexões formadas entre os diferentes padrões do continuum léxico-sintático. É o caso da relação de instanciação estabelecida na expressão “Fala sério!” com o padrão abstrato VERBO $_{\text{imperativo}}$ + ADVERBIO. A análise construcionista geralmente utiliza técnicas de introspecção do analista, na qual os linguistas inventam os dados e constroem sua argumentação tomando por base julgamentos subjetivos de gramaticalidade, que envolvem uso não apenas da sintaxe em si, mas também da semântica e da pragmática. As técnicas experimentais são mais usadas para verificar certas hipóteses explicativas de fenômenos sintáticos (OTHERO; KENEDY, 2015).

Tabela 2. O continuum léxico-sintaxe

Construção	Exemplos
Palavra	Verdura carro, roupa
Classificação morfológica	Verbo substantivo, adjetivo
Expressão idiomática	Abrir mão, abandonar o barco abotoar o paletó
Idiomatismo formal Dar uma de ADJ	Dar uma de maluco Dar uma de desentendido
Construção bitransitiva SUJ V OD OI	João enviou uma carta à Maria Clovis deu um presente para Jacinta
Construção passiva SUJ AUX SV $_{\text{participio}}$ PP $_{\text{por}}$	O prêmio foi dado ao vencedor A bicicleta foi comprada pela empresa

Fonte: Adaptado de (OTHERO; KENEDY, 2015)

A Tabela 2 apresenta o continuum de construções gramaticais organizadas em função do grau de preenchimento e do grau de complexidade interna. Um dos extremos deste continuum trata da noção tradicional do léxico enquanto o outro

corresponde ao domínio da sintaxe. Uma palavra do léxico é considerada uma construção inteiramente preenchida e de complexidade baixa, enquanto uma estrutura bitransitiva é considerada uma construção aberta e de complexidade alta (OTHERO; KENEDY, 2015).

4.2 Definição

O analisador proposto nesta tese apoia-se no continuum apresentado na Tabela 2. Observa-se que tal estrutura pode ser interpretada como uma pilha de operações especializadas em que o resultado gerado por aquelas que se encontram em um nível mais baixo influencia o resultado das operações de nível mais alto. Ou, no jargão construcionista, resultados das construções de menor complexidade e maior completude influenciam na seleção de construções de maior complexidade e menor completude. O corolário desta interpretação é que erros na seleção de construções menos complexas cometidos em etapas iniciais do processo influenciarão negativamente na escolha de construções mais complexas realizadas nas etapas seguintes. Um exemplo prático está na identificação da classificação morfológica das palavras. Se uma palavra possuir mais de uma classificação morfológica e for atribuída a ela uma classificação incorreta, a construção abstrata selecionada a partir deste resultado criará um conteúdo semântico diferente daquele pretendido pelo falante. Outro exemplo é a identificação de expressões idiomáticas. Se a operação de identificação de expressões idiomáticas não levar em conta que esta estrutura é montada a partir da combinação de palavras, ela não será capaz de detectá-las, produzindo uma construção semântica de conteúdo equivocado. Portanto, além do conceito de pilha de operações, vemos que a análise construcionista também necessita de um conjunto de regras para selecionar as estruturas corretas.

No entanto, apenas o conhecimento das regras de seleção e da pilha de operadores não é suficiente, pois estes podem mudar seu comportamento em função da mensagem que o falante deseja transmitir. Tal comportamento é percebido através do contexto na qual a comunicação está inserida, que determina a rede de construções e a semântica a ela associada. Um exemplo é a frase “Bonito hein?” que, dependendo da entonação, pode ter conotação de ironia ou de admiração.

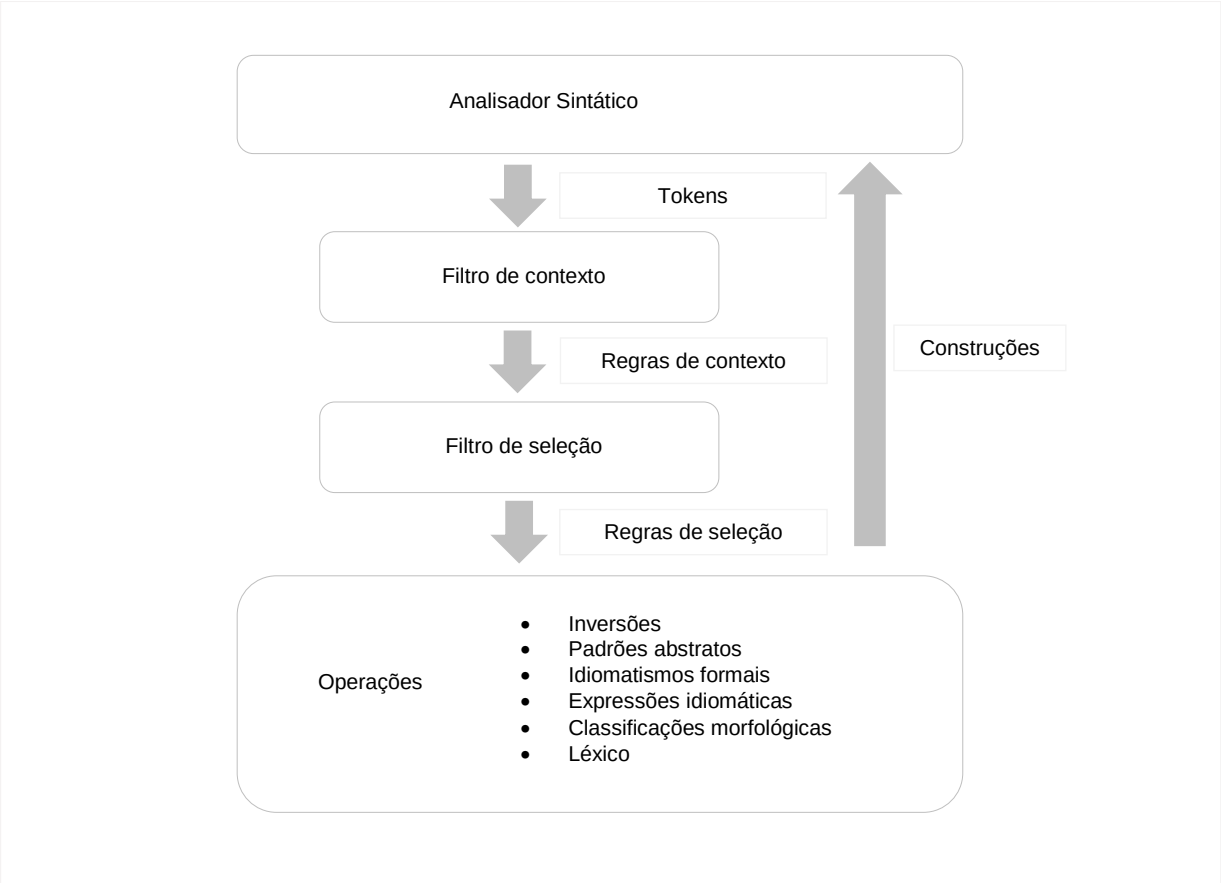


Figura 6 – Analisador morfossintático

Fonte: Autor

Outro exemplo é o caso da interpretação literal de expressões idiomáticas, como é o caso da frase “Joana abandonou o barco”, que pode significar a desistência de uma situação difícil ou a saída efetiva da embarcação. Consequentemente, além da pilha de operações e do conjunto de regras de seleção, também é necessário um mecanismo para a escolha das regras de forma que elas reflitam a real intenção do falante durante a comunicação, ou seja, que considerem o contexto morfossintático, semântico e pragmático. Logo, na representação de um analisador construcionista, as construções estão inseridas em um contexto mais amplo, que fornece informações para filtrar e desambiguar as regras usadas para a seleção completa da estrutura abstrata que representa a sentença. A Figura 6 apresenta uma visão geral de tal analisador, segundo interpretação das características observadas da abordagem construcionista. Nota-se que o analisador sintático utiliza informações de contexto e regras de seleção de construções ao longo de toda a pilha de operações,

desde a identificação do léxico até a seleção dos padrões abstratos. Por exemplo, no caso da construção bitransitiva “João enviou uma carta à Maria.”, o analisador inicialmente precisa identificar as palavras, dividindo a sentença original em *tokens.* Em seguida, o analisador precisa identificar as classificações morfológicas do *tokens*. No passo seguinte, o analisador precisa ser capaz de identificar palavras compostas que possam representar expressões idiomáticas. Depois, é necessário analisar se existem idiomatismos formais e, por fim, o analisador deve escolher o padrão abstrato que melhor represente a sentença. No caso do exemplo citado seria SUJ V OD OI. Nota-se, portanto, que o analisador deve selecionar uma construção preenchendo-a com as informações de que dispõe em cada nível da análise.

Considerando que as construções podem ser criadas levando-se em consideração relações de constituição ou de dependência, concluímos que a análise sintática pode ser feita a partir da busca de construções pré-existentes, com o preenchimento das informações faltantes a partir conteúdo do texto em análise. Um exemplo de construção para representar a relação de constituição da frase acima, usando parênteses para indicar os constituintes e o conjunto de etiquetas do LXGram (COSTA; BRANCO, 2010), seria:

(S (S (NP (N **token**)) (VP (V **token**) (NP (ART **token**) (N' (N **token**) (PP (P **token**) (NP (ART **token**) (N **token**))))))) (PNT .))

Por esta técnica, a análise sintática é feita através da substituição da palavra chave “token” pelos *tokens* da sentença analisada, gerando, ao final do processo, a seguinte representação:

(S (S (NP (N **João**)) (VP (V **enviou**) (NP (ART **uma**) (N' (N **carta**) (PP (P **a**_) (NP (ART **a**) (N **Maria**))))))) (PNT **.**))

Portanto, para que a análise sintática seja realizada, é necessário um conjunto de construções previamente estabelecidas, um conjunto de regras de contexto para auxiliar na seleção de construções e as regras de seleção e substituição.

Esta abordagem traz as seguintes caraterísticas:

a) A análise sintática é feita através da escolha de construções e substituições de palavras-chave;

b) As informações do contexto são usadas tanto na escolha das construções como na identificação dos elementos usados na substituição das palavras-chave;

c) As informações do léxico assumem uma importância significativa, pois ele tem um papel chave na escolha das construções;

d) Quanto maior for a precisão do etiquetador morfológico, menor será a complexidade algorítmica do analisador sintático, pois a análise sintática passa a ser feita por meio de uma busca em um conjunto menor de construções. Na hipótese da existência de um etiquetador morfológico perfeito, o analisador sintático passaria a ser um autômato finito determinístico.

Por outro lado, um conjunto de classificações morfológicas tais como o LXGram, que dispõe de 20 etiquetas, pode gerar uma quantidade intratável de construções. Em uma sentença de 20 palavras, admitindo que as palavras possam assumir livremente qualquer classificação morfológica, chegaríamos a 20^{20} possibilidades, sem considerar as possíveis combinações de constituintes. Tal número reforça a necessidade da aplicação de regras que limitem o conjunto de construções àqueles que realmente representam a linguagem em análise. A Linguística passa a ser a fonte de informações mais imediata para a identificação de tais regras. Técnicas estocásticas também podem ser usadas para desambiguação, dada a dificuldade de criar regras que reflitam toda a complexidade da linguagem natural. Ainda é possível imaginar o uso de heurísticas, quando os padrões anteriores não produzirem os resultados desejados.

Dado este cenário, é importante que o formalismo usado para representar o analisador sintático seja flexível o suficiente para tratar todos os casos, sem adicionar complexidade significativa no processo. A tecnologia adaptativa fornece estes recursos, devido à possibilidade de automodificação, permitindo o carregamento de regras de acordo com o contexto apresentado. No entanto, é necessário definir o formalismo adaptativo que será usado para a representação do analisador sintático. Conforme foi apresentado anteriormente, o analisador sintático não utiliza gramáticas diretamente e, sim, construções delas derivadas. Além disso,

o formalismo escolhido precisa representar as informações de contexto e as regras usadas para a seleção das construções em cada etapa da análise. Vimos que as informações do léxico assumem papel significativo e que quanto maior for a precisão da classificação morfológica, menor será a complexidade algorítmica do analisador. Por fim, o formalismo deve considerar regras linguísticas para desambiguação e seleção de construções, porém sem provocar um *overhead* significativo no analisador. A próxima seção apresenta tal formalismo.

4.3 Formalização

De acordo com Neto (2001), um dispositivo com adaptatividade básica se reduz a um dispositivo não adaptativo acoplado a um mecanismo capaz de alterar o conjunto de regras que define seu comportamento. O mecanismo que converte um dispositivo não adaptativo em um dispositivo com adaptatividade básica é composto por um conjunto de funções especiais, denominadas funções adaptativas, que são acionadas a partir da aplicação de regras a elas associadas que definem o comportamento do dispositivo adaptativo.

O mecanismo responsável pela escolha de opções de análise sintática ME de acordo com parâmetros recebidos (descritos na seção 4.4) pode ser modelado como um dispositivo adaptativo com a seguinte formulação:

$$\mathrm{ME} = (\mathrm{C}, \mathrm{RC}, \mathrm{E}, \mathrm{FA}, \mathrm{c}_0, \mathrm{RC}_0, \mathrm{A})$$

onde:

C é o conjunto de todas as possíveis configurações de ME;

RC é o conjunto das regras de contexto de ME;

E é o conjunto de símbolos válidos de entrada de ME;

FA é o conjunto fixo de funções adaptativas de ME;

$\mathrm{c}_0 \in \mathrm{C}$ é a configuração inicial, única, de ME;

RC_0, é um conjunto fixo de regras de contexto iniciais de ME;

$\mathrm{A} \subseteq \mathrm{C}$ é o conjunto das configurações de aceitação de ME.

O analisador sintático é caracterizado como um dispositivo não adaptativo AS, que se converte em um dispositivo adaptativo em virtude das funções adaptativas FA, acionáveis a partir das regras de contexto RC.

Usando o formalismo de Neto (2001), temos a seguinte formulação para AS:

$$\mathrm{AS} = (\mathrm{C}, \mathrm{S}, \mathrm{c}_0, \mathrm{A}, \mathrm{R})$$

onde:

C é o conjunto de todas as possíveis configurações de AS;

S é o conjunto de símbolos válidos de entrada de AS;

$\mathrm{c}_0 \in \mathrm{C}$ é a única configuração inicial de AS;

$\mathrm{A} \subseteq \mathrm{C}$ é o conjunto de todas as configurações de aceitação de AS;

R é uma relação de mudança de configuração para AS:

$$\mathrm{R} \subseteq \mathrm{C} \times (\mathrm{S} \cup \{\varepsilon\}) \times \mathrm{C}$$

cujos elementos constituem os parâmetros que determinam o funcionamento do dispositivo AS:

$$\mathrm{r} = (\mathrm{c_j}, \mathrm{s}, \mathrm{c'_j}) \in \mathrm{R}, \text{com } \mathrm{c_j}, \mathrm{c'_j} \in \mathrm{C}; \mathrm{s} \in \mathrm{S}$$

os quais podem ser denotados na forma:

$$(\mathrm{c_j}, \mathrm{s}) \rightarrow \mathrm{c'_j}$$

s é o estímulo que, aplicado ao dispositivo AS na configuração $\mathrm{c_j}$, leva-o à sua nova configuração $\mathrm{c'_j}$.

$$\mathrm{c_j} \Rightarrow_{\mathrm{s}} \mathrm{c'_j}$$

A linguagem definida por AS é representada por L(AS):

$$\mathrm{L(AS)} = \{\mathrm{w} \in \mathrm{S}^* \mid \mathrm{c_j} \Rightarrow^*_{\mathrm{w}} \mathrm{c}, \mathrm{c} \in \mathrm{A}\}$$

onde x^* denota o fecho de Kleene de um conjunto ou operador x.

4.4 Análise Sintática

A aplicação de um dispositivo adaptativo simples proposto por (NETO, 2000) e formalizado nesta tese (APÊNDICE A) teve como objetivo dar subsídios a ensaios preliminares de aplicação de técnicas adaptativas em PLN. O dispositivo foi usado para controlar a parametrização do contexto de análise dos textos, através do chaveamento das funcionalidades de PLN:

a) Padrão de construções: Define o padrão usado para selecionar as construções a partir da sequência de etiquetas morfológicas da sentença analisada;

b) Domínio da informação: Define a categoria de textos analisada, por exemplo, textos jornalísticos, literários ou educacionais;

c) Quantidade de *tokens* da sentença: Maneira de filtrar as sentenças pelo seu comprimento, restringindo a análise a sentenças cujo comprimento não exceda um determinado número máximo de *tokens*;

d) Técnica de etiquetagem: Determina a técnica de etiquetagem usada para identificar a categoria morfológica dos *tokens (PoS)*;

e) Similaridade: Refere-se à identificação de uma construção similar, mas não igual, àquela formada pela sequência das etiquetas dos *tokens* da sentença. É o caso, por exemplo, do uso de técnicas de similaridade de *strings*;

f) Estatística: Determina a técnica estatística usada para desambiguar construções válidas para a sentença analisada. Pode ser, por exemplo, a construção mais frequente ou qualquer modelo estatístico de inferência;

g) Identificação de regras candidatas: Identifica os critérios usados para selecionar regras de produção candidatas a partir da leitura dos *tokens* das sentenças;

h) Técnica de desambiguação: Define os critérios de desambiguação usados para escolher a regra de produção entre as regras candidatas;

i) Sentido de análise: Identifica se a sentença será analisada da esquerda para a direita ou da direita para a esquerda;

j) Uso de método híbrido: Identifica se a análise será feita usando método híbrido.

O controle do chaveamento das funcionalidades de PLN é parte integrante dos módulos de análise por construções estáticas, análise por construções dinâmicas e análise combinando construções estáticas e dinâmicas. Estes módulos são apresentados detalhadamente nas próximas seções, assim como as etapas preparatórias.

5 ETAPAS PREPARATÓRIAS

Um analisador sintático é parte de uma estrutura mais ampla de um processo que se inicia na preparação dos textos, passa por etapas de transformação e, por fim, chega à análise em si, com a aplicação de algoritmos específicos que geram as árvores de derivação ou dependência, conforme o formalismo estabelecido. Este capítulo apresenta a especificação dos módulos de apoio necessários para que a análise seja realizada.

5.1 Processo

O processo se inicia com a leitura dos documentos que se deseja analisar. Uma etapa de preparação é necessária para formatar os documentos, dividindo-os primeiramente em sentenças e, depois, em *tokens*. Em seguida, identificam-se palavras compostas, nomes de pessoas, nome de entidades, siglas e abreviaturas.

No passo seguinte, efetua-se a análise morfológica, classificando-se cada *token* de acordo com sua característica morfológica. As etiquetas geradas nesta etapa são conhecidas como *Part-of-Speech* – PoS. No passo final, o analisador realiza a análise sintática propriamente dita, gerando as construções sintáticas. A Figura 7 apresenta esquematicamente o processo do analisador sintático.

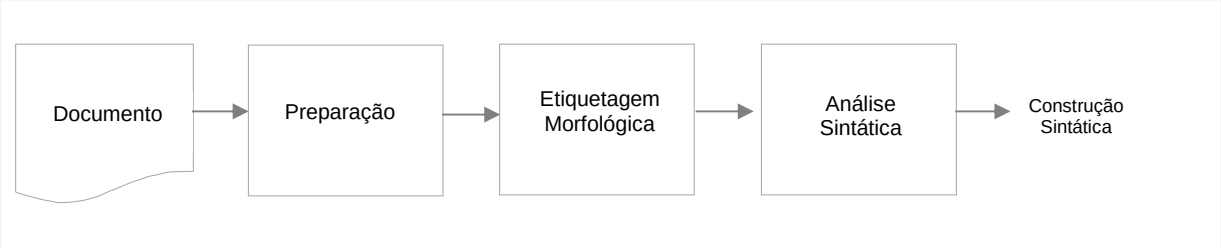


Figura 7 – Processo de Análise Sintática

Fonte: Autor

Cada etapa utiliza informações específicas armazenadas em estruturas de dados preparadas para atender a cada necessidade particular. Assim, a etapa de preparação pode fazer uso de bases de dados de nomes, siglas e abreviaturas. Já na etiquetagem morfológica, é necessária uma estrutura de etiquetas padronizadas, que são usadas na rotulagem dos *tokens* que compõem as sentenças. Por fim, na etapa de análise sintática, são utilizadas estruturas gramaticais que caracterizam as

construções, além de algoritmos criados para filtrar aquelas que são mais adequadas no contexto de análise.

5.2 Preparação

A Figura 8 apresenta o módulo de preparação de textos. Trata-se de um analisador de cadeias de símbolos que divide o texto em sentenças de acordo com a pontuação, abreviaturas e siglas; e as sentenças em *tokens,* considerando o léxico, um dicionário de palavras compostas, nomes próprios, e as abreviaturas e siglas também utilizadas na etapa anterior.

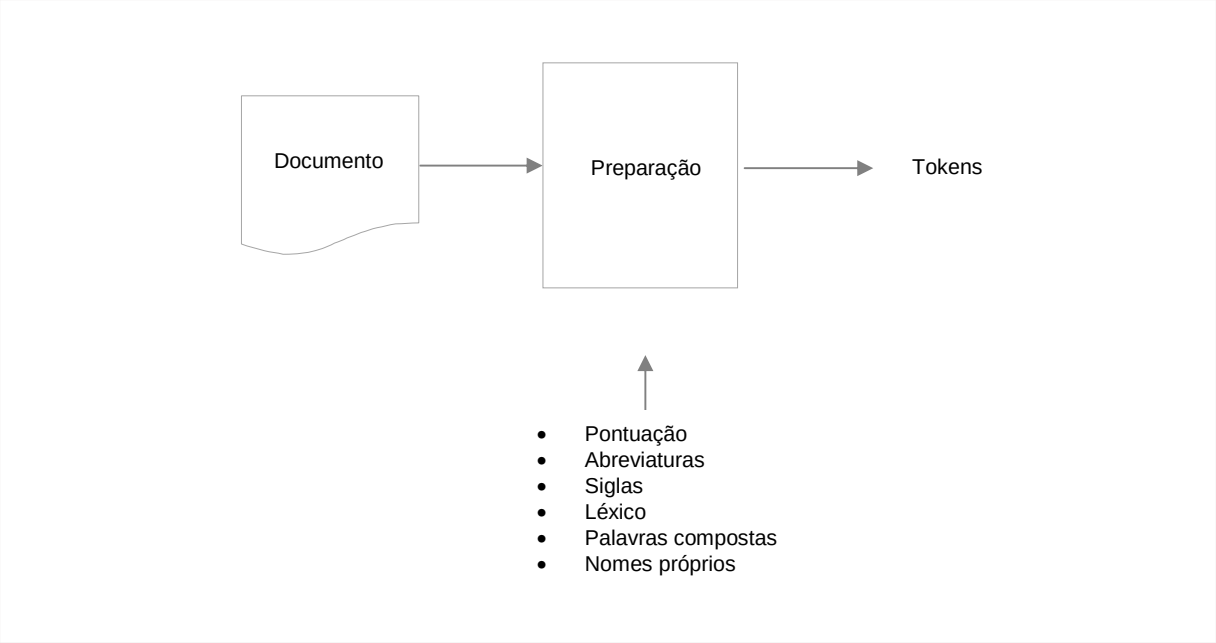


Figura 8 – Módulo de Preparação

Fonte: Autor

O módulo de preparação segue a seguinte sequência de etapas:

a) Inicialmente são identificadas as abreviaturas, as siglas, as palavras compostas e os nomes próprios;

b) Em seguida, o texto é dividido em sentenças, usando como regra de separação os sinais de pontuação seguidos de espaço, com exceção dos termos identificados na etapa anterior;

c) Por fim, identificam-se os *tokens*, considerando os elementos da primeira etapa do processo, e usando como regra de separação, o espaço entre os elementos das sentenças.

Para ilustrar o funcionamento do módulo, tomamos por base as seguintes sentenças:

“O preço de lista nas revendas brasileiras é de US$ 422. Esse equilíbrio era tido como pré-condição para o sucesso do plano econômico.”

De acordo com o processo descrito na primeira etapa, são identificados símbolos e palavras compostas:

“O preço de lista nas revendas brasileiras é de **((US$ 422))** . Esse equilíbrio era tido como **((pré-condição))** para o sucesso do plano econômico.”

No passo seguinte, o texto é dividido em sentenças, levando em consideração a pontuação e o espaçamento:

Sentença 1: “O preço de lista nas revendas brasileiras é de **((US$ 422))**.”

Sentença 2: “Esse equilíbrio era tido como **((pré-condição))** para o sucesso do plano econômico.”

No passo final, cada sentença é dividida em *tokens* a partir do espaçamento dos elementos constituintes de cada sentença:

Tokens [Sentença 1]: [“O”, “preço”, “de”, “lista”, “nas”, “revendas”, “brasileiras”, “é”, “de”, “US$ 422”, “.”]

Tokens [Sentença 2]: [“Esse”, “equilíbrio”, “era”, “tido”, “como”, “pré-condição”, “para”, “o”, “sucesso”, “do”, “plano”, “econômico”, “.”]

Ao final do processo, os *tokens* ficam disponíveis para a etapa seguinte, na qual é realizada a etiquetagem morfológica.

5.3 Etiquetagem morfológica

A Figura 9 apresenta o módulo de etiquetagem morfológica usado no processo de análise sintática. O módulo utiliza algoritmos pré-treinados utilizados em forma de

back-off – quando a etiqueta não é encontrada por um algoritmo, o etiquetador tenta outro, até obter uma classificação.

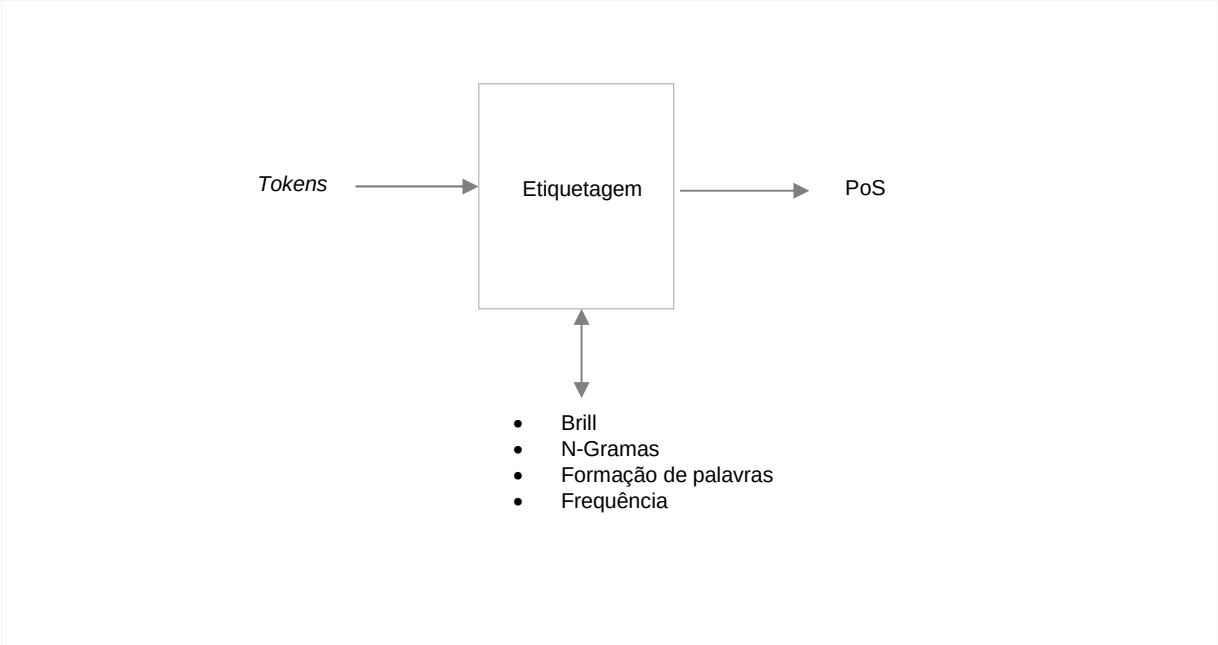


Figura 9 –Módulo de Etiquetagem Morfológica

Fonte: Autor

O módulo de etiquetagem segue a seguinte sequência de etapas:

a) Inicialmente o etiquetador busca pela classificação morfológica dos *tokens*, usando o método transformacional de Brill (1993), inferindo a etiqueta em função dos *tokens* do entorno do *token* analisado e das suas respectivas classificações;

b) Caso não seja encontrada nenhuma etiqueta para o *token em análise,* o etiquetador utiliza a técnica de n-Gramas (JURAFSKY; MARTIN, 2009), inferindo a classificação morfológica em função das últimas n classificações observadas e do *token* analisado;

c) Caso não seja encontrada nenhuma classificação, o etiquetador procura encontrar o *token* e sua respectiva classificação através do gerador de palavras do Linguístico (PADOVANI; NETO, 2017);

d) Caso, ainda assim, a etiqueta não seja encontrada, utiliza-se a classificação de maior frequência do domínio do qual faz parte a sentença analisada.

No primeiro passo, o etiquetador identifica as regras de classificação morfológica por meio de 37 gabaritos aplicados ao corpus de treinamento do domínio do qual faz parte a frase em análise (vide APÊNDICE B), de acordo com a técnica transformacional de Brill (1993).

Os gabaritos levam em consideração os *tokens* do entorno do *token* analisado, as etiquetas com as classificações morfológicas, e combinações de *tokens* e classificações morfológicas. Por exemplo, o gabarito brill.Template(brill.Word([0]), brill.Word([1]), brill.Word([2])) analisa o *token* em referência (brill.word([0]) e os dois *tokens* subsequentes da sentença (brill.word([1])) e brill.word([2])).

Já o gabarito brill.Template(brill.Pos([-1]), brill.Pos([1])) analisa a classificação morfológica do *token* anterior (brill.Pos([-1])) e posterior (brill.Pos([1])) ao *token* de análise. O gabarito brill.Template(brill.Pos([1]), brill.Word([0]), brill.Word([1])) analisa uma combinação composta pela classificação morfológica do *token* posterior ao *token* de análise (brill.Pos([1])), o *token* em análise (brill.Word([0])) e o *token* posterior (brill.Word([1])). O resultado final são regras do tipo Rule('012', “ART”, “P”, [(Word([0]),'a'), (Pos([1]),'V')]), que tem a seguinte interpretação: pela regra 12, substitua a classificação do *token* de artigo “ART” por preposição “P”, sempre que a etiqueta seguinte ao *token* em análise for um verbo “V”.

No segundo passo, o etiquetador utiliza a técnica de n-Gramas (trigramas, bigramas e unigramas) para obter a classificação do *token.* Esta técnica infere a etiqueta do *token* em análise, procurando pela sequência de etiquetas de maior probabilidade de ocorrência para os *tokens* observados. Por exemplo, ao analisar o *token “está”* na sentença “A encomenda está no armazém.”, observa-se que a sequência de etiquetas com maior probabilidade de ocorrência para a sequência de *tokens* “A”, “encomenda”, “está” é “ART”, “N”, “V”, o que faz com que o etiquetador escolha a classificação “V”. As probabilidades são calculadas a partir de um corpus etiquetado do mesmo domínio do qual a sentença faz parte. Como o corpus de treinamento pode não incluir todas as sequências possíveis, aplica-se uma solução de contorno, buscando inicialmente por trigramas, depois bigramas e, por fim, unigramas. No

exemplo anterior, utilizou-se um trigrama. No caso de usar um bigrama, o etiquetador consideraria a sequência de *tokens* “encomenda”, “está”, procurando pela sequência de etiquetas com maior probabilidade de ocorrência, “N”, “V”. No caso de um unigrama, o etiquetador consideraria apenas o *token* “está”, procurando pela etiqueta de maior frequência para a palavra, “V”.

No terceiro passo, o etiquetador usa um algoritmo formador de palavras, com a correspondente classificação morfológica. As palavras são formadas a partir do vocabulário do thesaurus TeP2.0 (DIAS-DA-SILVA; MORAES, 2003), formado por substantivos, adjetivos e verbos, e por um conjunto de regras de prefixação, sufixação e regressão verbal descrito em Basílio (2004). Por exemplo, as palavras “anoitecer”, “pernoitar” e “anoitar” (sinônimo de “anoitecer”) são formadas a partir da combinação dos prefixos “a” e “per”, e os sufixos “ecer” e “ar” ao radical “noit” do substantivo noite. Para reduzir o risco de aceitar derivações inexistentes, são utilizadas as regras que Basilio (2004) destaca como sendo mais prováveis. É o caso da nominalização de verbos com o uso dos sufixos –ção , –mento e – da. Além da formação de novas palavras, o formador de palavras também aplica regras flexão verbal e nominal ao vocabulário do TeP2.0, ampliando a quantidade de palavras por meio algorítmico.

No último passo, denominado “*Default”,* o etiquetador utiliza a classificação morfológica mais frequente do corpus de treinamento do domínio da frase analisada. Por exemplo, se no corpus de treinamento a classificação mais frequente for “N”, substantivo, esta será a etiqueta que o etiquetador selecionará neste passo.

Ao final do processo, as classificações morfológicas dos *tokens* (PoS) ficam disponíveis para a etapa seguinte, na qual é realizada a análise sintática. Um exemplo das saídas geradas pelo módulo de preparação é apresentado no APÊNDICE C.

5.4 Algoritmos

O Algoritmo 1 especifica o funcionamento do etiquetador morfológico. Ele incorpora seis diferentes métodos, sempre com a opção “*backoff*”, que significa que caso o etiquetador não encontre a etiqueta, ele chama outro etiquetador. Por exemplo, se o

método de Brill não retornar a etiqueta, o etiquetador faz uma nova tentativa usando trigramas. Se este método não retornar a etiqueta, então o etiquetador tenta um bigrama, e assim sucessivamente. Regras de transformação, exemplos de etiquetas morfológicas e etiquetas sintáticas são apresentados nos APÊNDICES D, E e F respectivamente.

Algoritmo 1. Etiquetador morfológico

Function *tagger.tag* (token, method)

```
tag ← "empty"
If method is "brill" then
        tag ← brill_tagger.tag(token, backoff)
if method is "trigram" then
        tag ← Trigram_tagger.tag(token, backoff)
if method is "bigram" then
        tag ← Bigram_tagger.tag(token, backoff)
if method is "unigram" then
        tag ← Unigram_tagger.tag(token, backoff)
if method is "linguistico" then
        tag ← Linguistico_tagger.tag(token, backoff)
if method is "default" then
        tag ← Default_tagger.tag(token, backoff)
return tag
```

end function

Ressalta-se que é possível expandir o etiquetador com a inclusão de outras técnicas, desde que seja utilizado o padrão de etiquetas do LX-Gram (BRANCO; COSTA, 2014).

6 CONSTRUÇÕES ESTÁTICAS

Este capítulo apresenta o módulo para analisar os textos por meio de construções estáticas, ou seja, estruturas prontas que representam construções gramaticais.

6.1 Processo

A análise sintática inicia-se com o processamento dos *tokens* da sentença etiquetados morfologicamente (*PoS*). O Analisador de Contexto identifica as restrições do contexto analisado e o Seletor de Construções filtra as estruturas adequadas. Ao final, o Analisador Sintático gera a construção sintática ou uma indicação de que não foi possível encontrar nenhuma construção. O processo é apresentado em detalhes nas seções seguintes.

6.2 Análise sintática

A Figura 10 apresenta a estrutura do Analisador Sintático de Construções Estáticas. Ele é composto por um Analisador de Contexto, um Seletor de Construções e um Transdutor. A análise sintática é realizada da seguinte maneira:

a) O Analisador de Contexto instancia os parâmetros usados para identificar o contexto de análise: padrão de construções, domínio da informação, quantidade limite de *tokens* da sentença, técnica de etiquetagem, uso de função de similaridade e estatística;

b) O Seletor de Construções identifica as sentenças que possuem número de *tokens* maior do que o limite definido em parâmetro, e as descarta, prosseguindo com as demais;

c) O Seletor de Construções faz uma filtragem inicial, trazendo de uma base de dados previamente preparada durante a fase de treinamento, apenas as construções do domínio da informação que possuem o mesmo número de *tokens* da sentença analisada;

d) Em seguida, o Seletor de Construções procura por construções que se adéquem à sequência de etiquetas dos *tokens,* de acordo com a estrutura definida pelo padrão de construções. Caso o Seletor de Construções não encontre nenhuma construção, ele solicita ao Etiquetador Morfológico nova

etiquetagem de acordo com as técnicas de etiquetagem definidas em parâmetro;

e) Caso o Seletor de Construções encontre mais de uma construção, ele aplica o critério estatístico definido em parâmetro para selecionar a melhor, retornando a estrutura escolhida ao final do processo;

f) Caso o Seletor de Construções não encontre nenhuma construção após tentar todas as técnicas de etiquetagem, ele pode tentar buscar uma construção similar, se esta opção estiver definida em parâmetro;

g) Ao final, o Transdutor Sintático recebe a estrutura escolhida e substitui a palavra chave “token” pelos *tokens* da sentença analisada.

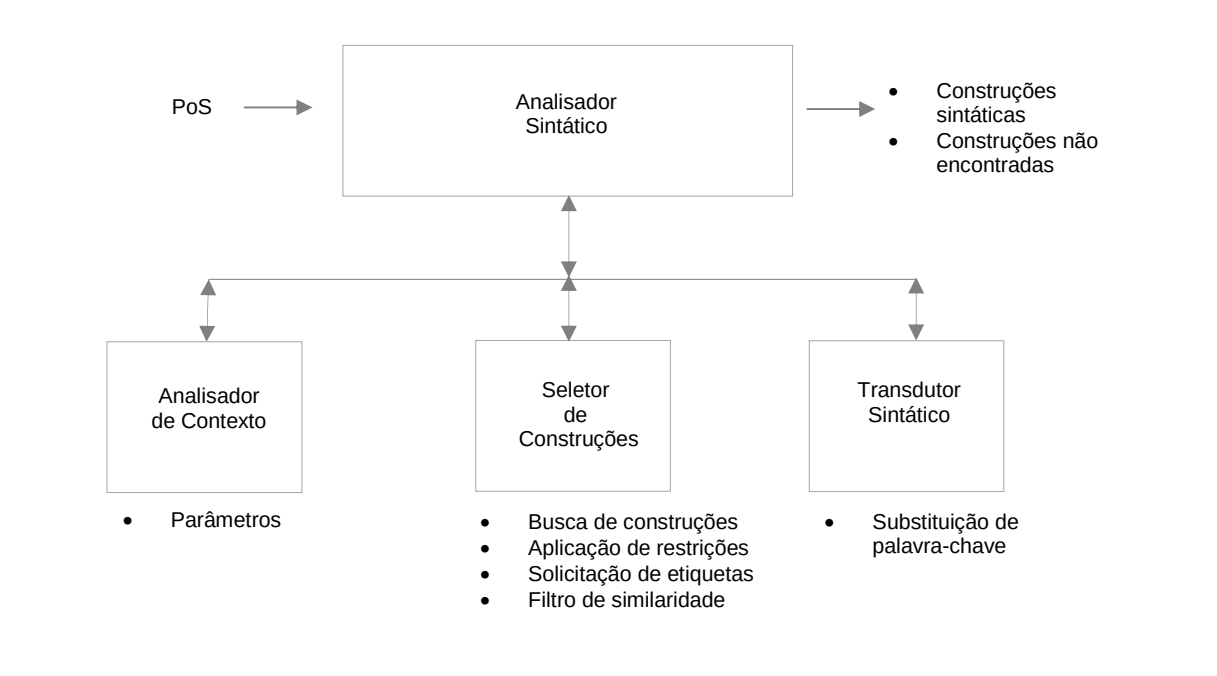


Figura 10 – Analisador Sintático por Construções Estáticas

Fonte: Autor

As construções usadas pelo Analisador Sintático são compostas por uma parte fixa e outra variável. A parte fixa representa as árvores de derivação. Já a parte variável é preenchida pelos *tokens* provenientes da sentença analisada. A palavra chave “token” é usada para indicar os possíveis espaços de substituição. O exemplo abaixo ilustra uma construção utilizada pelo Analisador Sintático.

(S (S (V token) (NP (CARD token) (N token))) (PNT token))

Analisando o exemplo, vemos que se trata de uma construção que permite quatro substituições. A primeira substituição deve ser feita por um *token* cuja classificação morfológica é um verbo; a segunda substituição deve ser feita por um *token* cuja classificação morfológica é um numeral cardinal; a terceira substituição deve ser feita por um *token* cuja classificação morfológica é um substantivo e a quarta, por um *token* cuja classificação morfológica é um ponto final. De acordo com este paradigma, a análise sintática passa a ser um processo de busca de construções que se encaixem na sequência de classificações morfológicas dos *tokens* das sentenças analisadas. Ao analisar, por exemplo, a sentença “Desapareceram dois carros.”, o Analisador Sintático vai selecionar inicialmente as construções que permitem quatro possíveis substituições. Deste conjunto, ele filtra as construções que se adéquam ao padrão de substituição requerido pela sequência de etiquetas, no caso, [(V token), (V Desapareceram)], [(CARD token), (CARD dois)], [(N token), (N carros)], [(PNT token), (PNT .)]

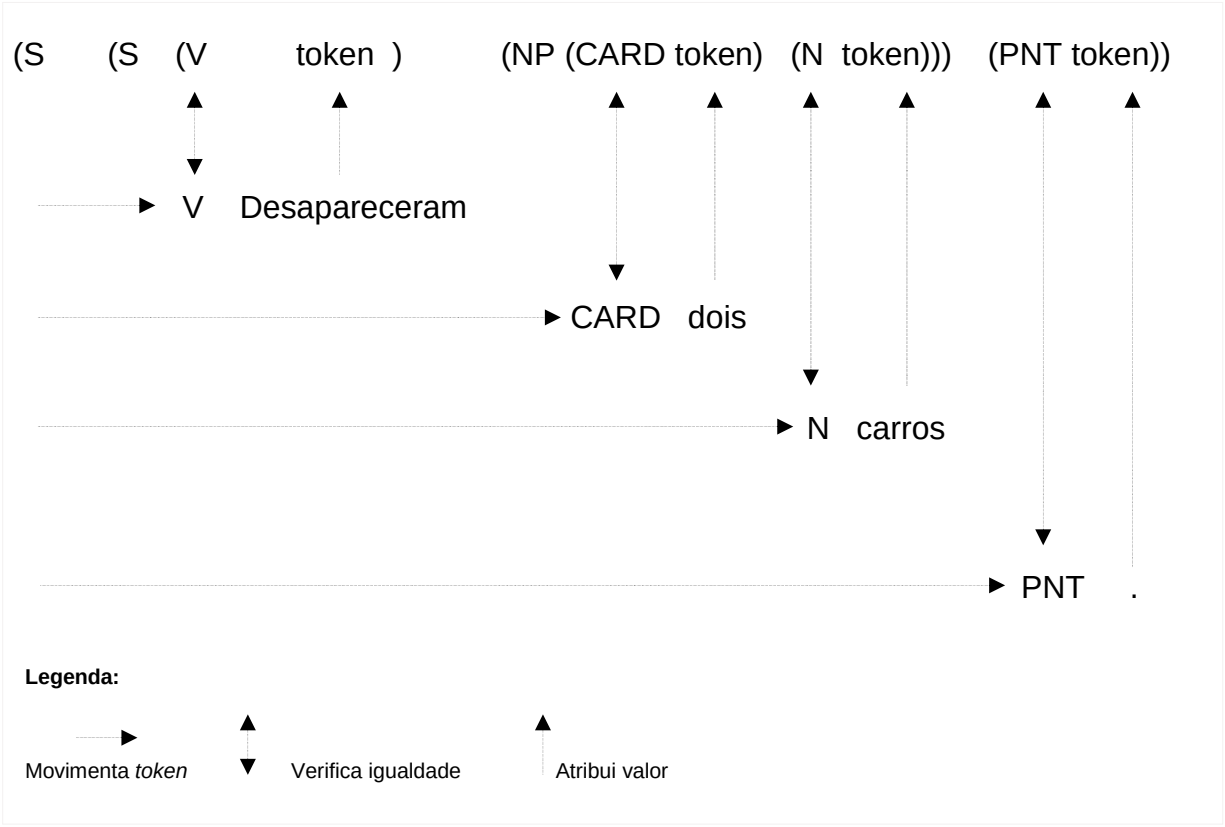


Figura 11 – Exemplo de Análise Sintática

Fonte: Autor

As substituições são realizadas *token* a *token*, de forma que a parte fixa permaneça inalterada. Desta forma, o Analisador Sintático gera a construção (S (S (V Desapareceram) (NP (CARD dois) (N carros))) (PNT .)). A Figura 11 ilustra visualmente este exemplo. As estruturas que não atendem aos critérios estabelecidos são descartadas pelo Seletor de Construções. No exemplo apresentado, apenas construções que permitem quatro substituições são selecionadas. Além disso, as construções que não atendem à sequência de etiquetas da sentença também são desprezadas. Caso, ao final, nenhuma construção seja escolhida, o Analisador Sintático gera uma cadeia de caracteres com uma mensagem padronizada, para indicar que nenhuma construção foi selecionada. A Figura 12 ilustra visualmente o cenário de uma construção descartada.

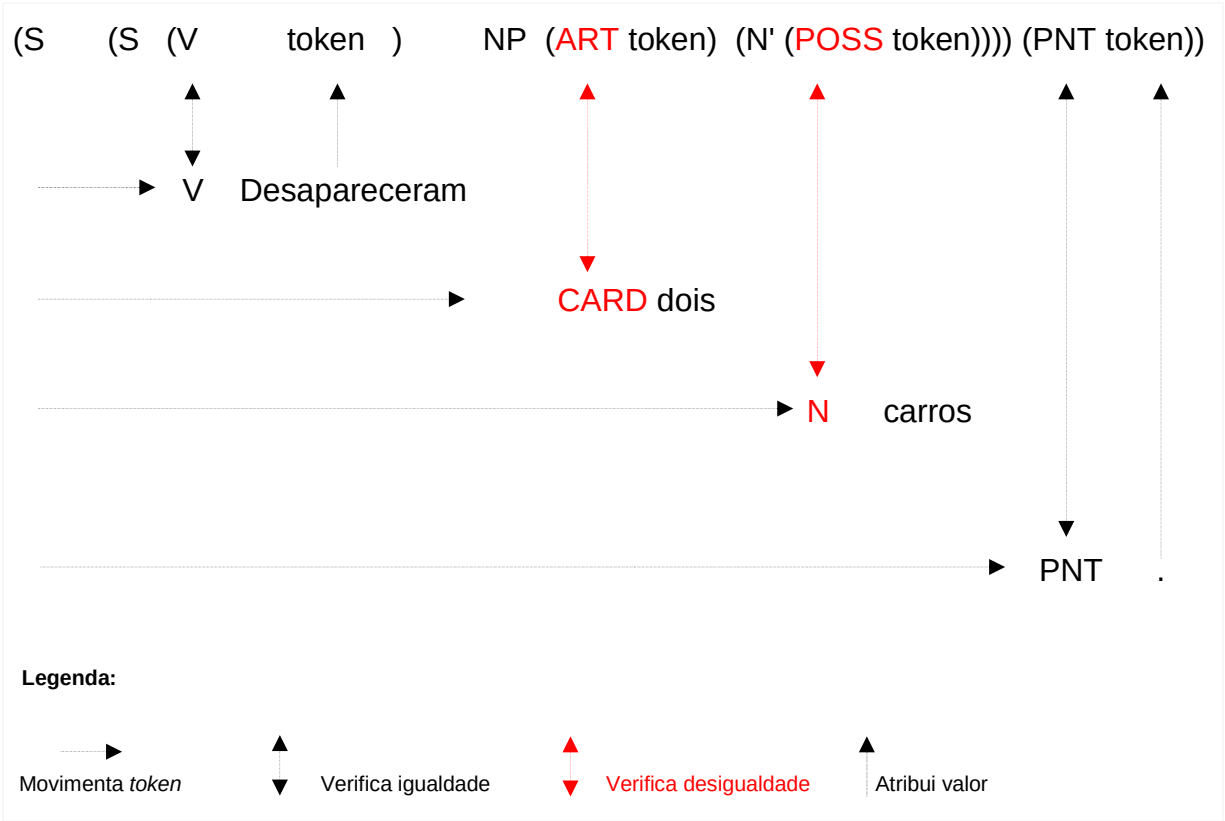


Figura 12 – Exemplo de Construção Descartada

Fonte: Autor

Esta forma de analisar sintaticamente as sentenças é flexível no sentido que permitir o uso de diversos tipos de construções, não apenas as construções abstratas. É o

caso de construções para identificação de idiomatismos formais. Por exemplo, a expressão “dar uma de desentendido” pode ser modelada pela construção “dar uma de ADJ”. Diferentemente das construções abstratas, os idiomatismos formais combinam *tokens* com símbolos; no entanto, o padrão de busca se mantém, com o Analisador Sintático selecionando as construções com o mesmo número de *tokens* de substituição e realizando as trocas em função das etiquetas morfológicas. A Figura 13 ilustra visualmente este cenário.

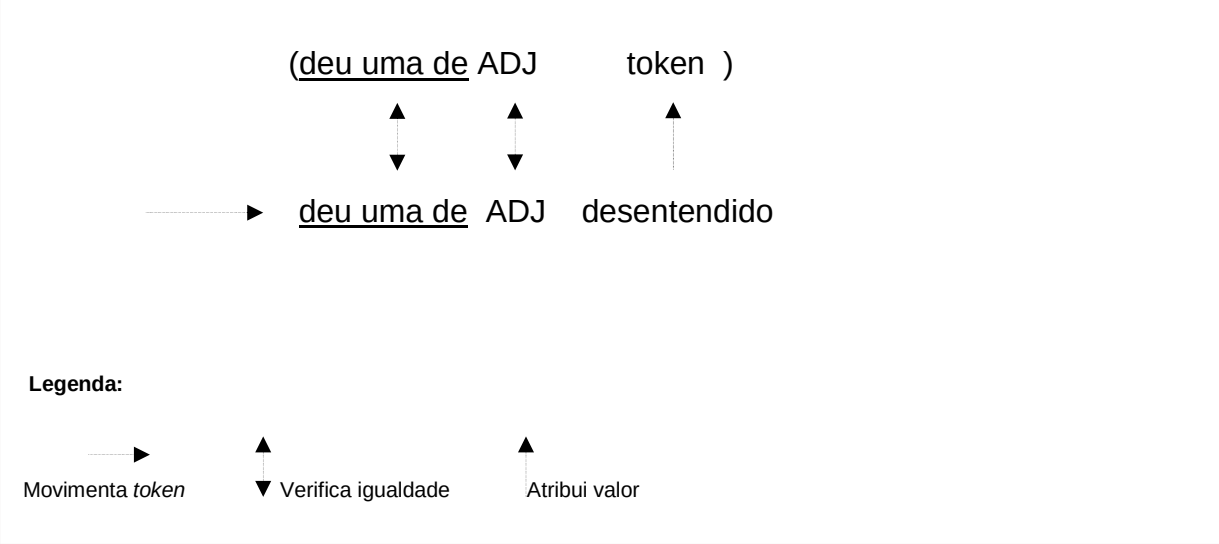


Figura 13 – Exemplo de Idiomatismo Formal

Fonte: Autor

O Etiquetador Morfológico pode ser acionado quando o Seletor de Construções não encontrar nenhuma construção para a sentença analisada. Neste caso, o Etiquetador Morfológico muda o contexto morfossintático, utilizando outra técnica de etiquetagem morfológica e informando ao Seletor de Construções, que realiza uma nova busca a partir das novas etiquetas geradas. O Analisador de Contexto também incorpora uma função de similaridade, através da qual o Seletor de Construções busca por construções com etiquetas similares, mas não iguais àquelas informadas pelo Etiquetador Morfológico. Esta função tem por objetivo corrigir eventuais erros provenientes da classificação morfológica, evitando que o Seletor de Construções deixe de encontrar a construção da sentença analisada. Por exemplo, no caso da sentença “O cliente encomendou um computador baratíssimo”, é esperado que o Etiquetador Morfológico informe a seguinte sequência de etiquetas:

(ART O) (N cliente) (V encomendou) (ART um) (N computador) (A baratíssimo) (PNT .)

Para a qual seria selecionada a seguinte construção:

(S (S (NP (ART token) (N token)) (VP (V token) (NP (ART token) (N' (N token) (A token))))) (PNT token))

que, uma vez preenchida, ficaria da seguinte forma:

(S (S (NP (ART o) (N cliente)) (VP (V encomendou) (NP (ART um) (N' (N computador) (A baratíssimo))))) (PNT .))

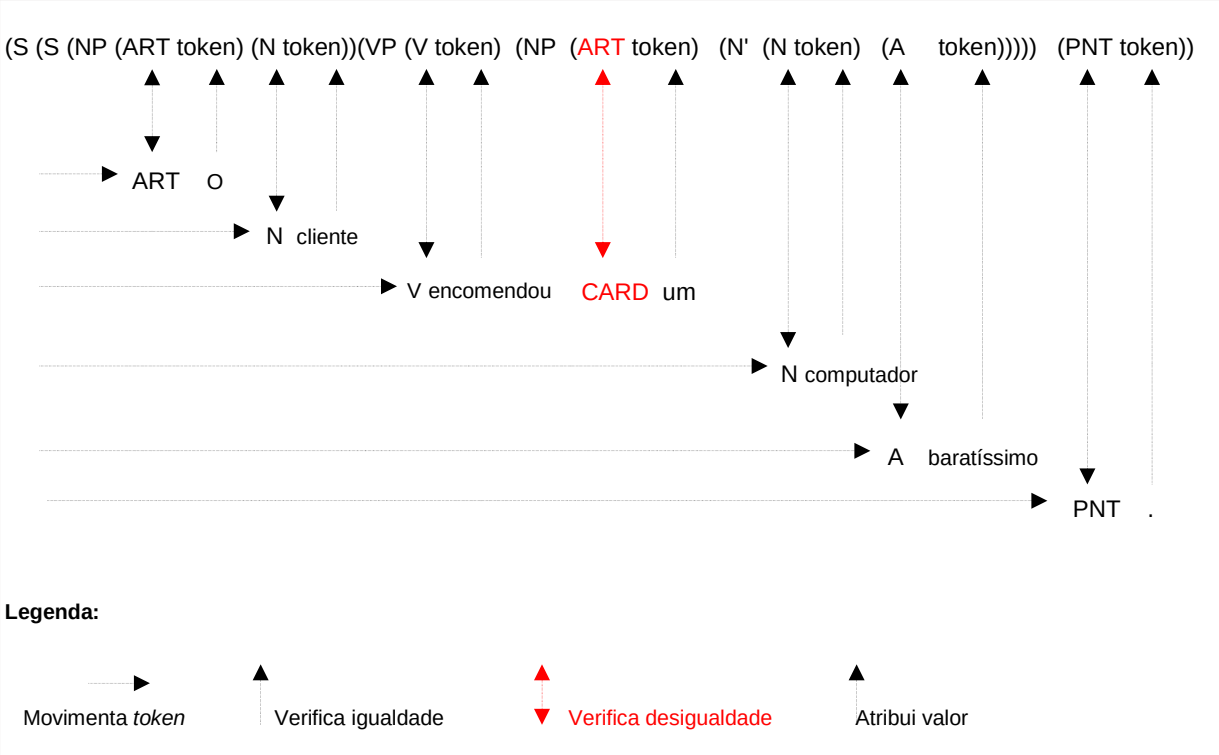


Figura 14 – Análise de Similaridade

Fonte: Autor

No entanto, caso o Etiquetador Morfológico informe a sequência de etiquetas (ART O) (N cliente) (V encomendou) (CARD um) (N computador) (A baratíssimo), ele poderá levar o Seletor de Construções a não encontrar nenhuma construção, pois o artigo “um” foi classificado como numeral cardinal. Neste caso, o Analisador de Contexto pode acionar a função de similaridade para selecionar a construção mais próxima da sequência de etiquetas recebida, corrigindo o erro proveniente do Etiquetador Morfológico. A Figura 14 ilustra visualmente este cenário.

O Analisador Sintático também pode utilizar critérios estatísticos para desambiguação de construções, no caso de haver mais de uma construção que se encaixe na sequência de etiquetas morfológicas da sentença analisada. O Analisador de Contexto informa ao Seletor de Construções as características estatísticas de cada construção e este as utiliza no processo de desambiguação. É o que acontece, por exemplo, quando é selecionada a construção de maior frequência no corpus de treinamento do domínio da sentença analisada.

A inversão de estruturas gramaticais também pode ser representada através de construções. É o caso da Voz Passiva. A sentença “A reunião foi interrompida” é representada pela construção apresentada na Figura 15.

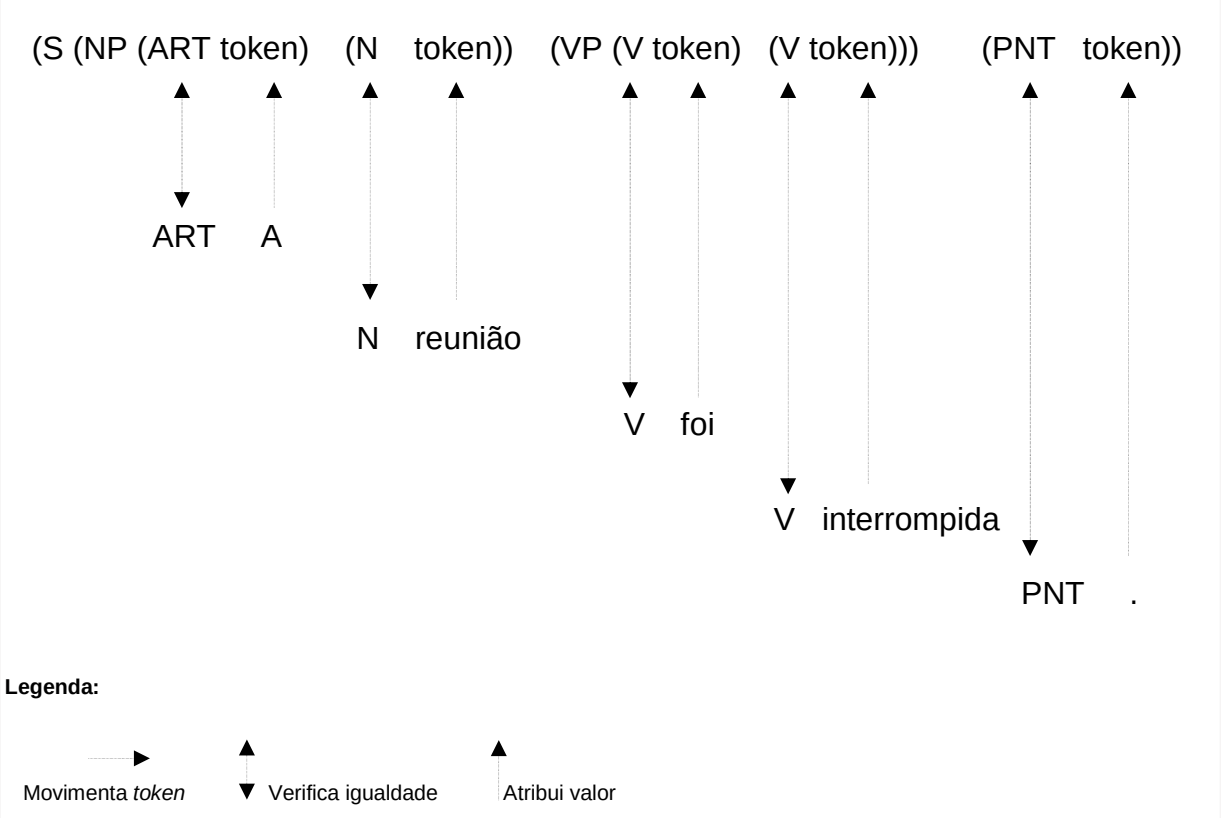


Figura 15 – Voz passiva

Fonte: Autor

6.3 Autômato adaptativo

A Análise Sintática é coordenada por meio de um autômato adaptativo inicialmente configurado com um único estado e seis possíveis transições, que permitem

automodificações em função do domínio analisado, quantidade de *tokens,* técnica de etiquetagem*,* similaridade, estatística e padrões de construções. Os símbolos π, δ, ρ, τ, σ e θ são utilizados para representar uma escolha entre várias opções, de forma que as transições sejam realizadas para ficarem aderentes a estas escolhas. Embora a notação seja semelhante a de um autômato finito, neste caso, cada transição adaptativa implica a sua autoeliminação do diagrama assim que ela é executada. A Figura 16 apresenta a configuração inicial do autômato.

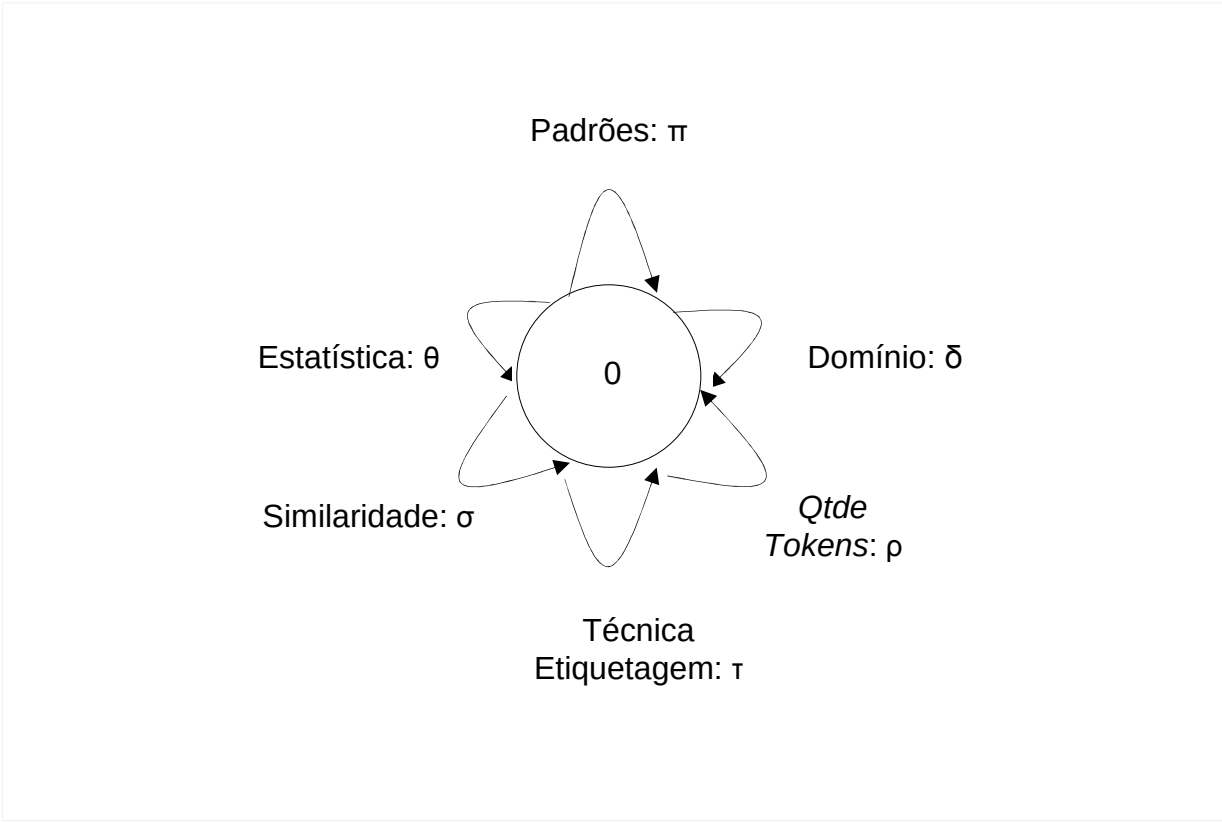


Figura 16 – Analisador Sintático – Configuração Inicial

Fonte: Autor

A característica adaptativa do autômato faz com que ele se automodifique dinamicamente em função do contexto de análise, evitando, com isso, o carregamento desnecessário de todos os possíveis estados e transições. A Figura 17 apresenta a configuração do autômato após a definição do domínio de análise D.

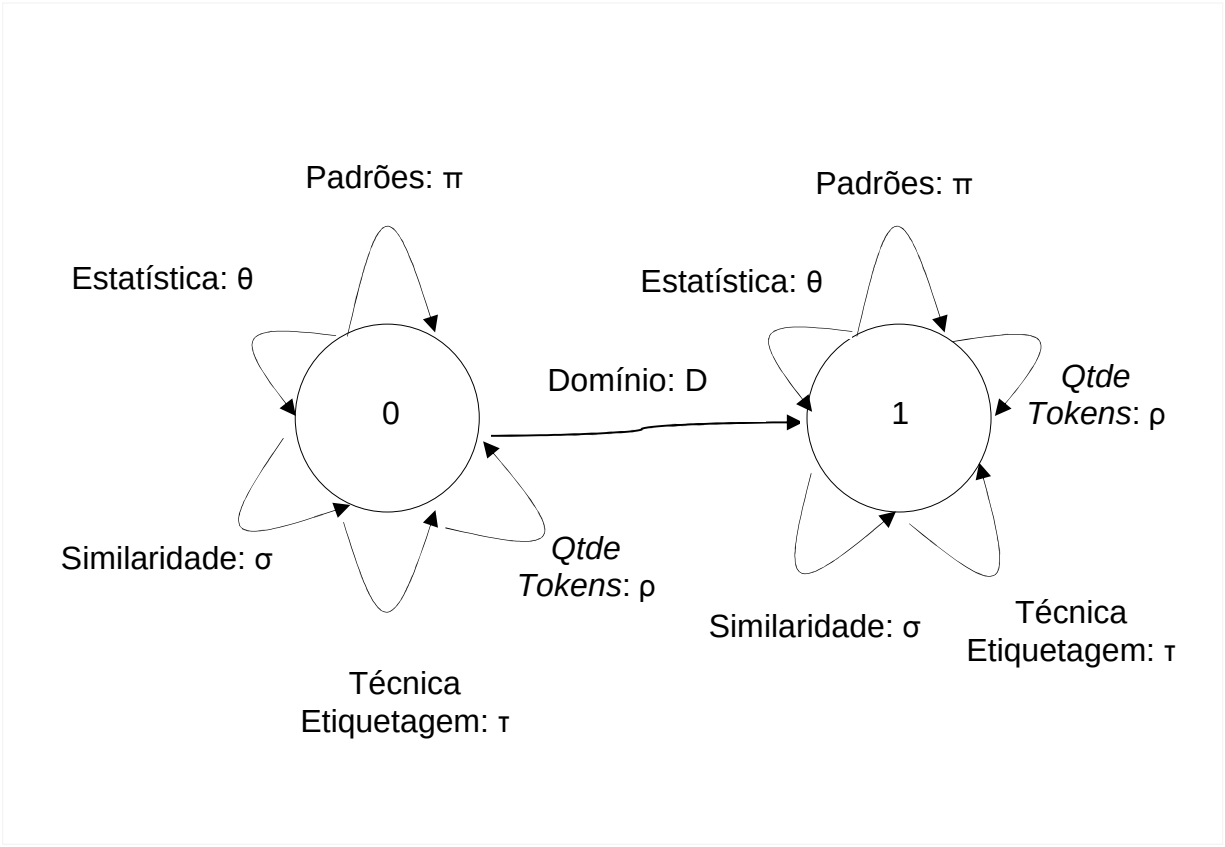


Figura 17 – Analisador Sintático - Após a identificação do domínio

Fonte: Autor

As mudanças de configuração do autômato são realizadas através das seguintes funções adaptativas:

a) α (k): { l*: - [(k, δ): → k] + [(k, D): → l, β (l)] }

b) β (l): { + [(l, π): → l] + [(l, ρ): → l] + [(l, τ): → l] + [(l, σ): → l] + [(l, θ): → l] }

onde:

α (k) e β (l) são funções executadas respectivamente antes e após o consumo do indicador de domínio D;
k indica o estado 0;
l* indica que o estado l deve ser criado;
l indica o estado 1;
π, ρ, τ, σ e θ indicam as transições para o novo estado l.

Portanto, a função α (k) aplicada ao estado k=0 cria o estado l=1, elimina a transição (0, δ): → 0 e cria a transição (0, D): → 1. Em seguida, a função adaptativa β (l) é chamada com parâmetro l=1, criando as transições
(1, π): → 1, (1, ρ): → 1, (1, τ): → 1, (1, σ): → 1 e (1, θ): → 1.

Na etapa seguinte, identifica-se a quantidade de *tokens* limite das sentenças analisadas. A Figura 18 apresenta a configuração do autômato quando esta passagem é realizada.

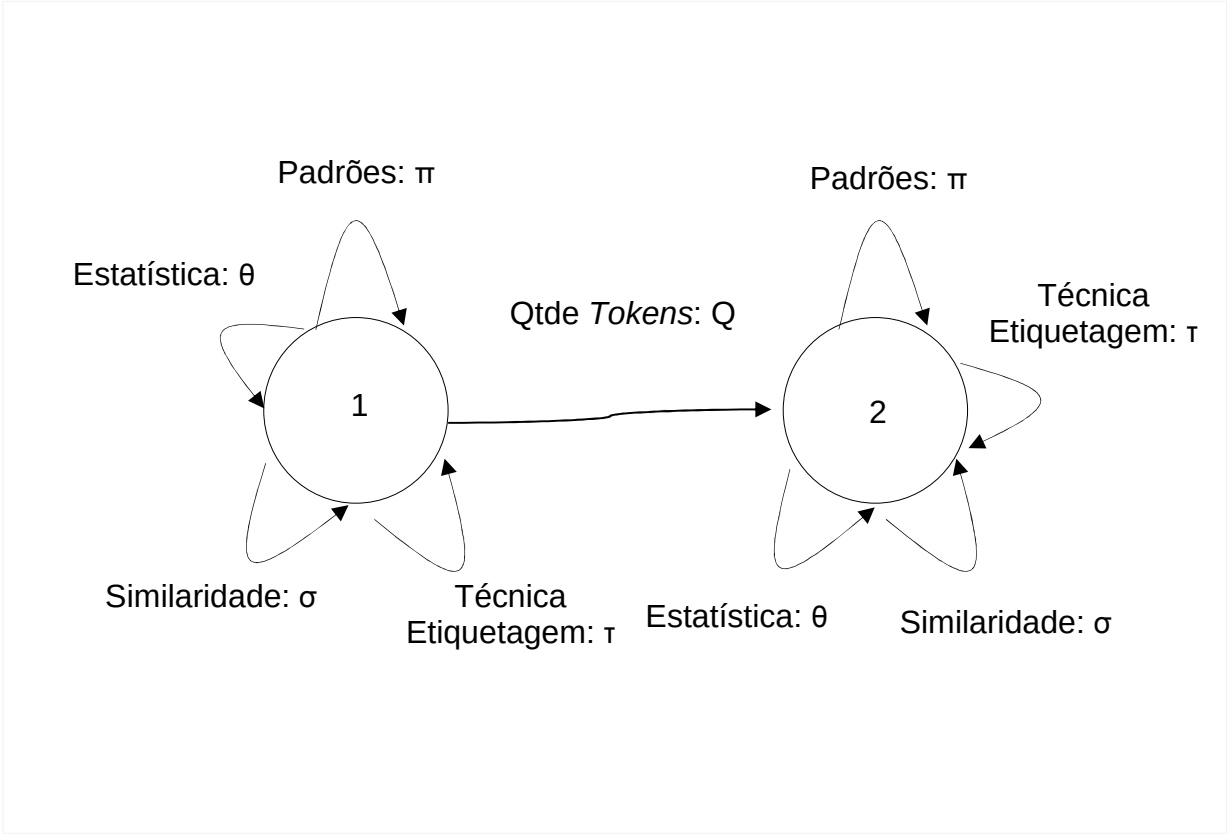


Figura 18 – Analisador Sintático - Após a identificação da quantidade de tokens
Fonte: Autor

a) α (l): { m*: - [(l, ρ): → l] + [(l, Q): → m, β (m)] }

b) β (m): { + [(m, π): → m] + [(m, τ): → m] + [(m, σ): → m] + [(m, θ): → m] }

onde:

α (l) e β (m) são funções executadas respectivamente antes e após o consumo do indicador da quantidade de *tokens* Q;
l indica o estado 1;
m* indica que o estado m deve ser criado;
m indica o estado 2;
π, τ, σ e θ indicam as transições para o novo estado m.

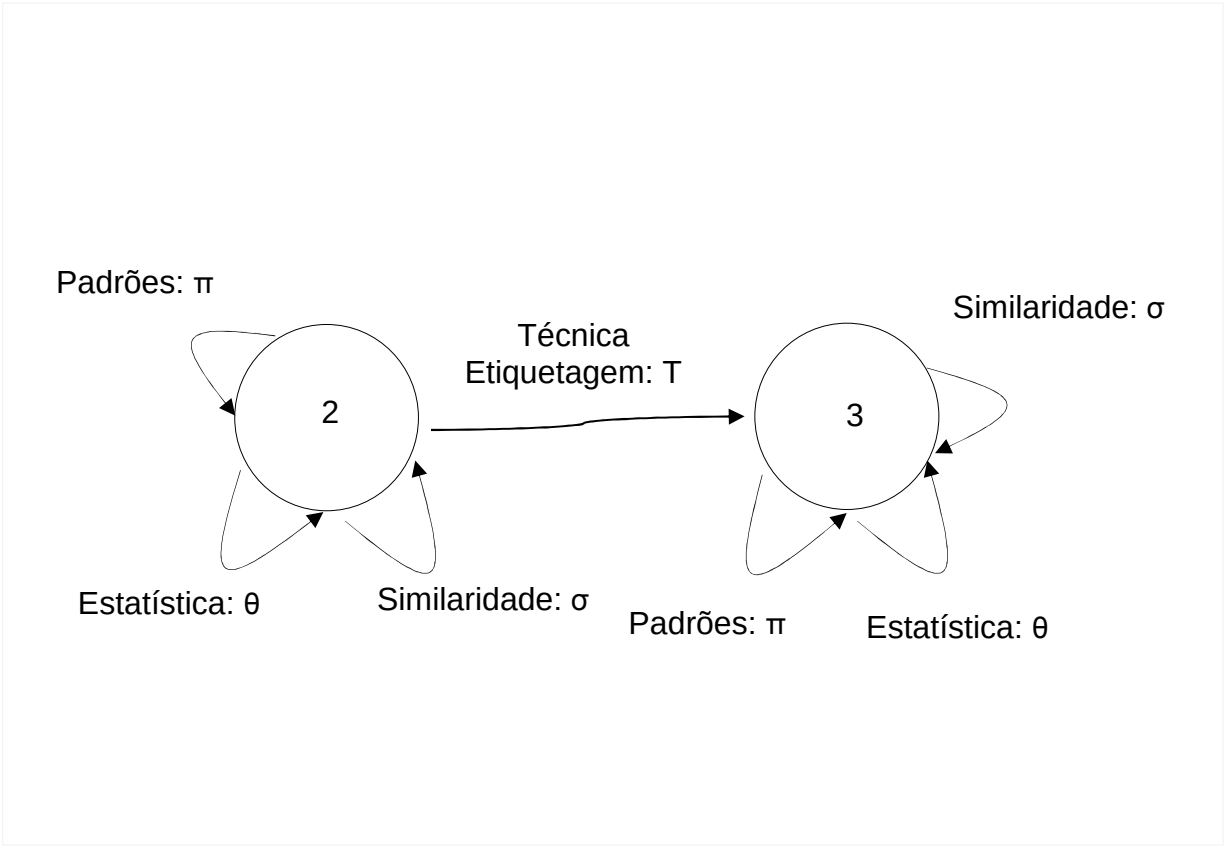


Figura 19 – Analisador Sintático - Após a identificação da técnica de etiquetagem
Fonte: Autor

A função α (l) aplicada ao estado l=1 cria o estado m=2, elimina a transição (1, ρ): → 1 e cria a transição (1, Q): → 2. Em seguida, a função adaptativa β (m) é chamada com parâmetro m=2, criando as transições
(2, π): → 2, (2, τ): → 2, (2, σ): → 2 e (2, θ): → 2.

Em seguida, é identificada a técnica de etiquetagem. A Figura 19 ilustra esta passagem. Neste passo, as funções adaptativas são definidas da seguinte forma:

a) α (n): { o*: - [(n, τ): → n] + [(n, T): → o, β (o)] }

b) β (o): { + [(o, π): → o] + [(o, σ): → o] + [(o, θ): → o] }

onde:

α (n) e β (o) são funções executadas respectivamente antes e após o consumo do indicador da técnica de etiquetagem T;
n indica o estado 2;
o* indica que o estado o deve ser criado;
o indica o estado 3;
π, σ e θ indicam as transições para o novo estado o.

A função α (n) aplicada ao estado n=2 cria o estado o=3, elimina a transição (2, τ) : → 2 e cria a transição (2, T): → 3. Em seguida, a função adaptativa β (o) é chamada com parâmetro o=3, criando as transições (3, π): → 3, (3, σ): → 3 e (3, θ): → 3.

No passo seguinte, é identificado o uso de similaridade. A Figura 20 ilustra esta etapa. Novamente, são acionadas as funções adaptativas do autômato. Neste caso, com as seguintes definições:

a) α (o): { p*: - [(o, σ): → o] + [(o, S): → p, β (p)] }

b) β (p): { + [(p, π): → p] + [(p, θ): → p] }

onde:

α (o) e β (p) são funções executadas respectivamente antes e após o consumo do indicador do uso de similaridade S;
o indica o estado 3;
p* indica que o estado p deve ser criado;
p indica o estado 4;
π e θ indicam as transições para o novo estado n.

A função α (o) aplicada ao estado o=3 cria o estado p=4, elimina a transição (3, σ) e cria a transição (3, S): → 4. Em seguida, a função adaptativa β (p) é chamada com parâmetro p=4, criando as transições (4, π): → 4 e (4, θ): → 4.

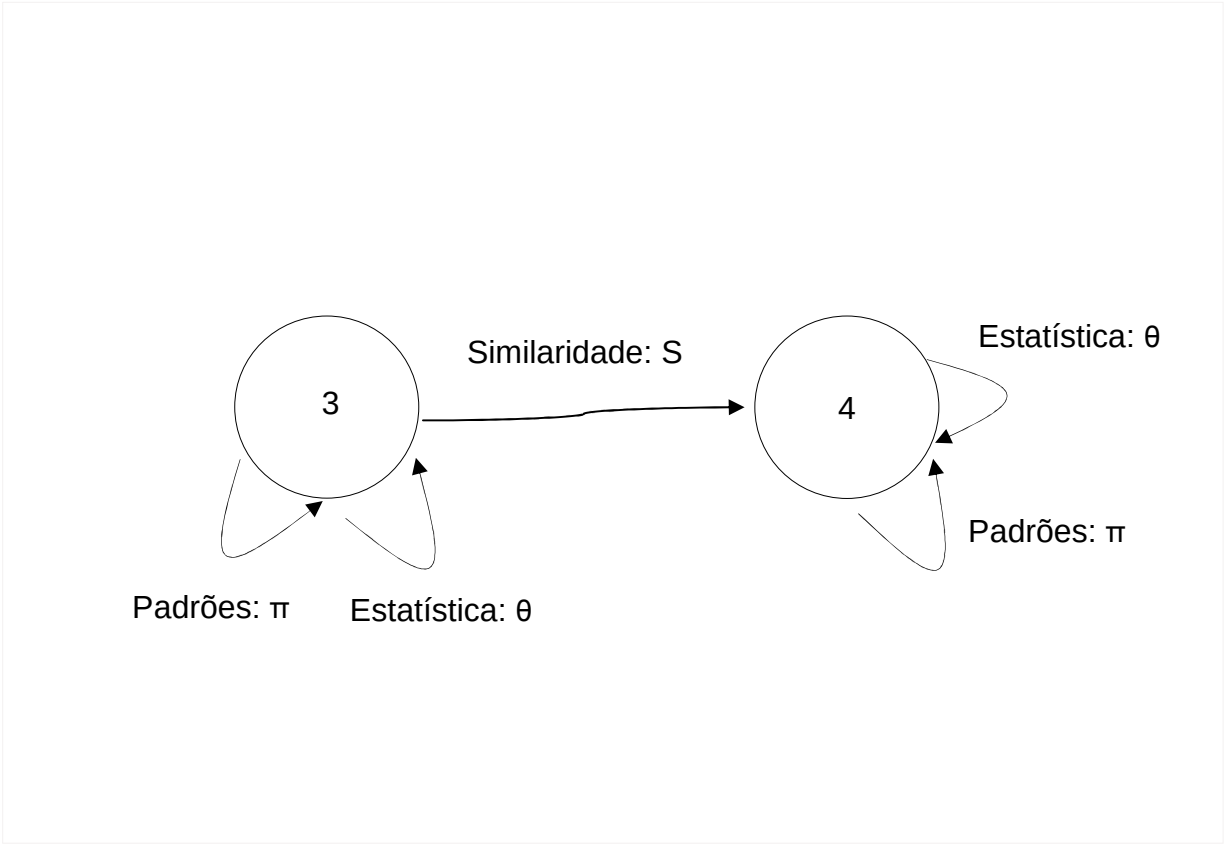


Figura 20 – Analisador Sintático - Após a identificação da técnica de similaridade

Fonte: Autor

A Figura 21 apresenta a transição para consumo do indicador do uso de estatística. De amaneira similar aos passos anteriores, as funções adaptativas são redefinidas para permitir a modificação do autômato, ficando da seguinte forma:

a) α (p): { q*: - [(p, θ): → p] + [(p, E): → q, β (q)] }

b) β (q): { + [(q, π): → q] }

onde:

α (p) e β (q) são funções executadas respectivamente antes e após o consumo do indicador do uso de estatística E;

p indica o estado 4;

q* indica que o estado q deve ser criado;

q indica o estado 5;

π indica as transições para o novo estado q.

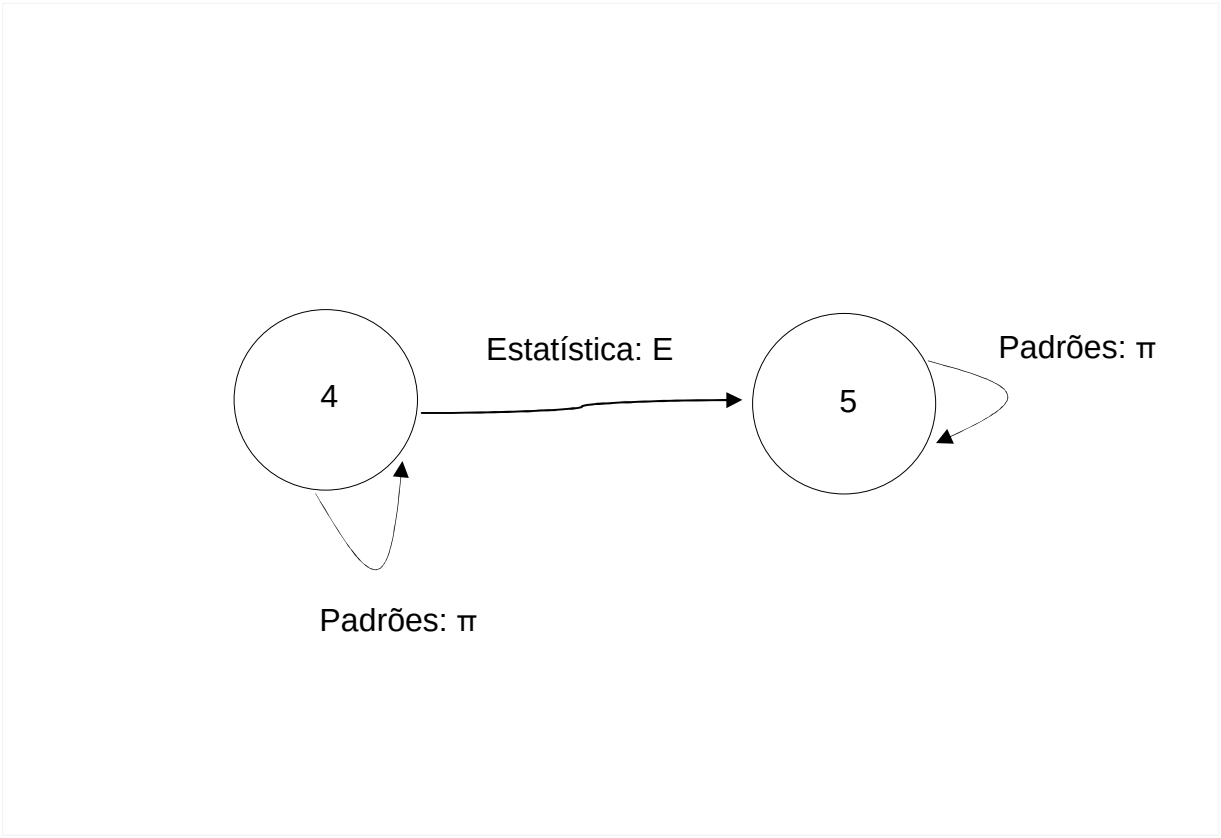


Figura 21 – Analisador Sintático - Após a identificação da variável estatística

Fonte: Autor

A função α (p) aplicada ao estado p=4 cria o estado q=5, elimina a transição (4, θ): → 4, e cria a transição (4, E): → 5. Em seguida, a função adaptativa β (q) é chamada com parâmetro q=5, criando as transições (5, π): → 5.

O último passo acontece quando o padrão de construção é identificado. Neste caso, o autômato adaptativo cria um estado para cada *token* da sentença e as transições correspondentes, visando processar as etiquetas morfológicas informadas pelo analisador morfológico.

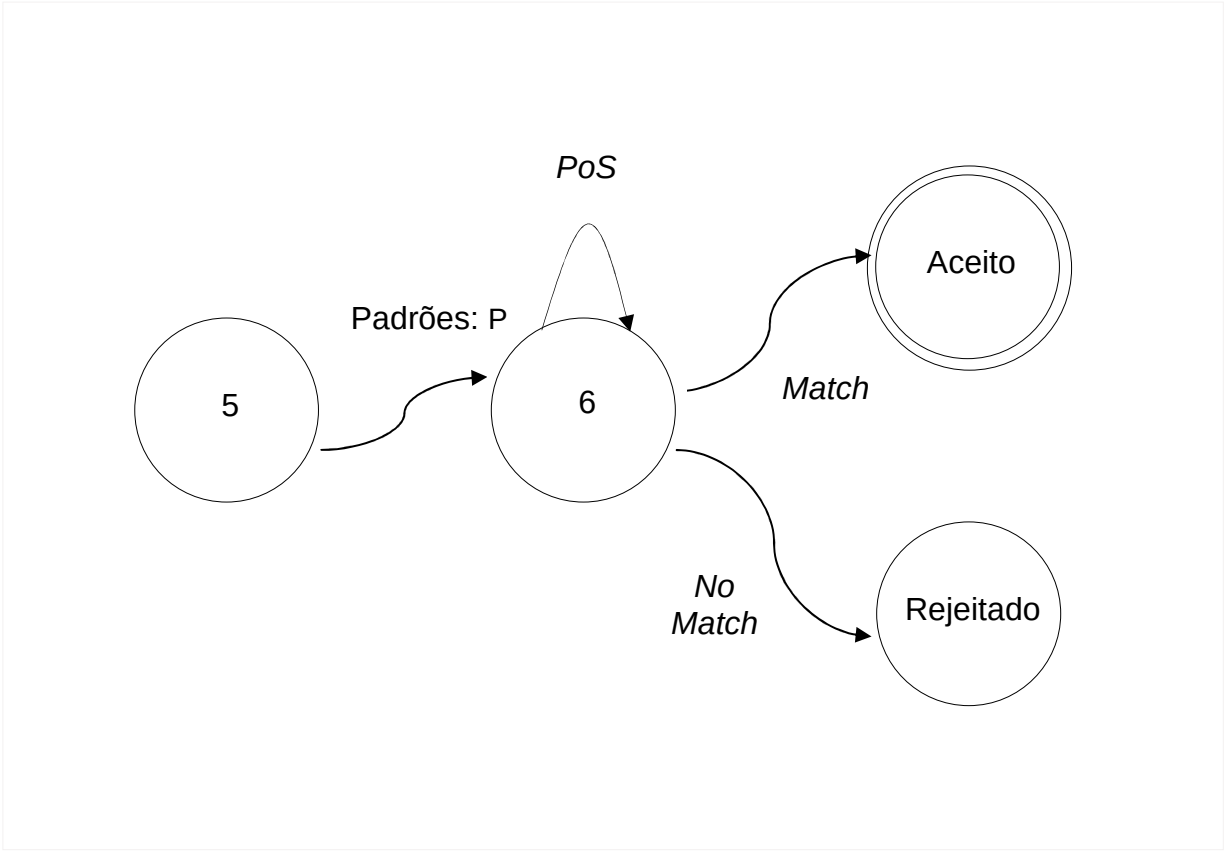


Figura 22 – Analisador Sintático - Após a identificação do padrão P

Fonte: Autor

As funções adaptativas ficam da seguinte forma:

a) α (q): { r*: - [(q, π): → q] + [(q, P): → r, β (r)] }

b) β (r): { s*, t*: + [(r, PoS): → r] + [(r, Match): → “Aceito”] + [(r, NotMatch): → “Rejeitado”] }

onde:

α (q) e β (r) são funções executadas respectivamente antes e após o consumo do indicador do padrão P;

q indica o estado 5;

r* indica que o estado r deve ser criado;

r indica o estado 6;

s* indica que o estado s deve ser criado;

s indica o estado “Aceito”;

t* indica que o estado t deve ser criado;
t indica o estado “Rejeitado”;
Match indica a transição para o novo estado “Aceito”;
NoMatch indica a transição para o novo estado “Rejeitado”.

A função α (q) aplicada ao estado q=5 cria o estado r=6, elimina a transição (5, π): → 5 e cria a transição (5, P): → 6. Em seguida, a função adaptativa β (r) é chamada com parâmetro r=6, criando os estados s= ”Aceito” e t = “Rejeitado”, e as transições (6, PoS): → 6, (6, Match): → “Aceito” e (6, NotMatch): → “Rejeitado”. A Figura 22 ilustra esta última configuração.

De acordo com a definição de Neto (2001), o mecanismo de funcionamento do dispositivo adaptativo básico prevê a aplicação de regras não adaptativas em um determinado momento. É o que se observa no passo 6, quando o padrão de reconhecimento está definido e os *tokens* da sentença são processados de acordo com um autômato não adaptativo.

6.4 Algoritmos

O Algoritmo 2 é o módulo principal do analisador sintático. Ele recebe as sentenças como parâmetro inicial, identifica as configurações que serão usadas na análise sintática, cria os padrões para cada elemento da sentença, e busca as construções correspondentes. Ao final, substitui a palavra chave 'token' pelos elementos da sentença.

Algoritmo 2. Módulo principal do Analisador Sintático

```
Function adaptDevice.SyntacticAnalysis (sentences)
    matrixParse ← "empty"
    currentConfiguration ← adaptFunc.setConfiguration()
    for each element in sentences do
        sentencePattern ← adaptdevice.CreatePattern(element)
        sentenceConstruction ←
          adaptDevice.SearchConstructions(sentencePattern)
        sentenceParse ←
          adaptDevice.ReplaceKeyword(sentenceConstruction, tokens)
        matrixParse ← matrixParse + sentenceParse
    return matrixParse
end function
```

O Algoritmo 3 é responsável por definir as configurações de contexto e atribuir as variáveis correspondentes. Já o Algoritmo 4 é responsável por identificar as configurações que serão usadas na análise.

Algoritmo 3. Define as configurações de contexto

Function *adaptFunc.setConfiguration* ()

```
currentConfig ← initNonAdaptiveConfig
if adaptFunc.searchContextConfig()  is not empty then
        currentConfig ← adaptFunc. searchContextConfig ()
end if
return currentConfig
```

end function

Algoritmo 4. Busca configurações de contexto

Function *adaptFunc.searchContextConfig* ()

```
contextConfig ← initAdaptiveConfig
currentConfig ← contextConfig.Pattern + contextConfig.Domain +
contextConfig.NumberTokens + contextConfig.Labeling +
contextConfig.Similarity + contextConfig.Statistics
return currentConfig
```

end function

O Algoritmo 5 monta o padrão de busca de construções a partir da sequência das etiquetas PoS dos *tokens* da sentença. Esse padrão é usado pelo Algoritmo 6 para identificar as construções capazes de atendê-lo. O Algoritmo 7 é usado pelo analisador ao final do processo, visando substituir a palavra chave ‘token’, presente na construção selecionada, pelos *tokens* que formam a sentença,

Algoritmo 5. Monta padrão a partir das etiquetas PoS

```
Function adaptdevice.CreatePattern (sentence)
    pattern ← "empty"
    output  ← "empty"
    PoS ←  "empty"
    tokens ← sentence.tokenize
    for each element in tokens do
        PoS ← PoS + (tagger.tag(element), token)
    for each element in PoS do
        pattern ← pattern + element.replace((element.tag, element.token),
                (element.tag,"token"))
    return pattern
end function
```

Algoritmo 6. Busca construção

```
Function adaptDevice.SearchConstruction (pattern)
    construction ← "empty"
    matrixConst ← initConstructions
    for each element in matrixConst do
        if search(pattern, element.construction) then
            construction ← element.construction
            break
        end if
    return construction
end function
```

Algoritmo 7. Substitui palavra-chave por *tokens* da sentença

```
Function adaptDevice.ReplaceKeyword (construction, sentence)
    tokens ← sentence.tokenize
    if construction is not ‘empty’ then
        for each element in tokens do
            strAsIs ← element.token + 'token'
            strToBe ← element.token + element.tag
            parse ← parse.replace(strAsIs, strToBe, 1)
    else parse  ← '(S(VAZIO))'
    end if
    return parse
end function
```

Apesar do crescente interesse em gramática de construções, ainda não existe um formalismo amplamente aceito para representá-la nem um modelo computacional robusto que operacionalize como as construções podem ser usadas na compreensão ou formulação de enunciados (STEELS, 2017). Nesta tese, os corpora CINTIL *Treebank* (COSTA; BRANCO, 2010) e Bosque (AFONSO et al., 2002) foram adaptados para serem usados como construções. Trata-se de uma simplificação, visando testar o método apresentado e comparar os resultados com outras técnicas de análise sintática. O Algoritmo 8 descreve os passos usados para criar as construções. Inicialmente, ele recebe o corpus como parâmetro de entrada, substitui os *tokens* das sentenças pela palavra chave ‘token’ e monta uma matriz com as sentenças modificadas, retornando o resultado para o analisador sintático usar nas análises.

Algoritmo 8. Extração de construções do corpus

Function *getConstructions* (corpus)

```
out ← ”empty”
matrix ← initMatrix
for each line in corpus do
        output ← line.sub(words|numbers|punctuation, ' token')
        matrix ← matrix.append(output, count(output))
return matrix
```

end function

Basicamente, o algoritmo usa uma expressão regular para substituir os símbolos não terminais das sentenças do corpus pela palavra-chave “token”. Após a substituição, cada sentença e o total de *tokens* são incluídos em uma matriz, que é disponibilizada para o analisador sintático. Por exemplo, a sentença “Choveu .” é apresentada no corpus CINTIL como (S (V Choveu) (PNT .)) Após a transformação da expressão regular, ela passa a ter o formato (S (V token) (PNT token)). Esta linha, junto com a contagem de *tokens*, é incluída na matriz de resultados, ficando disponível para o analisador sintático da seguinte maneira [[(S (V token) (PNT token)), 2]]. Exemplos das construções geradas por este método encontram-se nos APÊNDICES G, H, I e J.

7 CONSTRUÇÕES DINÂMICAS

Este capítulo apresenta o módulo para analisar os textos por meio de construções dinâmicas, ou seja, estruturas que são construídas ao longo da análise das sentenças.

7.1 Processo

O módulo de construções dinâmicas é usado para analisar textos por meio de um mecanismo que cria as construções dinamicamente durante a análise, sem a necessidade de usar estruturas prontas, criadas previamente por processos externos. Tal recurso mostra-se particularmente conveniente quando não se dispõe de todas as construções necessárias ou quando as construções existentes não são suficientes para analisar novas sentenças. O processo é apresentado em detalhes nas seções seguintes.

7.2 Análise sintática

Ao contrário da análise feita por meio de construções estáticas, no caso de análise por construções dinâmicas, é necessário que o analisador utilize elementos mais granulares do que as estruturas gramaticais prontas. Uma forma de se obter tais estruturas é através de regras de produção de uma gramática livre de contexto. As regras de produção são menos dependentes do domínio de análise, visto que são usadas para gerar todas as sentenças de uma determinada linguagem e, portanto, mais adequadas para montar as construções de que o analisador não dispõem. Além disso, as regras de produção podem ser aprendidas por meio de um algoritmo de aprendizado de máquina processando um corpus pré-anotado, criando, assim, as condições necessárias para que o analisador possa montar as construções durante a análise da sentença. A Figura 23 apresenta a estrutura do Analisador Sintático de Construções Dinâmicas.

A análise segue a seguinte sequência de etapas:

a) Inicialmente, o Analisador de Contexto instancia os parâmetros usados para identificar o contexto de análise: critério de seleção das regras de produção,

técnica usada para desambiguar as regras, sentido de análise da sentença e uso de método híbrido;

b) Depois, o Seletor de Produções seleciona todas as possíveis regras de produção a partir da leitura completa da sentença de entrada, representada pela classificação morfológica dos *tokens* da sentença (*PoS*);

c) Em seguida, o Montador de Construções faz a desambiguação, levando em consideração abrangência das regras, frequência de ocorrência, histórico de símbolos e verificação de restrições, conforme definido na parametrização;

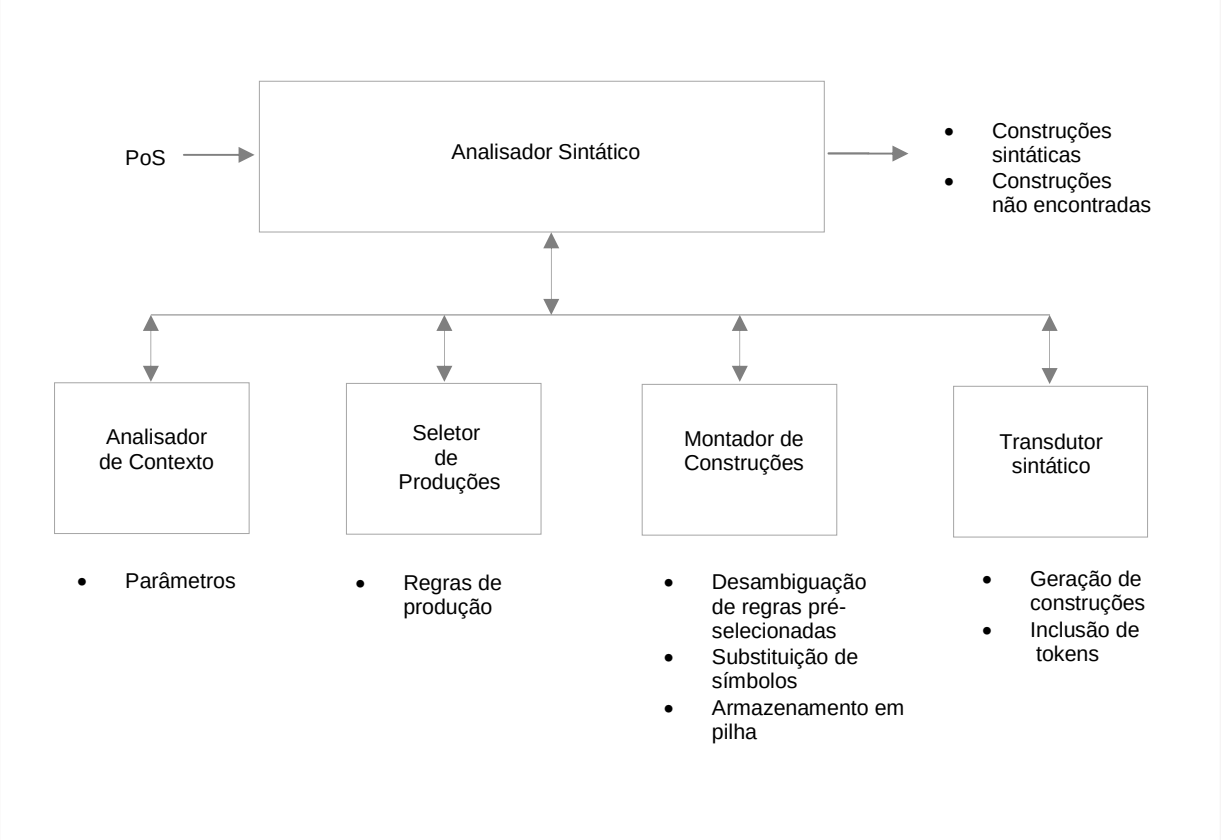


Figura 23 – Analisador Sintático por Construções Dinâmicas

Fonte: Autor

d) Depois, o Montador de Construções substitui os *tokens* da sentença equivalentes aos símbolos do lado direito da regra de produção escolhida no passo anterior pelo símbolo do lado esquerdo da regra (*bottom-up parsing*) e armazena o resultado em uma pilha. O processo se repete de forma recursiva

até que a sentença analisada fique com um único elemento. Se este elemento for “S”, então a sentença é reconhecida, caso contrário ela é rejeitada;

e) Na última etapa, o Transdutor Sintático realiza o processo inverso, lendo as informações armazenadas na pilha e gerando as construções sintáticas da sentença analisada. Ao final, o Transdutor Sintático acrescenta os *tokens* da sentença analisada de acordo com a classificação morfológica.

A regra de produção abaixo é um exemplo das regras usadas pelo Analisador Sintático para montagem das construções:

AP -> A CONJP [0.00985]

O lado esquerdo da regra apresenta o símbolo AP, um sintagma adjetival, usado para substituir os símbolos do lado direito da regra, respectivamente um adjetivo A e um sintagma conjuntivo CONJP. Entre colchetes, encontra-se a frequência com que a regra aparece no corpus de treinamento, determinada durante o processo de aprendizado. Outros exemplos de regras de produção encontram-se no APÊNDICE K.

O Analisador Sintático segue um processo no qual toda a sentença é lida, identificando as regras de produção unárias, binárias, ternárias e quaternárias, definidas segundo o número de símbolos apresentados do lado direito: um, dois, três, ou quatro símbolos (o limite de quatro símbolos foi definido após a realização de testes preliminares).

Por exemplo, no caso da sentença V CARD N PNT, o analisador inicia identificando todas as regras unárias formadas pelo *token* V no lado direito, depois as regras binárias compostas pelos *tokens* V CARD, em seguida, as ternárias formadas pelos *tokens* V CARD N e, por fim, as regras quaternárias compostas pelos *tokens* V CARD N PNT. O analisador se desloca para a direita da cadeia e repete o processo, iniciando, agora, pelo *token* CARD e, seguindo, sucessivamente, até o final da sentença. A Figura 24 ilustra visualmente este processo.

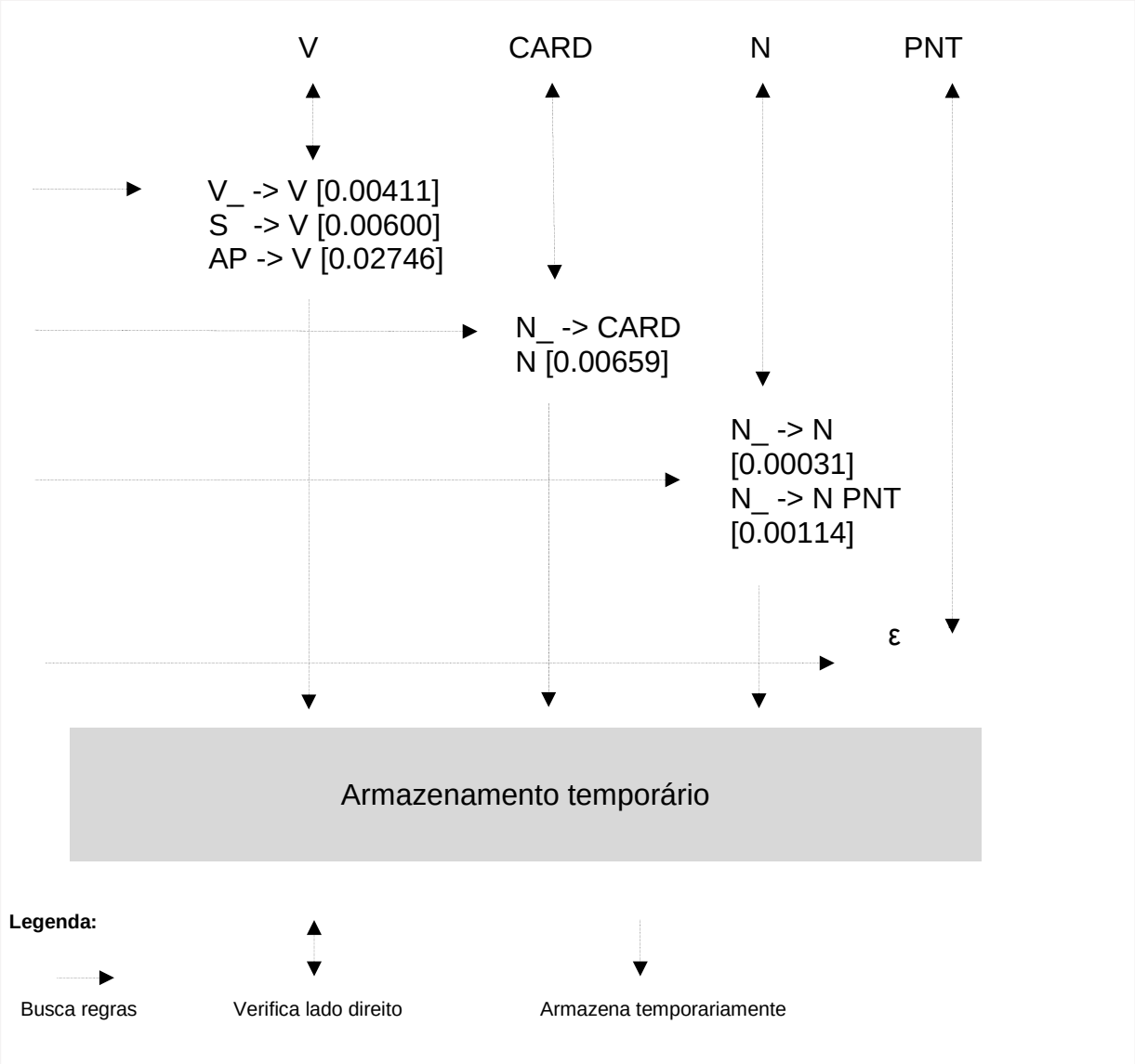


Figura 24 – Exemplo de Seleção de Regras de Produção

Fonte: Autor

No passo seguinte, o analisador aplica os critérios de desambiguação das regras e seleciona a que será utilizada. Os critérios de desambiguação, definidos em parâmetros, são:

a) Abrangência da regra: Utiliza a abrangência do lado direito da regra de produção como principal critério de desambiguação, vindo, em seguida, a frequência com que a regra aparece no corpus de treinamento;

b) Frequência de ocorrência da regra: Utiliza a frequência com que a regra aparece no corpus de treinamento como único critério de desambiguação;

c) Histórico de símbolos: Indica que a seleção da regra levará em consideração o histórico de etiquetas já selecionadas;

d) Verificação de restrições: Leva em consideração as restrições identificadas no corpus de treinamento, além do histórico de etiquetas já selecionadas.

A Figura 25 apresenta visualmente a aplicação dos critérios de desambiguação por abrangência e frequência.

Regras Candidatas

AP -> V [0.02746]	N_ -> CARD N [0.00659]
S -> V [0.00600]	N_ -> N [0.00031]
V_ -> V [0.00411]	N_ -> N PNT [0.00114]

Regras de Maior Abrangência

N_ -> CARD N [0.00659]

N_ -> N PNT [0.00114]

Regra de Maior Frequência

AP -> V [0.02746]

Regra de Maior Abrangência e Frequência

N_ -> CARD N [0.00659]

Figura 25 – Exemplo de Aplicação dos Critérios de Abrangência e Frequência
Fonte: Autor

O critério de desambiguação por histórico de símbolos toma por base as escolhas realizadas pelo analisador nos passos anteriores para a escolha da regra seguinte. Diferentemente dos critérios anteriores, neste caso o analisador considera o lado direito da regra como uma palavra e o lado esquerdo como a etiqueta que classifica

a palavra, funcionando como um classificador morfológico n-gramas, em que n representa o número de símbolos do histórico usados na análise. Se n for igual a 1, apenas a frequência da regra analisada é levada em consideração; se n for igual a 2, a frequência da regra e o símbolo anterior são considerados na análise; se n for igual a 3, a frequência da regra e os dois últimos símbolos são considerados na análise. Como a palavra é representada pelo lado direito da regra de produção e ele pode variar em função da abrangência, o analisador escolhe a regra de maior abrangência para ser submetida ao classificador n-gramas. A Figura 26 ilustra o critério visualmente.

Regras Candidatas

N_ -> CARD N [0.00659]

N_ -> N [0.00031]

N_ -> N PNT [0.00114]

Último Símbolo

Vazio (primeira substituição)

Palavras analisadas

V, CARD N, N, N PNT

Regras de Maior Abrangência

N_ -> CARD N [0.00659] N_ -> N PNT [0.00114]

Palavra Submetida ao Classificador N-gramas

CARD N

Símbolo escolhido

N_

Figura 26 – Exemplo de critério de Histórico de Símbolos

Fonte: Autor

O Analisador Sintático pode verificar se existem restrições à aplicação das regras de produção. A análise de restrições leva em consideração os símbolos vizinhos aos símbolos analisados, tanto do lado esquerdo quanto do lado direito, e avalia se alguma regra gramatical é violada, caso a regra de produção seja aplicada. Por exemplo, no caso da regra N_ -> CARD N [0.00659], os símbolos CARD e N são avaliados juntamente com os seus símbolos vizinhos C e PNT, para determinar se a substituição por N_ pode ser realizada. A Figura 27 ilustra o exemplo visualmente.

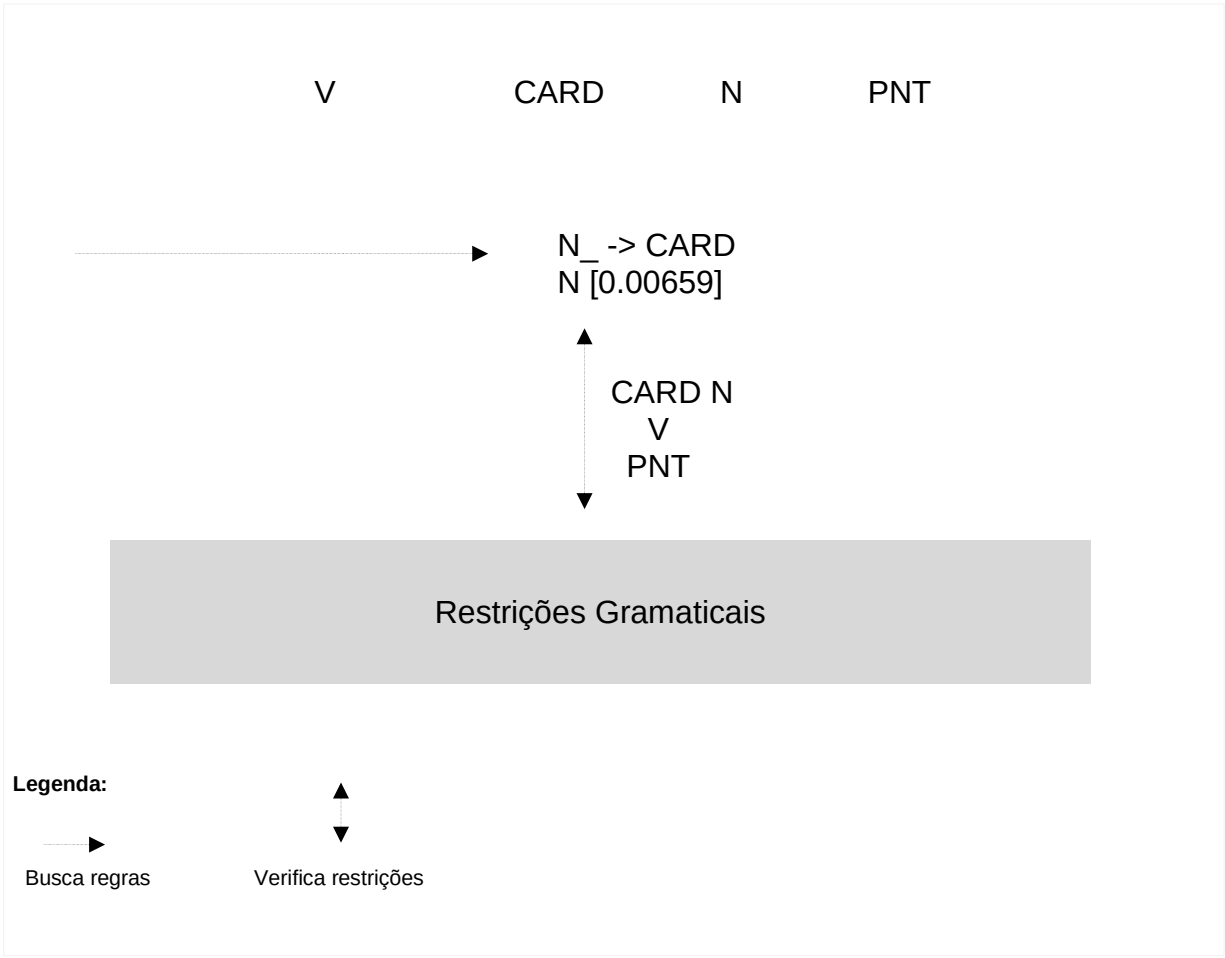


Figura 27 – Exemplo de Análise de Contexto

Fonte: Autor

As restrições foram identificadas no corpus CINTIL através da análise de parênteses, como é o caso assinalado em vermelho na sentença anotada abaixo. A regra NP -> ART N não pode ser aplicada na posição assinalada em vermelho, pois entre os símbolos ART e N existe o símbolo N_. De posse desta informação, o

analisador deveria correr a cadeia em busca de outra possibilidade de substituição, que é encontrada ao final, assinalada em negrito.

(S (S (NP) (VP (V) (NP (N_ (N) (PP (P) (NP (ART) (N_ (N) (PP (P) (NP **(ART) (N)**)))))))) (PNT))

No entanto, o analisador não dispõe desta informação, pois para ele a sentença é apresentada da seguinte maneira:

V N P ART N P ART N PNT

Logo, existem duas possibilidades para aplicar a regra NP -> ART N, sendo que a primeira apresenta restrições e, se for utilizada, pode provocar erros, reduzindo o desempenho geral da análise. Para que este problema pudesse ser contornado, foi desenvolvido um modelo de aprendizado de máquina que busca inferir restrições deste tipo em função do lado direito da regra e dos *tokens* do lado esquerdo e do lado direito. Os detalhes do modelo de aprendizado encontram-se no APÊNDICE O.

É importante ressaltar que podem existir outras regras linguísticas que restrinjam a aplicação das regras de produção. No entanto, neste trabalho, consideraram-se apenas as regras que puderam ser determinadas pelo critério apresentado. Também não foram consideradas outras variáveis além dos *tokens* da vizinhança como forma de inferir restrições. Características tais como altura da arvore de análise sintática, subcategorias ou influência do léxico também não foram consideradas e podem criar algum tipo de restrição que impeça a aplicação da regra. Por outro lado, trabalhos neste sentido podem ser incorporados ao analisador, dada a independência dele em relação ao modelo de restrições.

O fluxo de análise muda quando são consideradas restrições gramaticais, sendo necessário verificar se a regra de produção pode ser aplicada antes que a substituição seja efetivada. A Figura 28 ilustra o critério visualmente. Nota-se que o fluxo é praticamente o mesmo do critério de escolha de símbolos pelo histórico, com a diferença de que agora o analisador verifica a existência de eventuais restrições antes de aplicar a regra.

Regras Candidatas

AP -> V [0.02746]	N_ -> CARD N [0.00659]
S -> V [0.00600]	N_ -> N [0.00031]
V_ -> V [0.00411]	N_ -> N PNT [0.00114]

Último Símbolo

Vazio (primeira substituição)

Palavras analisadas

V, CARD N, N, N PNT

Regras de Maior Abrangência

N_ -> CARD N [0.00659] N_ -> N PNT [0.00114]

Regras com Restrição

N_ -> N PNT [0.00114]

Palavra Submetida Classificador N-gramas

CARD N

Símbolo escolhido

N_

Figura 28 – Exemplo de critério de Histórico de Símbolos e Restrições Linguísticas
Fonte: Autor

O critério híbrido cria duas sequências de construções e escolhe a melhor ao final. A primeira sequência é usada como referência e trabalha com a regra de maior abrangência e frequência. A segunda regra é montada a partir de uma semente que é gerada a partir do segundo melhor elemento da regra de abrangência e frequência ou a partir da regra de maior frequência. A partir daí, todas as escolhas utilizam a regra de maior abrangência e frequência. A Figura 29 ilustra a formação deste

critério visualmente, mostrando como seria a seleção da regra para cada opção de semente.

Regras Candidatas

N_ -> CARD N [0.00659]

N_ -> N [0.00031]

N_ -> N PNT [0.00114]

Regras de Maior Abrangência

N_ -> CARD N [0.00659]

N_ -> N PNT [0.00114]

Regra de Maior Frequência

AP -> V [0.02746]

Regra de Maior Abrangência e Frequência

N_ -> CARD N [0.00659]

Opção1: Segunda Maior Abrangência e Frequência

N_ -> N PNT [0.00114]

Opção 2: Maior Frequência

AP -> V [0.02746]

Figura 29 – Exemplo de formação do critério Híbrido de Abrangência e Frequência
Fonte: Autor

Com a regra de produção identificada, o analisador sintático realiza a troca dos *tokens* representados pelo lado direito pelo símbolo do lado esquerdo da regra e registra a nova cadeia em uma pilha, indicando a posição do símbolo substituído. O processo se repete até que a cadeia seja reduzida a um único *token*. Caso o símbolo deste último *token* seja ‘S’, a cadeia é aceita; caso contrário, ela é rejeitada.

As construções das sentenças aceitas são criadas através de um transdutor que lê a pilha gerada pelo analisador na etapa de reconhecimento e cria a estrutura correspondente através de um processo reverso, substituindo cada símbolo encontrado pelos símbolos correspondentes do lado direito da regra de produção. A Figura 30 ilustra visualmente o funcionamento do transdutor.

Registros na Pilha

<S>

<S> -> <S> <PNT>

<S> -> <V> <NP>

<NP> -> <N_>

<N_> -> <CARD> <N>

Substituições

<S>

(S <S> <PNT>)

(S (S <V> <NP>) <PNT>)

(S (S <V> (NP <N_>)) <PNT>)

(S (S <V> (NP (N_ <CARD> <N>))) <PNT>)

Figura 30 – Exemplo de transdução
Fonte: Autor

No último passo, o transdutor incorpora os símbolos terminais de acordo com a classificação morfológica dos *tokens* e gera a construção final. A Figura 31 ilustra o funcionamento deste passo visualmente. Exemplos de construções geradas pelo analisador encontram-se nos APÊNDICES L e M.

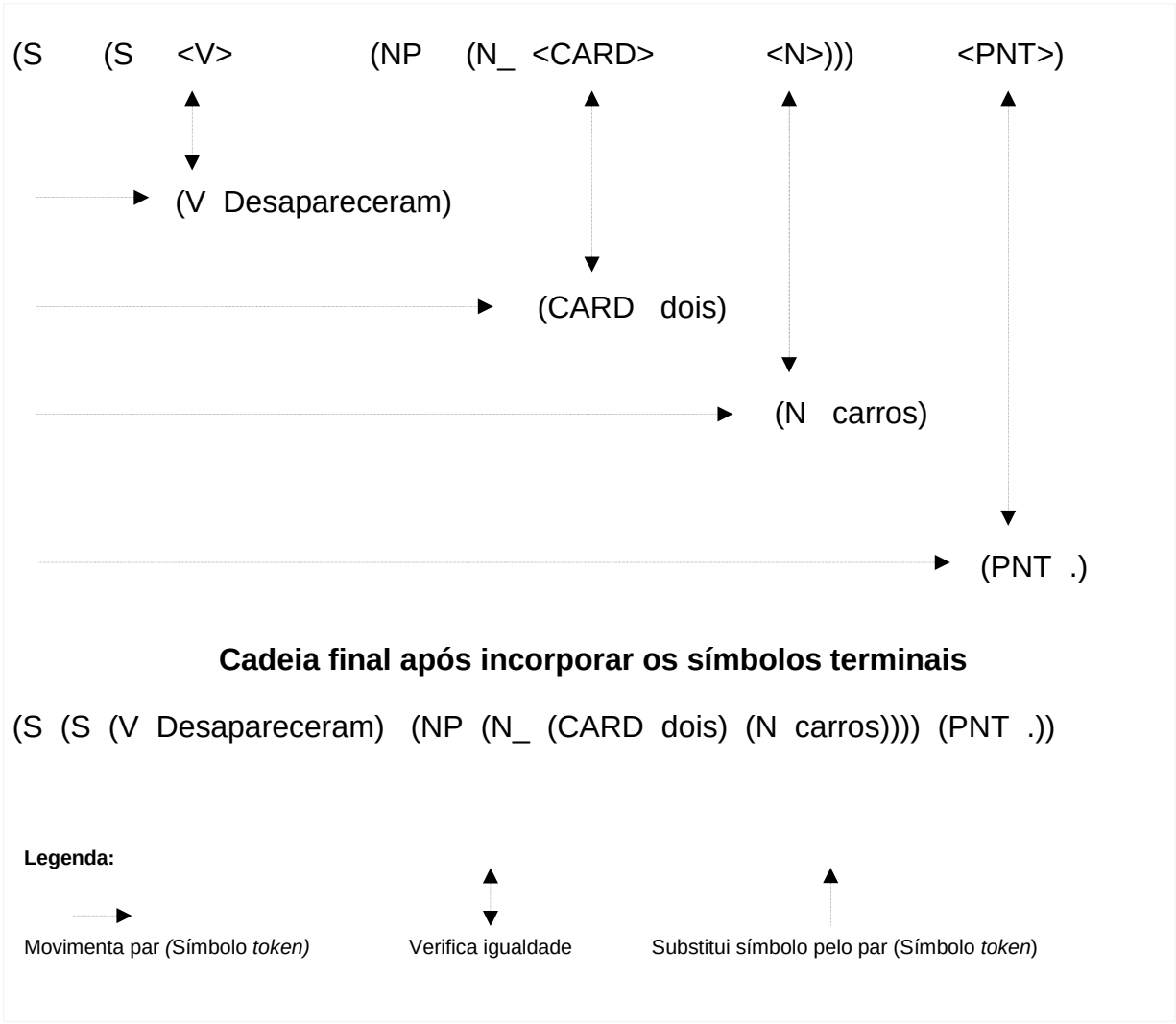


Figura 31 – Exemplo de Incorporação de Símbolos Terminais

Fonte: Autor

7.3 Autômato adaptativo

Para que as construções sejam geradas dinamicamente, é necessário definir outro autômato adaptativo que controle a parametrização utilizada no fluxo de execução apresentado na seção anterior. A configuração inicial deste autômato é apresentada na Figura 32. Observa-se um estado inicial e quatro transições, representando o critério de seleção das regras de produção, a técnica usada para desambiguar as regras, o sentido de análise para identificação das regras de produção e se haverá uso de método híbrido.

O critério de seleção de regras de produção está relacionado à abrangência do lado direito da regra, estando previstas opções para regras unárias , unárias e binárias; unárias, binárias e ternárias; e unárias, binárias, ternárias e quaternárias.

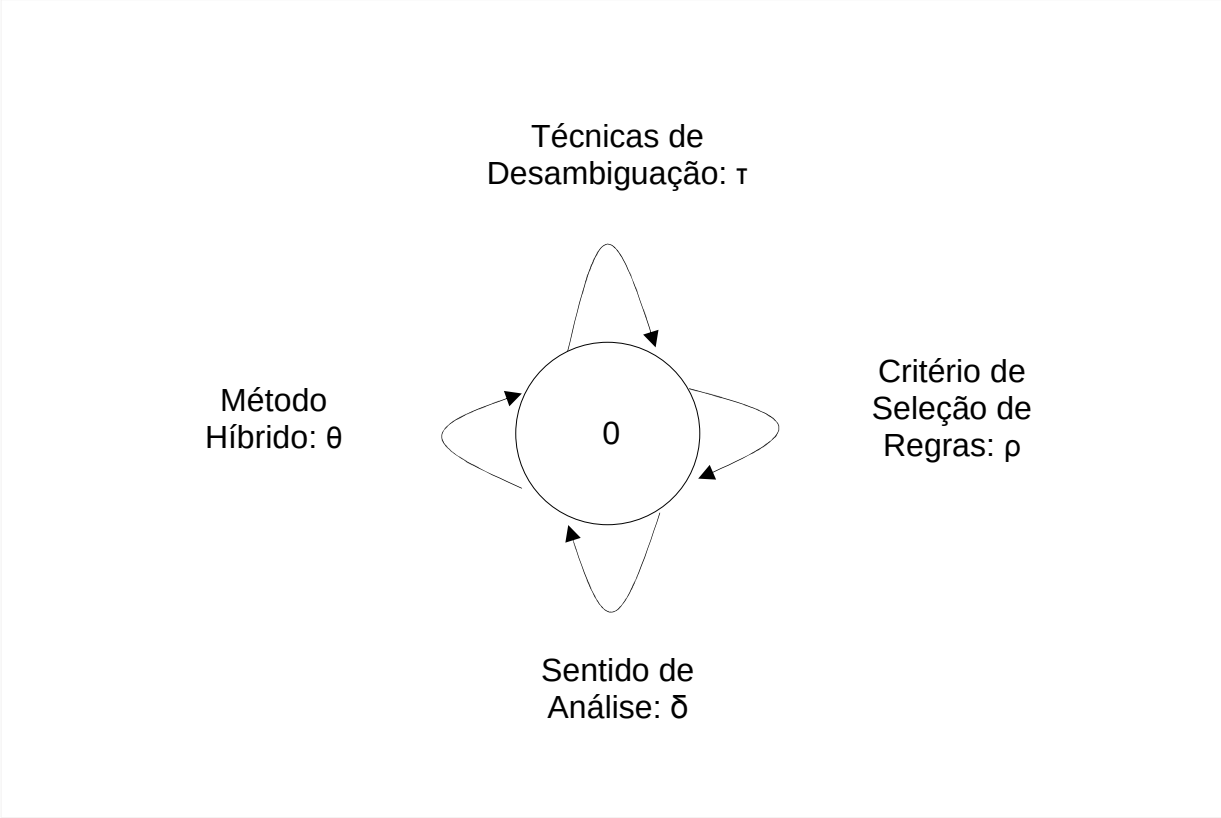


Figura 32 – Analisador Sintático – Configuração Inicial

Fonte: Autor

A Figura 33 apresenta a configuração do autômato após a escolha do critério de seleção de regras de produção. As mudanças de configuração do autômato são realizadas através das seguintes funções adaptativas:

a) α (k): { l*: - [(k, ρ): → k] + [(k, R): → l, β (l)] }

b) β (l): { + [(l, δ): → l] + [(l, θ): → l] + [(l, τ): → l] }

onde:

α (k) e β (l) são funções executadas respectivamente antes e após o consumo do indicador do critério de seleção de regras R;
k indica o estado 0;

l* indica que o estado l deve ser criado;

l indica o estado 1;

δ, θ e τ indicam as transições para o novo estado l.

A função α (k) aplicada ao estado k=0 cria o estado l=1, elimina a transição (0, ρ): → 0 e cria a transição (0, R): → 1. Em seguida, a função adaptativa β (l) é chamada com parâmetro l=1, criando as transições

(1, δ): → 1, (1, θ): → 1 (1, τ): → 1

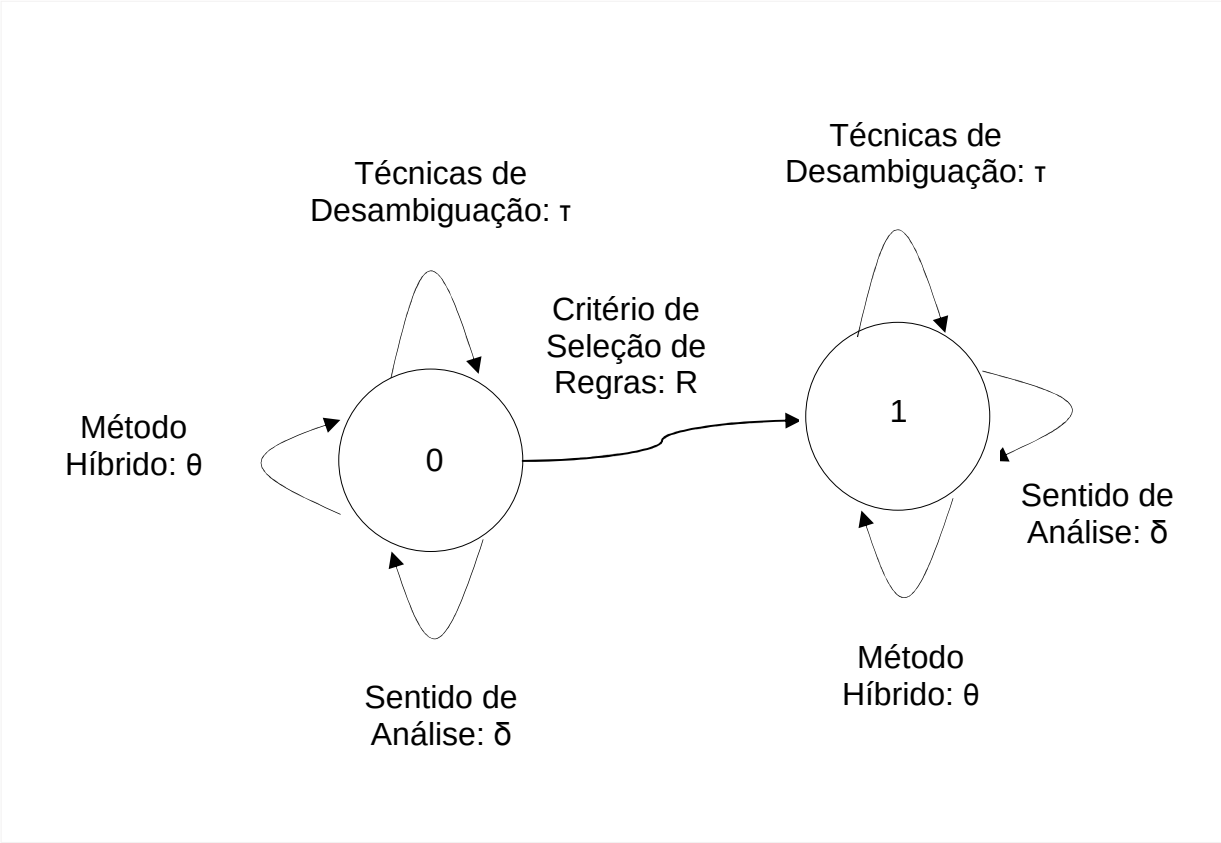


Figura 33 – Analisador Sintático - Após a identificação do critério de seleção das regras de produção

Fonte: Autor

Na etapa seguinte, seleciona-se o sentido de leitura que será usado para aplicar as regras de produção. A Figura 34 apresenta a configuração do autômato após esta

escolha. As mudanças de configuração do autômato são realizadas através das seguintes funções adaptativas:

a) α (k): { l*: - [(k, δ): → k] + [(k, D): → l, β (l)] }

b) β (l): { + [(l, τ): → l] + [(l, θ): → l] }

onde:

α (k) e β (l) são funções executadas respectivamente antes e após o consumo do indicador do critério de Sentido de Análise D;
k indica o estado 1;
l* indica que o estado l deve ser criado;
l indica o estado 2;
τ e θ indicam as transições para o novo estado l.

A função α (k) aplicada ao estado k=1 cria o estado l=2, elimina a transição (1, δ): → 1 e cria a transição (1, D): → 2. Em seguida, a função adaptativa β (l) é chamada com parâmetro l=2, criando as transições (2, τ): → 2 e (2, θ): → 2.

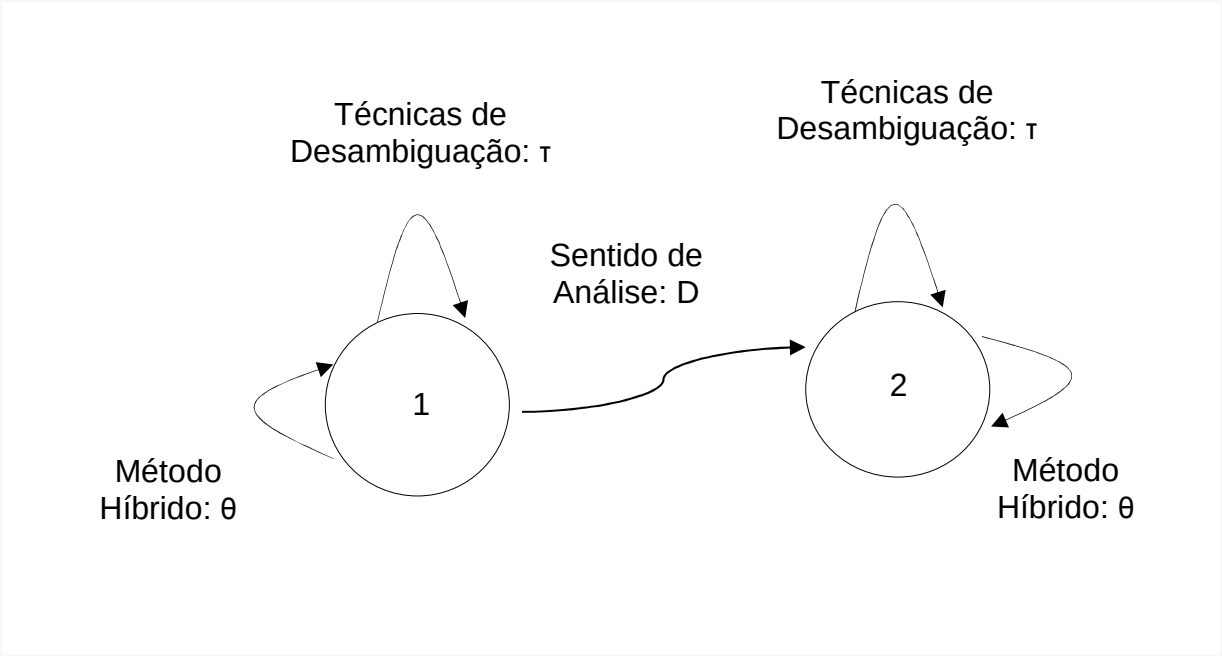


Figura 34 – Analisador Sintático - Após a identificação do sentido de análise

Fonte: Autor

No passo seguinte, define-se o uso ou não do modelo híbrido. A Figura 35 apresenta a configuração do autômato após esta etapa.

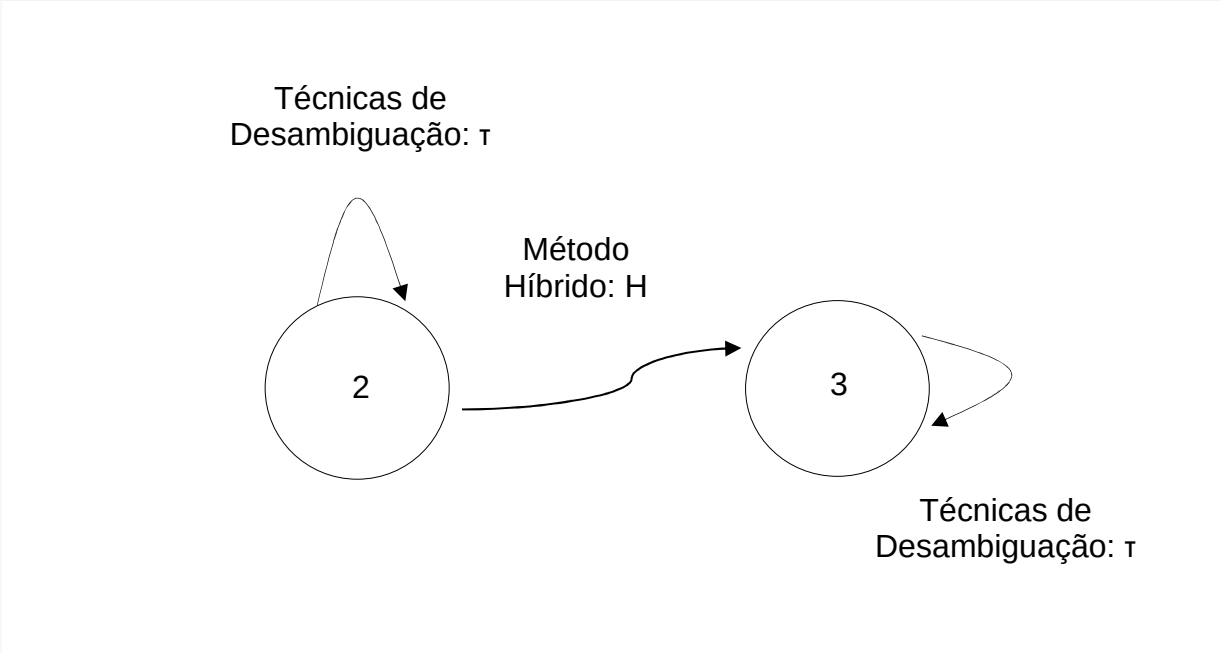


Figura 35 – Analisador Sintático - Após a identificação do uso de modelos híbridos

Fonte: Autor

As mudanças de configuração do autômato são realizadas através das seguintes funções adaptativas:

a) α (k): { l*: - [(k, θ): → k] + [(k, H): → l, β (l)] }

b) β (l): { + [(l, τ): → l] }

onde:

α (k) e β (l) são funções executadas respectivamente antes e após o consumo do indicador do critério de Método Híbrido H;
k indica o estado 2;
l* indica que o estado l deve ser criado;
l indica o estado 3;
τ indica a transição para o novo estado l.

A função α (k) aplicada ao estado k=2 cria o estado l=3, elimina a transição (2, θ): → 2 e cria a transição (2, H): → 3. Em seguida, a função adaptativa β (l) é chamada com parâmetro l=3, criando a transição (3, τ): → 3.

No último passo, é definida a técnica de desambiguação usada para definir qual critério será aplicado para escolher a regra de produção entre as pré-selecionadas. Estão previstas quatro técnicas de desambiguação:

a) Abrangência: Esta opção utiliza a abrangência do lado direito da regra de produção como principal critério de desambiguação, vindo, em seguida, a frequência com que a regra aparece no corpus de treinamento;

b) Frequência: Esta opção utiliza a frequência com que a regra aparece no corpus de treinamento como único critério de desambiguação;

c) Seleção com base no histórico de etiquetas: Esta opção indica que a seleção da regra levará em consideração o histórico de etiquetas já selecionadas;

d) Seleção com verificação de restrições: Esta opção indica que a seleção da regra levará em consideração as restrições identificadas no corpus de treinamento, além do histórico de etiquetas já selecionadas.

A Figura 36 apresenta a configuração do autômato após a escolha da Técnica de Desambiguação. As funções adaptativas ficam da seguinte forma:

a) α (q): { r*: - [(q, τ): → q] + [(q, T): → r, β (r)] }

b) β (r): { s*, t*: + [(r, R): → r] + [(r, ‘S’): → “Aceito”] + [(r, No ‘S’): → “Rejeitado”] }

onde:

α (q) e β (r) são funções executadas respectivamente antes e após o consumo do indicador da técnica de escolha de regras T;
q indica o estado 3;
r* indica que o estado r deve ser criado;
r indica o estado 4;

s* indica que o estado s deve ser criado;
s indica o estado “Aceito”;
t* indica que o estado t deve ser criado;
t indica o estado “Rejeitado”;
‘S’ indica a transição para o novo estado “Aceito”;
No ‘S’ indica a transição para o novo estado “Rejeitado”.

A função α (q) aplicada ao estado q=3 cria o estado r=4, elimina a transição (3, τ): → 3 e cria a transição (3, T): → 4. Em seguida, a função adaptativa β (r) é chamada com parâmetro r=4, criando os estados s= ”Aceito” e t = “Rejeitado”, e as transições (4, R): → 4, (4, ‘S’): → “Aceito” e (4, No ‘S’): → “Rejeitado”.

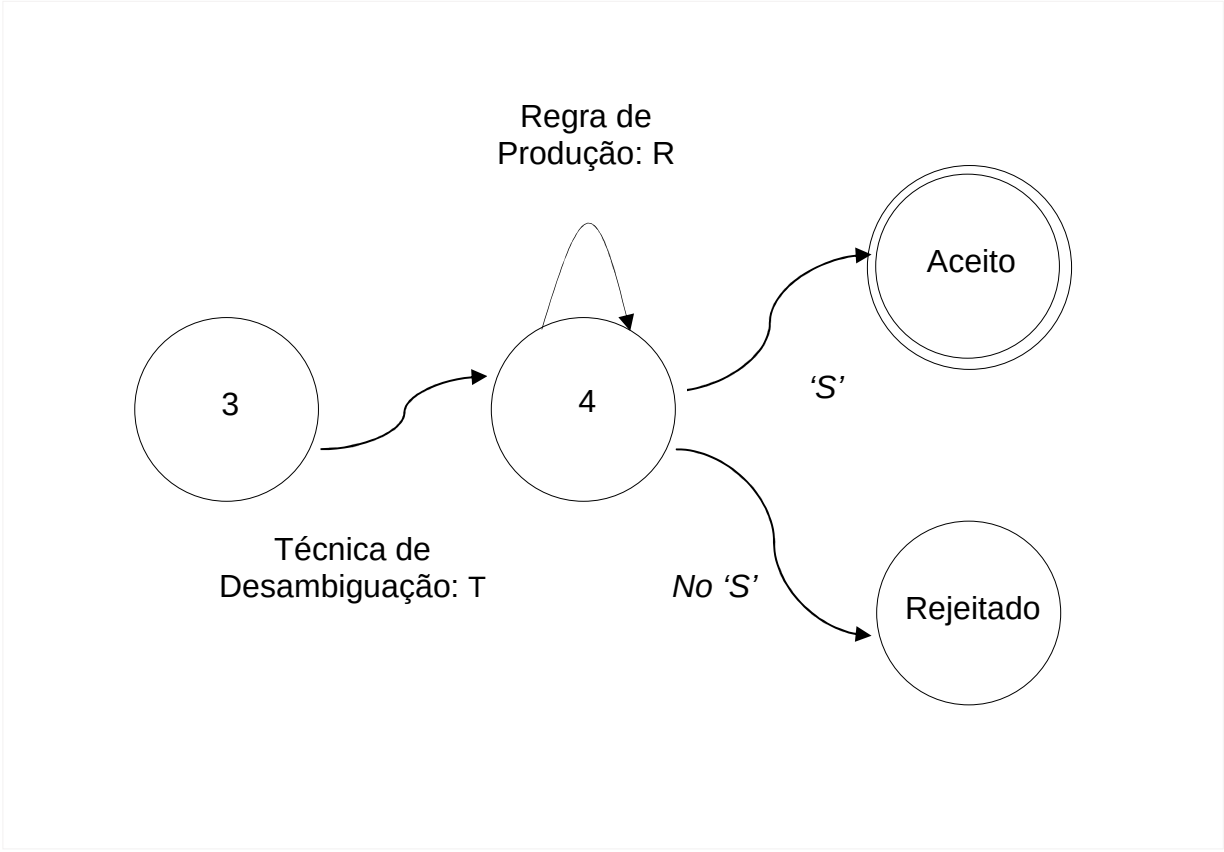


Figura 36 – Analisador Sintático - Após a seleção da técnica de desambiguação
Fonte: Autor

7.4 Algoritmos

Os algoritmos apresentados nesta seção referem-se à leitura das sentenças, seleção das regras de produção candidatas, aplicação do critério de escolha de regras e transdução do reconhecimento para a construção sintática.

O Algoritmo 9 é responsável pela leitura da sentença, controle da pilha e chamada do transdutor. Ele recebe a sentença, um indicativo da direção a ser utilizada na análise, outro informando o critério de escolha de regras e um último indicando se será utilizado método híbrido. Ele inicia o processo obtendo as regras da gramática e dividindo a sentença em *tokens.* Em seguida*,* ele entra em um laço que aplica os critérios de escolha de regras de produção e substitui os *tokens* da cadeia pelo símbolo do lado esquerdo da regra aplicada, parando quando a sentença analisada tem comprimento 1 e aceitando-a se o símbolo for igual a ‘S’. Ao final, o Algoritmo 9 chama o transdutor responsável por gerar as construções das sentenças reconhecidas.

O Algoritmo10 seleciona as regras de produção tomando por base o número de elementos do lado direito (regras unárias, binárias, ternárias e quaternárias), retornando as regras candidatas. O Algoritmo 11 aplica os critérios de escolha das regras candidatas pré-selecionadas pelo Algoritmo 10, utilizando, como critérios, a abrangência do lado direito, a frequência com que as regras aparecem no corpus de treinamento, o histórico de regras já aplicadas e restrições linguísticas.

O Algoritmo 12 faz a transdução das sentenças reconhecidas, substituindo os símbolos gravados na pilha pelos símbolos do lado direito da regra gramatical correspondente. As sentenças não reconhecidas são convertidas em uma construção padronizada indicando que nenhuma análise foi gerada.

O Algoritmo 13 descreve os passos usados pelo analisador para criar duas análises e escolher a melhor delas ao final. O Algoritmo 13 interage com o Algoritmo 9 para obter as duas análises, invocando-o uma vez para obter a análise gerada pelas regras gramaticais de maior frequência no corpus de treino e outra para obter a análise gerada pelas regras de maior abrangência. O Algoritmo 14 descreve como é gerada a primeira regra usada pelo Algoritmo 13.

Algoritmo 9. Lê a sentença, grava a pilha e chama transdutor

Function *parseSentence (*sentence, direction, dtype, hybrid)

```
unchuncked ← sentence, chuncked ← 'empty'
matrixRules ← getGrammarRules(), tokens ← tokenize(sentence)
while unchuncked is different from chuncked  do
    if len (unchuncked) is 1then if unchuncked is 'S' then break
    candidateRules ← getCandidateRules(tokens, matrixRules)
    if hybrid is True then
        if  unchuncked is first then
            selectedRule ← getSeedHybridMethod (candidateRules, dtype)
    else selectedRule ← applyDisambiguationRules (candidateRules,
                                    dtype,stackHistory, direction)
    if direction is 0 (left to right) then
      chuncked ←  reduceLR(unchuncked, selectedRule)
    else (right to left) then
      chuncked ←  reduceRL(unchuncked, selectedRule)
    stack ← [selectedRule, span , chuncked]
    unchuncked ← chuncked, chuncked ← 'empty'
parse ← getTransduction (stack, PoS)
```

end function

Algoritmo 10. Seleciona as regras de produção candidatas

```
Function getCandidateRules (tokens, matrixRules,dType)
    i ← 0
    end ← len(tokens)
    while i < end do
        for each element in matrixRules do
            if dType is 0(unary+binary) then
                if len(element.RHS) is 1 then
                    if element.RHS is token[i] then
                        CandidateRules ← CandidateRules + element
                if  len(element.RHS) is 2 then
                    if element.RHS is (token[i] + token[i+1]) then
                        CandidateRules ← CandidateRules + element
            if dType is 1(unary+binary+ternary) then
                if  len(element.RHS) is 3 then
                    if element.RHS is (token[i] + token[i+1]+ token[i+2]) then
                        CandidateRules ← CandidateRules + element
            if dType is 2 (unary+binary+ternary+quaternary) then
                if  len(element.RHS) is 4 then
                    if element.RHS is (token[i] + token[i+1]+ token[i+2]+ token[i+3])
                  then
                        CandidateRules ← CandidateRules + element
     return CandidateRules
end function
```

Algoritmo 11. Aplica critérios de escolha de regras de produção

Function *applyProductionSearchRules* (candidateRules, dtype, stackHistory, direction)

```
finalSpan = 0
freq. = 0.0
for each element in candidateRules do
    if dtype is span then
        if element.span > finalSpan then
            if element.freq > freq. then
                selectedRule ← element
    if dtype is freq then
        if element.freq>freq. then
            selectedRule ← element
    if dtype is history then
        if element.span > finalSpan then
            selectedRule ← getHMM(element.RHS, stackHistory)
    if dtype is restrictions then
        if element.span > finalSpan then
            if not getRestrictions(element) then
                selectedRule ← getHMM(element.RHS, stackHistory)
return selectedRule
```

end function

Algoritmo 12. Faz a transdução dos elementos gravados na pilha

```
Function getTransduction (stack, PoS)
    for each element in reversed(stack) do
        if element is first then
            parse ← element.rule
        else
            span ← getSpan(element.rule)
            parse ← replace (parse.symbol, element.rule, span)
    for each element in PoS do
        parse ← replace(parse.symbol, PoS.pair)
    if parse is 'empty' then parse ← '(S(VAZIO))'
    return parse
end function
```

Algoritmo 13. Apura melhor de duas sequências pelo método híbrido

```
Function bestSequence (sentence, direction, hybrid)
    for each element in sentence do
        parseSpanFreq ← parseSentence (element, direction, dtype = 'freq',
                        hybrid=True)
        parseHybrid ← parseSentence (element, direction, dtype = 'span,
                      hybrid=True)
    if prob(parseHybrid) > prob(parseSpanFreq) then parse ← parseHybrid
    else parse ← parseSpanFreq
    return parse
end function
```

Algoritmo 14. Gera semente do método híbrido

```
Function getSeedHybridMethod (candidateRules, dtype)
    finalSpan = 0,  freq. = 0.0
    for each element in candidateRules do
        if dtype is span then
            if element.span > finalSpan then
                if element.freq>freq. then
                    selectedRule ← selectedRule .append(element)
        if dtype is freq then
            if element.freq>freq. then
                selectedRule ← selectedRule .append(element)
    if dtype is span then
        selectedRule ← selectedRule .pop()
    return selectedRule
end function
```

8 COMBINAÇÃO DE CONSTRUÇÕES

Este capítulo apresenta o módulo para analisar os textos por meio de um mecanismo que combina construções estáticas com construções criadas dinamicamente durante a análise. Tal recurso mostra-se útil quando se dispõe de parte das construções estáticas e se deseja ampliar este conjunto sempre que as construções existentes não se mostrem suficientes para analisar novas sentenças.

8.1 Processo

As etapas de preparação e de etiquetagem morfológica são feitas da mesma forma que nos módulos apresentados previamente. A mudança ocorre especificamente na análise sintática, que inicialmente utiliza o módulo de construções estáticas, e, caso não as encontre, lança mão do módulo de construções dinâmicas. A estrutura deste módulo é apresentada a seguir, de forma detalhada.

8.2 Análise sintática

A Figura 37 apresenta o fluxo usado no processamento de sentenças combinando os métodos de construções estáticas e construções dinâmicas. O analisador sintático segue a seguinte sequência de etapas:

a) Inicialmente, o analisador sintático chama o módulo de análise de construções estáticas, de acordo com os passos descritos na seção 6.1;

b) Caso a análise seja concluída com sucesso, o resultado é gerado sem chamada do módulo de construções dinâmica;

c) Caso o analisador não encontre construção estática para a sentença analisada, ele chama o módulo de construções dinâmicas, de acordo com os passos descritos na seção 7.1;

d) Caso a análise seja concluída com sucesso, a construção encontrada é apresentada e, caso contrário, o analisador gera uma cadeia indicando que a análise retornou um conjunto vazio de resultados.

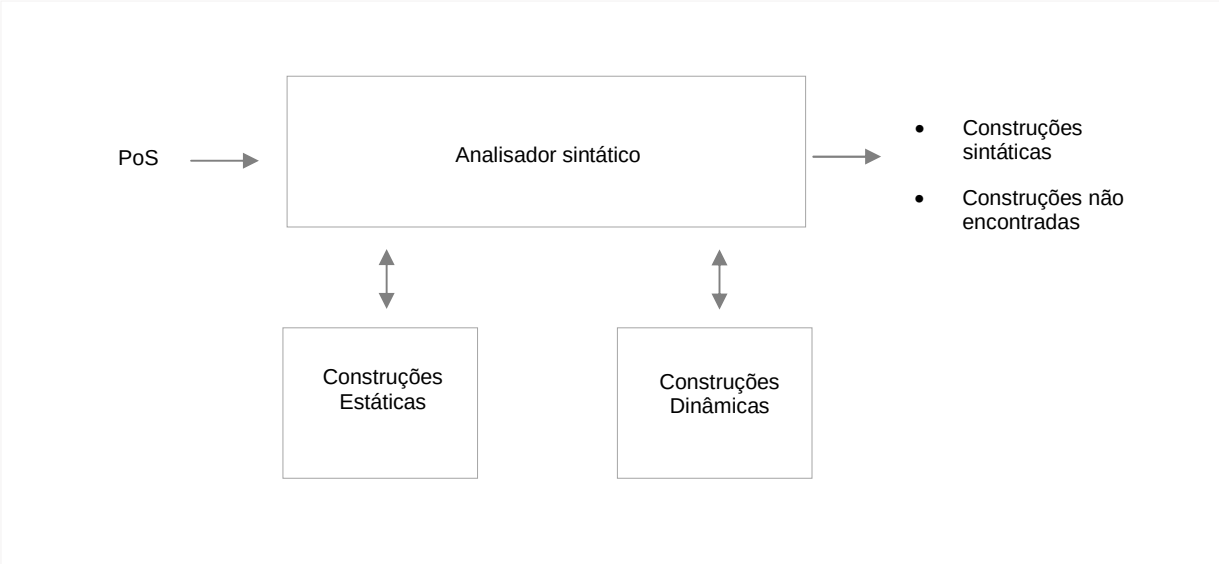


Figura 37 – Diagrama da Combinação de Construções

Fonte: Autor

8.3 Autômato adaptativo

O método combinado utiliza os autômatos adaptativos dos métodos de construções estáticas e dinâmicas, com uma modificação no passo final do autômato de construções estáticas para representar a chamada ao autômato de construções dinâmicas, quando ele não encontra nenhuma construção para a sentença analisada. A Figura 38 apresenta o autômato com esta nova configuração.

As mudanças de configuração do autômato são realizadas através das seguintes funções adaptativas:

a) α (k): { l*: + [(k, ε): → l, β (l)] }

b) β (l): { + [(l, τ): → l] + [(l, ρ): → l] + [(l, δ): → l] + [(l, θ): → l] }

onde:

α (k) e β (l) são funções executadas respectivamente antes e após a transição em vazio a partir do estado “Rejeitado”;
k indica o estado ‘Rejeitado’;
l* indica que o estado l deve ser criado;
l indica o estado 0;

τ, ρ, δ e θ indicam as transições para o novo estado l.

A função α (k) aplicada ao estado k=’Rejeitado’ cria o estado l=0 e a transição (’Rejeitado’, ε): → 0. Em seguida, a função adaptativa β (l) é chamada com parâmetro l=0, criando as transições (0, τ): → 0, (0, ρ): → 0, (0, δ): → 0 e (0, θ): → 0. A partir daí, o autômato se comporta como o autômato de construções dinâmicas, descrito na seção 7.1.

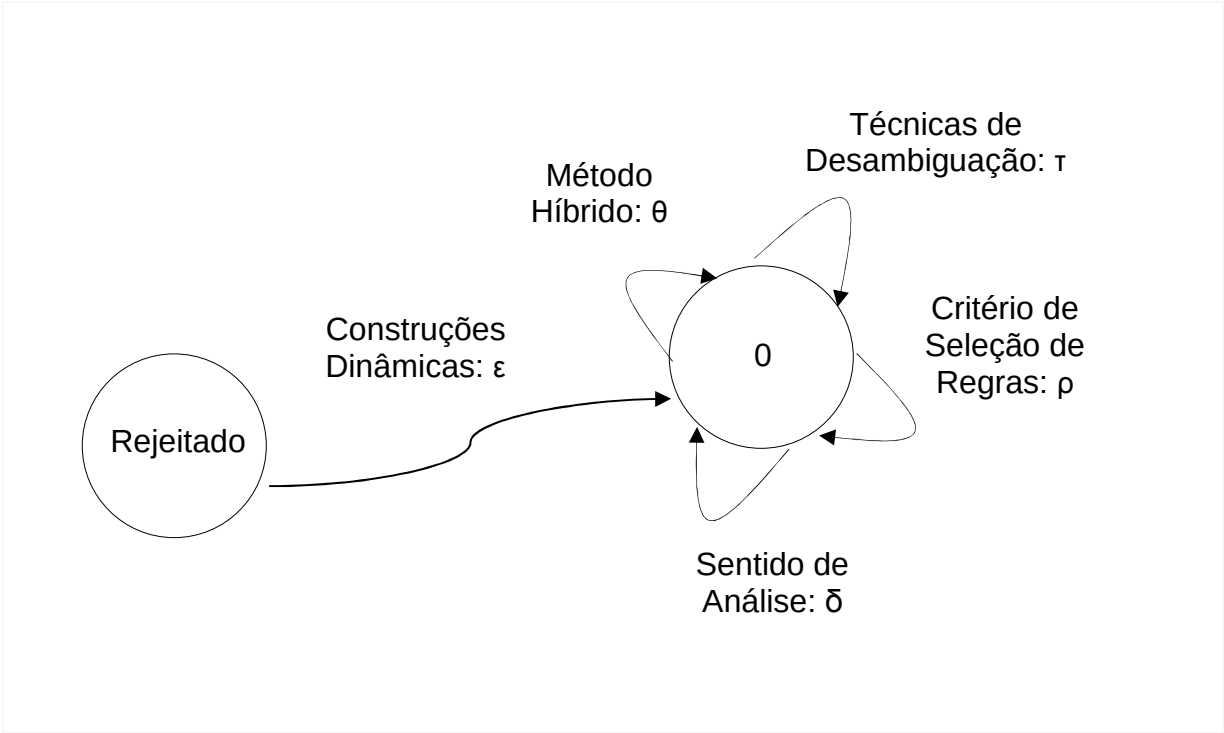


Figura 38 – Analisador Sintático - Após a chamada ao autômato de construções dinâmicas

Fonte: Autor

8.4 Algoritmos

Assim como acontece com os autômatos, os algoritmos usados pelo método de combinação de construções são os mesmos, com exceção do módulo principal do analisador de construções estáticas que é alterado para refletir a chamada do módulo de construções dinâmicas, quando a construção estática não é encontrada. O algoritmo 15 apresenta a alteração mencionada.

Algoritmo 15. Módulo principal do Analisador de Construções Combinadas

```
Function adaptDevice.SintacticAnalysis (sentences)
    matrixParse ← "empty"
    currentConfiguration ← adaptFunc.setConfigurations()
    for each element in sentences do
      sentencePattern ← adaptdevice.CreatePattern(element)
      sentenceConstruction ←
        adaptDevice.SearchConstructions(sentencePattern)
      sentenceParse ←
        adaptDevice.ReplaceKeyword(sentenceConstruction)
      if sentenceParse is 'empty' then
        sentenceParse ← parseSentence (sentence, direction, hybrid)
      matrixParse ← matrixParse + sentenceParse
    return matrixParse
end function
```

9 COMPLEXIDADE ALGORITMICA

Este capítulo apresenta uma breve discussão sobre a complexidade algorítmica dos métodos de construções estáticas e construções dinâmicas, comparando-os com os a complexidade dos algoritmos clássicos descritos na seção 2.7. O método de combinação de construções basicamente realiza chamadas para os algoritmos dos métodos estáticos e dinâmicos, o que faz com que sua complexidade algorítmica seja aquela do método chamado.

9.1 Construções estáticas

Observando os algoritmos 5, 6 e 7, nota-se que as construções estáticas têm seu comportamento definido por três operações:

a) Montagem do padrão de construções a partir das etiquetas PoS;

b) Busca das construções que atendem ao padrão definido no item anterior;

c) Substituição da palavra-chave “token” por *tokens* da sentença.

A operação a varia em função do número de *tokens* da sentença; a operação b varia em função do número de construções e do número de *tokens*, e a operação c varia novamente em função do número de *tokens* da sentença.

A complexidade algorítmica das três operações pode ser descrita na notação O-Grande da seguinte forma:

$$O\,(n) + O(n.|C|) + O(n)$$

onde:

n é o tamanho da sentença, representado pelo número de *tokens;*

$|C|$ é o número de construções.

De acordo com a notação O-Grande, conclui-se que a operação mais onerosa é a b, na qual o algoritmo busca a construção que apresenta as mesmas etiquetas PoS dos *tokens* da sentença. Portanto, a análise por construções estáticas é linear em função do tamanho da sentença e linear em função do número de construções da gramática.

9.2 Construções dinâmicas

De acordo com os algoritmos 9, 10 e 11, as construções dinâmicas tem seu comportamento definido por três principais operações:

a) Busca de regras de produções candidatas;

b) Seleção da regra de produção a ser usada na redução da sentença;

c) Posicionamento e redução da sentença, substituindo os símbolos correspondentes ao lado direito da regra de produção pelo símbolo do lado esquerdo.

A complexidade das três operações é representada na notação O-Grande da seguinte forma:

$$O(K.n.|G|.\left(\frac{n}{m-1}\right)) + O(|g|.\left(\frac{n}{m-1}\right)) + O(n.\left(\frac{n}{m-1}\right))$$

onde:

n é o número de *tokens* da sentença;

m é o número de *tokens* do lado direito da regra selecionada ($1<m\leq 4$);

K é o número máximo de *tokens* do lado direito da regra ($K=4$);

$|G|$ é o número de regras da gramática;

$|g|$ é o número de regras selecionadas a partir dos *tokens* da cadeia.

De acordo com a notação O-Grande, nota-se que as operações a e c são quadráticas com relação ao tamanho da sentença, ao passo que a operação b é linear. Porém, a operação a varia também linearmente em função da quantidade de regras de produção da gramática e da constante K. Portanto, a operação a é dentre as três a que apresenta maior complexidade, determinando a complexidade do algoritmo.

A Tabela 3 apresenta um exemplo tomando por base a sentença “adcbdef” e a gramática G, definida abaixo, para facilitar a visualização.

Gramática |G|= 9	a' -> a	b' -> bde	f' -> gh
	a' -> adcb	d' -> def	g' -> i
	d' -> d	e' -> ef	g' -> ih

Tabela 3. Exemplo do algoritmo de construções dinâmicas

Sentença	Leitura **K=4**	Regras Candidatas **|g|=6**	*Regra Escolhida*	Sentença após 1ª redução
adcbdef **n = 7**	a ad adc adcb d dc dcb dcbd c cb cbd cbdef b bd bde bdef d de def e ef f	a' -> a; a' -> adcb d' -> d - b' -> bde d' -> def e' -> ef -	a' -> adcb **m=4**	a'def

Fonte: Autor

9.3 Análise Comparativa

Segundo Nederhof e Satta (2004), a complexidade dos algoritmos CYK, Earley e Shift-Reduce na notação O-Grande é a seguinte:

a) Algoritmo CYK

$$O\ (n^3.\ |G|)$$

onde:

n é o comprimento da cadeia;

$|G|$ é o número de regras da gramática.

b) Algoritmo Earley

$$O\ (n^3.\ |G|^2)$$

onde:

n é o comprimento da cadeia;

$|G|$ é o número de regras da gramática.

c) Algoritmo Shift-Reduce

$$O\ (|R|.\ |G|.\ pn^{p+1})$$

onde:

n é o comprimento da sentença;

p é o comprimento do lado direito da regra gramatical mais longa;

$|R|$ é o número de regras da gramática geradas a partir dos símbolos da operação de deslocamento (SHIFT) armazenados na pilha;

$|G|$ é o número de regras da gramática.

Premissa:

G é uma gramática ambígua.

A Tabela 4 apresenta um quadro resumo com as complexidades dos diferentes algoritmos. Comparando a complexidade dos algoritmos de Construções Estáticas e Construções Dinâmicas com a complexidade dos algoritmos clássicos observa-se que o algoritmo de Construções Estáticas é linear com relação ao tamanho da sentença de entrada, enquanto os algoritmos CYK e Earley são cúbicos. O algoritmo Shift-Reduce é, no pior caso, potência de n com relação ao comprimento da sentença (quando p é igual a n). O algoritmo de Construções Dinâmicas é quadrádico com relação ao tamanho da sentença de entrada. Conclui-se que o algoritmo de Construções Estáticas é menos complexo que os algoritmos CYK, Earley, Shift-Reduce e de Construções Dinâmicas. Em seguida, vêm o algoritmo de

Construções Dinâmicas, depois CYK , em seguida Earley e, por fim, o algoritmo Shift-Reduce.

Tabela 4. Resumo da Complexidade Algoritmica

Algoritmo	Tamanho Sentença	Número Regras Gramática	Número Construções	Tamanho Lado Direito	Fator Regras Pilha	O-Grande
Construções Estáticas	n	-	C	-	-	nC
Construções Dinâmicas	n^2	$4G$	-	-	-	$4n^2G$
CYK	n^3	G	-	-	-	n^3G
Earley	n^3	G^2	-	-	-	n^3G^2
Shift-Reduce	n^{p+1}	G	-	p	R	$pn^{p+1}GR$

Fonte: Autor

O algoritmo de Construções Estáticas é linear com relação ao número de construções da gramática. O algoritmo CYK é linear com relação ao número de regras da gramática, enquanto o algoritmo Earley é quadrádico com relação ao número de regras da gramática e o algoritmo Shift-Reduce é linear com relação ao número de regras da gramática, acrescido de um fator, que é a quantidade de regras do conjunto R. No pior caso, quando R é igual a G, a complexidade do algoritmo Shift-Reduce é quadrádica. O algoritmo de Construções Dinâmicas é linear com relação ao número de regras da gramática. Portanto, conclui-se que não é possível fazer uma comparação direta entre o algoritmo de Construções Estáticas e os demais, já que o número de construções não é utilizado pelos demais algoritmos. Com relação aos demais, CYK e Construções Dinâmicas apresentam a mesma complexidade que é menor do que os algoritmos Earley e Shift-Reduce.

Conclui-se, portanto, que o algoritmo de Construções Dinâmicas é menos complexo que os demais, com exceção do algoritmo de Construções Estáticas, que não permite uma comparação direta. É possível, no entanto, estabelecer uma relação entre as variáveis, visando identificar os limites nos quais os algoritmos são mais eficientes, como apresentado a seguir:

a) Construções Estáticas (CE) X Construções Dinâmicas (CD)

Considerando $O(CE) = nC$ e $O(CD) = n^2G$, as complexidades se igualam quando $nC = n^2G$, ou $C/G = n$. Portanto, os algoritmos se equivalem se a razão entre o número de construções e número de regras da gramática for igual ao comprimento da sentença. Caso a razão seja maior que n, CE é mais complexo que CD e, caso a razão seja menor que n, então CE é menos complexo que CD.

b) Construções Estáticas (CE) X CYK

Considerando $O(CE) = nC$ e $O(CYK) = n^3G$, as complexidades se igualam quando $nC = n^3G$, ou $C/G = n^2$. Portanto, os algoritmos se equivalem se a razão entre o número de construções e número de regras da gramática for igual ao comprimento da sentença elevado ao quadrado. Caso a razão seja maior que n^2, CE é mais complexo que CYK e, caso a razão seja menor que n^2, então CE é menos complexo que CYK.

c) Construções Estáticas (CE) X Earley

Considerando $O(CE) = nC$ e $O(Earley) = n^3G^2$, as complexidades se igualam quando $nC = n^3G^2$, ou $C/G = n^2G$. Portanto, os algoritmos se equivalem se a razão entre o número de construções e número de regras da gramática for igual ao comprimento da sentença elevado ao quadrado multiplicado pelo número de regras da gramática. Caso a razão seja maior que n^2G, CE é mais complexo que Earley e, caso a razão seja menor que n^2G, então CE é menos complexo que Earley.

d) Construções Estáticas (CE) X Shift-Reduce

Considerando $O(CE) = nC$ e $O(SR) = n^{n+2}G^2$, as complexidades se igualam quando $nC = n^{n+2}G^2$, ou $C/G = n^{n+2}G$. Portanto, os algoritmos se equivalem se a razão entre o número de construções e número de regras da gramática for igual ao comprimento da sentença elevado à potência $n+2$, multiplicado pelo número de regras da gramática. Caso a razão seja maior que $n^{n+2}G$, CE é mais complexo que Shift-Reduce e, caso a razão seja menor que $n^{n+2}G$, então CE é menos complexo que Shift-Reduce.

10 EXPERIMENTOS

Este capítulo apresenta os experimentos realizados para testar o método proposto. Está dividido em três seções que apresentam os resultados específicos de cada técnica e uma seção na qual os resultados gerais são comparados com analisadores do estado da arte para o Português.

10.1 Construções estáticas

O método foi testado em diferentes cenários, visando entender como as diferentes parametrizações, representadas pelo domínio de análise, técnica de etiquetagem morfológica, tamanho das sentenças, similaridade e informações estatísticas influenciam os resultados do analisador sintático.

No primeiro cenário, os testes foram realizados com o corpus CINTIL. As características deste corpus são apresentadas na Tabela 5. São 10733 sentenças anotadas, que geraram 8837 construções, se repetindo em média 1,22 vezes no corpus. O comprimento mínimo das sentenças do CINTIL é de 1 *token* e o comprimento máximo é de 42 *tokens;* 8070 construções possuem até 20 *tokens,* correspondendo a aproximadamente 92% do total.

Tabela 5. Características do corpus CINTIL

	Quantidade
Sentenças	10733
Construções	8837
Média de repetições	1,22
Comprimento mínimo (*tokens*)	1
Comprimento máximo (*tokens*)	42
Comprimento (<=20)	8070

Fonte: Autor

A Figura 39 apresenta a distribuição de construções e sentenças em função do limite de *tokens* das sentenças. Nota-se grande concentração de sentenças menores, de até 5 *tokens* (12% do comprimento da maior sentença do corpus), representando aproximadamente 53% do total das sentenças do corpus e 52% do total de

construções. Aumentando-se o limite para 10 *tokens* (24% do comprimento da maior sentença do corpus), chega-se a 70% das sentenças e 65% das construções. Com o limite em 15 *tokens* (36% do comprimento da maior sentença do corpus), chega-se a 84% do total de sentenças e 81% das construções. Por fim, com o limite em 20 *tokens* (48% do comprimento da maior sentença do corpus), chega-se a 93% das sentenças e 92% das construções.

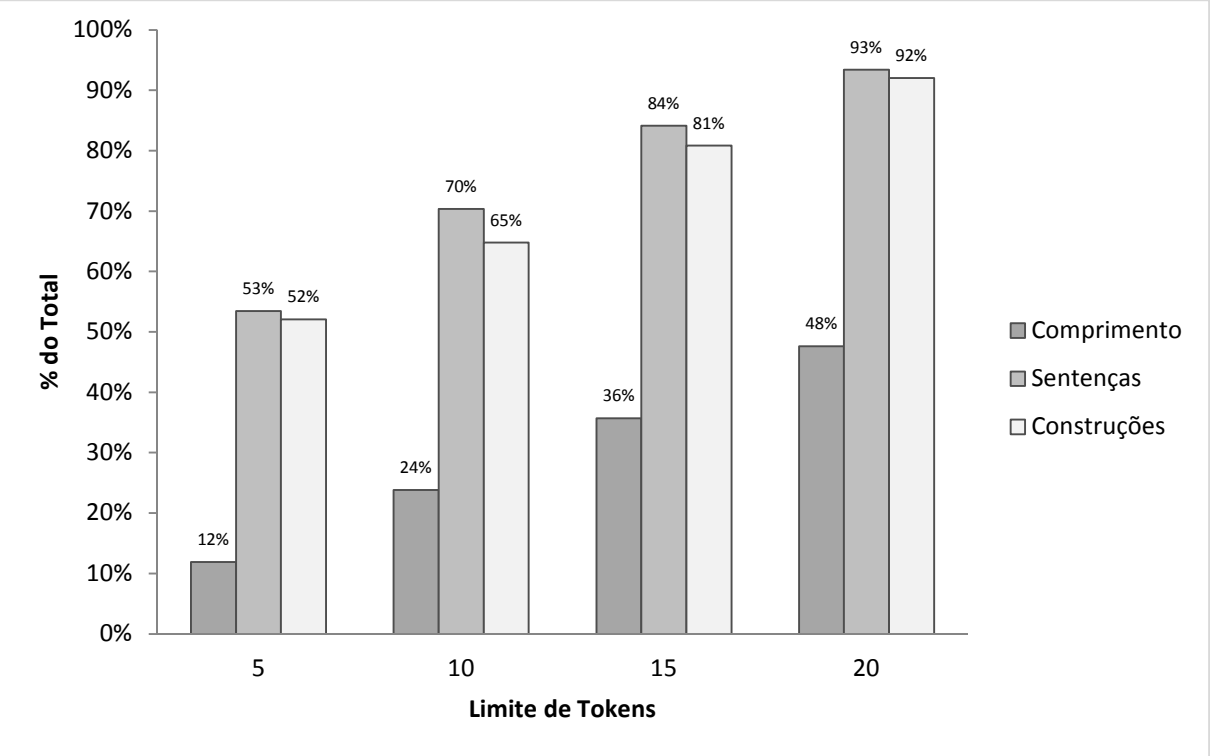


Figura 39 – Distribuição de construções e sentenças
Fonte: Autor

As construções foram geradas a partir do corpus CINTIL, usando o algoritmo 8, apresentado na seção 6.4. Inicialmente os testes foram realizados em 100% do corpus e os etiquetadores morfológicos foram treinados em 90% do corpus. O objetivo foi avaliar como seria o comportamento do analisador com conhecimento pleno das construções do corpus, mas sem total conhecimento das classificações morfológicas. Foram usadas as técnicas Brill, Trigramas e Linguístico, chamadas em sequência, sempre que não eram encontradas construções para as etiquetas informadas pelo etiquetador, em um processo chamado *fallback.* Em seguida, foi

acrescentado um critério estatístico para escolher a melhor construção. Ao invés de usar a primeira construção encontrada, o analisador sintático passou a usar a construção mais frequente do corpus. Por fim, foi aplicada a função de similaridade, de forma a trazer a construção mais próxima da sequência de etiquetas informada pelo etiquetador. A configuração de testes descrita anteriormente também foi aplicada a sentenças com menos de 20 *tokens* para avaliar se haveria algum ganho excluindo sentenças maiores e menos frequentes. Os resultados estão apresentados na Tabela 6. Nota-se que o uso de *fallback* melhora sensivelmente os resultados, que partem de um F-Score de 82,34%, usando o etiquetador de Brill, para F-score de 88,11%, quando os etiquetadores Brill, Trigramas e Linguístico trabalham em sequência. O indicador *Crossbrackets* apresenta pequeno aumento, saindo de 0,23% e chegando a 0,27%. Quando a construção mais frequente é escolhida, o F-score aumenta para 88,37%, com queda no indicador *Crossbrackets* para 0,23%, o que indica redução no cruzamento de parênteses.

Tabela 6. Resultados com etiquetador morfológico treinado em 90% do corpus CINTIL

	Precisão	Cobertura	*CrossBrackets*	F-score
Brill	82,36%	82,33%	0,25%	82,34%
Brill+Trigr	87,52%	87,51%	0,25%	87,52%
Brill+Trigr+Ling	88,20%	88,03%	0,27%	88,11%
Brill+Trigr+Ling+Freq	88,40%	88,34%	0,23%	88,37%
Brill+Trigr+Ling+Freq+Sim	89,56%	89,53%	1,14%	89,54%
Brill+Trigr+Ling+Freq+Sim+<20	90,99%	90,95%	1,13%	90,97%

Fonte: Autor

Por fim, com a aplicação da função de similaridade, o analisador sintático chegou ao melhor F-score, atingindo 89,54%, no entanto, apresentando aumento significativo do índice *Crossbrackets*, que chegou a 1,14%. Isto ocorre porque a saída que identifica construções não encontradas, (S(VAZIO)), cobre a toda a sentença analisada, sem cruzar os constituintes do padrão ouro, não influenciando, portanto, o indicador *crossbrackets*. A partir do momento em que o analisador sintático usa a

função de similaridade, ele passa a gerar uma construção que pode cruzar os constituintes do padrão ouro, afetando o indicador.

Observou-se, também, ganhos de eficiência em frases de até 20 *tokens*. Neste caso, o analisador chegou a um F-score de 90,97% e índice *Crossbrackets* de 1,13%. Em seguida, os mesmos testes foram repetidos, agora com os etiquetadores morfológicos treinados em 100% do corpus, visando avaliar a influência dos etiquetadores no desempenho do analisador. Os resultados estão apresentados na Tabela 7.

Tabela 7. Resultados com etiquetador morfológico treinado em 100% do corpus CINTIL

	Precisão	Cobertura	*CrossBrackets*	F-score
Brill	85,36%	85,34%	0,23%	85,35%
Brill+Trigr	91,44%	91,44%	0,24%	91,44%
Brill+Trigr+Ling	91,89%	91,79%	0,25%	91,84%
Brill+Trigr+Ling+Freq	92,10%	92,11%	0,20%	92,10%
Brill+Trigr+Ling+Freq+Sim	92,83%	92,88%	0,76%	92,86%
Brill+Trigr+Ling+Freq+Sim+<20	94,08%	94,12%	0,74%	94,10%

Fonte: Autor

Nota-se uma melhora sensível em todas as configurações de testes, havendo um ganho de aproximadamente 3,66% no indicador F-score quando o etiquetador Brill é usado isoladamente (de 82,34% para 85,35%) e chegando a 4,23% quando os etiquetadores Brill, Trigramas e Linguístico trabalham em *fallback* (de 88,11% para 91,84%). O ganho cai ligeiramente, para 4,22%, quando a construção mais frequente é escolhida (de 88,37% para 92,10%) e mais acentuadamente, para 3,70%, quando a função de similaridade é aplicada à configuração anterior (de 89,54% para 92,86%), chegando a um ganho mínimo, de 3,44%, quando a configuração anterior é aplicada a sentenças de até 20 *tokens* (de 90,97% para 94,10%). O índice *CrossBrackets* melhorou de maneira mais acentuada. O ganho maior foi na configuração na qual as construções mais frequentes são escolhidas com a aplicação da função de similaridade, quando o índice é 33,33% menor (de 1,14% para 0,76%). Em sentenças com até 20 *tokens* o ganho é de 34,51% (de

1,13% para 0,74%). Lembrando que quanto menor o índice *crossbrackets*, melhor é o resultado da análise. Estes resultados indicam que a acurácia do etiquetador morfológico é significativa nos resultados do analisador sintático de construções estáticas.

O cenário no qual 100% das construções do corpus estão disponíveis para o treinamento reflete uma situação hipotética que visa mostrar o desempenho do analisador em condições ideais. No entanto, conforme descrito na seção 2.9, é fundamental manter a disjunção entre os subconjuntos e treino e teste, deixando os dados de testes inteiramente intocados até que o sistema esteja finalizado e pronto para avaliação. Visando atender a este requisito, os testes foram realizados com o analisador sintático sendo treinado com 90% do corpus CINTIL randomizado e testado nos 10% restantes, repetindo-se o processo por dez vezes (*repeated random subsampling*). Foram utilizados os etiquetadores Brill, Trigramas e Linguistico com opção de *fallback* e regra de maior frequência para a escolha da construção. Os etiquetadores foram treinados em 90% e 100% do total do corpus. Ao final, foram calculadas as médias dos indicadores e os resultados estão apresentados na Tabela 8. Além destes, também foram incluídos os resultados dos testes equivalentes realizados anteriormente.

Tabela 8. Testes com amostragem

	Precisão	Cobertura	*CrossBrackets*	F-score
Brill+Trigr+Ling+Freq	88,40%	88,34%	0,23%	88,37%
Const. Est.+Etiq 90%	29,21%	27,99%	0,53%	28,59%
Brill+Trigr+Ling+Freq	92,10%	92,11%	0,20%	92,10%
Const. Est.+Etiq 100%	30,08%	28,83%	0,52%	29,45%

Fonte: Autor

Nota-se que todos os indicadores estão muito abaixo dos resultados apresentados quando o analisador foi treinado com 100% do corpus. O indicador F-score cai de 88,37% para 28,59%, no cenário em que o etiquetador morfológico é treinado com 90% do corpus e de 92,10% para 29,45%, quando o etiquetador é treinado com 100% do corpus. O mesmo comportamento se observa no índice *CrossBrackets*,

aumentando de 0,23% para 0,53%, no cenário em que o etiquetador morfológico é treinado com 90% do corpus, e aumentando de 0,20% para 0,52%, no caso em que o etiquetador é treinado com 100% do corpus. Tais resultados se explicam pelo baixo índice de repetição das construções no corpus CINTIL, em média 1,22 vezes, fazendo com que o analisador sintático não encontre na base de dados de treino as construções que aparecem na base de dados de testes. O impacto dos etiquetadores morfológicos foi menor, com o indicador F-score aumentando de 28,59 para 29,45%, representando um acréscimo de 3,01%, quando 100% do corpus são utilizados no treinamento.

Em seguida, o método foi testado em outro domínio de informação, representado por um subconjunto randomizado do corpus Bosque, gerado pelo *parser* Linguístico, usando o padrão de etiquetas do corpus CINTIL. Não foram feitos testes com o padrão de etiquetas do corpus Bosque, pois não seria possível comparar os resultados com os obtidos pelos *parsers* do estado da arte para o Português, que foram testados com o Corpus CINTIL. As características deste subconjunto são apresentadas na Tabela 9. São 414 sentenças anotadas, que geraram 406 construções, se repetindo em média 1,02 vezes no corpus. O comprimento mínimo das sentenças do subconjunto do Bosque é de 1 *token* e o comprimento máximo é de 75 *tokens;* 228 construções possuem até 20 *tokens.*

Tabela 9. Características do subconjunto do corpus Bosque

	Quantidade
Sentenças	414
Construções	406
Média de repetições	1,02
Comprimento mínimo (*tokens*)	1
Comprimento máximo (*tokens*)	75
Comprimento (<=20)	228

Fonte: Autor

A Figura 40 apresenta a distribuição de construções e sentenças em função do limite de *tokens* das sentenças. Assim como no corpus CINTIL, nota-se grande

concentração de sentenças menores, de até 5 *tokens* (7% do comprimento da maior sentença do corpus), representando aproximadamente 51% do total das sentenças do corpus e do total de construções. Aumentando-se o limite para 10 *tokens* (13% do comprimento da maior sentença do corpus), chega-se a 56% das sentenças e 55% das construções. Com o limite em 15 *tokens* (20% do comprimento da maior sentença do corpus), chega-se a 62% do total de sentenças e 61% das construções. Por fim, com o limite em 20 *tokens* (27% do comprimento da maior sentença do corpus), chega-se a 71% das sentenças e 70% das construções.

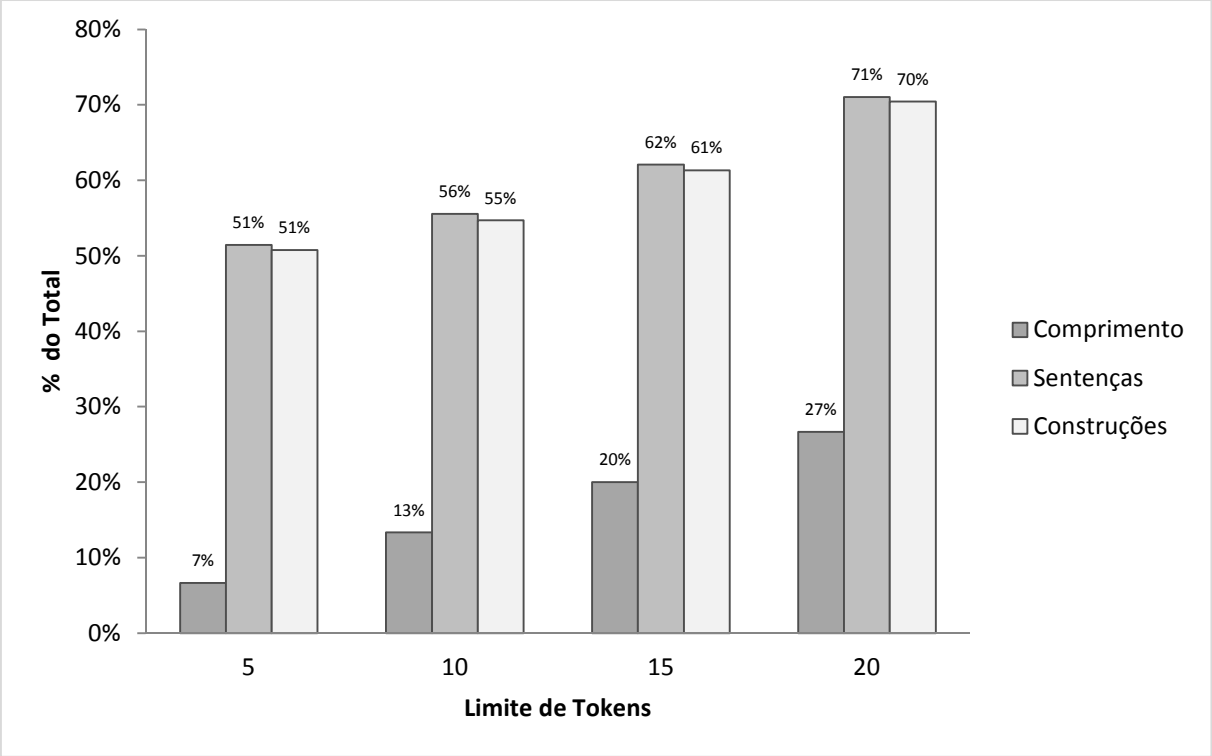


Figura 40 – Distribuição de construções e sentenças

Fonte: Autor

O analisador sintático foi parametrizado para utilizar as técnicas Brill, Trigramas e Linguístico em *fallback*, aplicando o critério de construção mais frequente, gerando as análises com e sem a aplicação da função de similaridade. Os testes foram feitos com o etiquetador morfológico e o analisador sintático treinados em 100% do corpus em todos os cenários, visando avaliar o desempenho do sistema em uma situação

ideal. Inicialmente, foram utilizadas as construções geradas a partir do corpus CINTIL e o etiquetador morfológico também foi treinado no corpus CINTIL. O indicador F-score gerado pelo analisador foi de 6,51%, sem a função de similaridade, e 19,97% quando a função foi aplicada. Percebe-se o impacto significativo quando as construções são geradas a partir do próprio corpus Bosque. Neste caso, ocorre aumento significativo do F-Score, para 43,18% sem a função de similaridade, e 59,18% quando ela é aplicada. Por fim, quando o etiquetador é treinado a partir das sentenças do Bosque, observa-se que o desempenho sobe para 81,97% e 86,79% sem e com a aplicação da função de similaridade. Nota-se, portanto, que o analisador sintático apresenta grande dependência de domínio, tanto por causa da falta de etiquetas morfológicas como de construções sintáticas. O índice *Crossbrackets* também foi impactado pela mudança de domínio, chegando a 20,15% quando o etiquetador e as construções foram originados a partir do corpus CINTIL. A influência das construções também foi observada, fazendo com que o índice caísse para 13,22% quando o etiquetador foi treinado no corpus CINTIL e as construções foram geradas a partir do Bosque. Por fim, quando tanto o etiquetador quanto as construções foram geradas a partir do Bosque, o índice caiu para 4,24%. Os resultados estão apresentados na Tabela 10.

Tabela 10. Resultados dos testes no subconjunto do corpus Bosque

	Precisão	Cobertura	*CrossBrackets*	F-score
Etiq e Constr CINTIL	6,82%	6,23%	0,29%	6,51%
Etiq e Constr CINTIL+Sim	20,38%	19,57%	20,15%	19,97%
Etiq CINTIL e Constr Bosque	43,21%	43,15%	0,00%	43,18%
Etiq CINTIL e Constr Bosque+Sim	59,23%	59,14%	13,22%	59,18%
Etiq Bosque e Constr Bosque	81,97%	81,96%	0,00%	81,97%
Etiq Bosque e Constr Bosque+Sim	86,81%	86,78%	4,24%	86,79%

Fonte: Autor

Mesmo em uma situação ideal, o método de análise sintática por construções estáticas não obteve resultados significativos quando o treino foi realizado em um domínio e o teste foi feito em outro domínio totalmente diferente. Analisando as

sentenças para as quais o analisador sintático não obteve as construções, identificam-se as seguintes causas:

a) O corpus Bosque apresenta muitos fragmentos de sentenças, cujas construções não foram encontradas em construções com o mesmo número de *tokens* no corpus CINTIL. É caso da sentença “Conhecimento de causa”, que requer a construção (S (NP (N token)) (VP (PP (P token) (NP (N token))))). Existem construções geradas a partir do corpus CINTIL que permitem a substituição da sequência (N token) (P token) (N token), porém são construções geradas a partir de sentenças completas, como é o caso da construção (S (NP (N_ (N token) (PP (P token) (NP (N_ (N token) (N_ (CONJ token) (N token)))))))), para as quais a sequência observada é parte de uma estrutura maior;

b) Em alguns casos, o etiquetador morfológico gerou etiquetas incorretas, o que fez com que o analisador não encontrasse a construção, mesmo dispondo dela. É o caso da sentença “Sinto isso.”, etiquetada da seguinte forma: (N Sinto) (DEM isso) (PNT .). O verbo “Sinto” foi incorretamente etiquetado como substantivo (etiqueta N), o que fez com que a construção (S (S (VP (V token) (NP (DEM token)))) (PNT token)) não fosse selecionada;

c) Algumas construções, comuns no corpus Bosque, tais como as que contêm a sequência (N token) (V token) (N token), não foram encontradas no corpus CINTIL. Consequentemente, sentenças como “Menem seria alvo de míssil”, que requerem a construção (S (NP (N token)) (VP (V token) (NP (N_ (N token) (PP (P token) (NP (N token))))))), não puderam ser analisadas.

Os resultados apresentados reforçam a necessidade de um mecanismo para criar as construções dinamicamente, visto o baixo desempenho apresentado pelo analisador quando ele não dispõe de todas as construções ou quando ele troca de domínio. Tal dinamismo deve levar em conta a necessidade de usar componentes que estejam mais bem distribuídos pelas sentenças de análise, o que, como visto anteriormente, não é o caso das construções estáticas obtidas a partir de sentenças do corpus.

10.2 Construções dinâmicas

Assim como com as construções estáticas, aqui também o método foi testado em diferentes cenários, visando entender melhor como as diferentes parametrizações influenciam no resultado final do analisador sintático. Os testes foram realizados com o analisador sintático sendo treinado com 90% do corpus CINTIL randomizado e testado nos 10% restantes, repetindo-se o processo por dez vezes (*repeated random subsampling)*. O etiquetador morfológico foi treinado com 90% e 100% do corpus CINTIL para avaliar a influência das etiquetas morfológicas no resultado da análise sintática.

No primeiro cenário, foi utilizada a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento (Um estudo sobre o impacto do uso da frequência das regras de produção encontra-se disponível no APÊNDICE N).

No segundo cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando o histórico de etiquetas.

No terceiro cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando o histórico de etiquetas e verificação de restrição às regras de produção.

Os resultados estão apresentados na Tabela 11. Nota-se melhora progressiva no F-score, que parte de 69,52% no cenário 1, com o etiquetador treinado com 90% do corpus, para 71,72% no cenário 3, ou seja, um aumento de 3,16%. Quando a comparação é feita com o etiquetador treinado com 100% do corpus, chega-se a resultado semelhante, com o F-score partindo de 70,01%, no cenário 1, para 72,33%, no cenário 3, um aumento de 3,31%.

Tabela 11. Comparativo dos cenários 1, 2 e 3

	Precisão	Cobertura	*CrossBrackets*	F-score
Cenário 1+ Etiq. 90%	65,34%	74,26%	13,33%	69,52%
Cenário 1+ Etiq. 100%	65,80%	74,80%	13,12%	70,01%
Cenário 2+ Etiq. 90%	67,66%	73,62%	13,57%	70,51%
Cenário 2+ Etiq. 100%	68,03%	74,21%	13,33%	70,98%
Cenário 3+ Etiq. 90%	69,06%	74,60%	12,79%	71,72%
Cenário 3+ Etiq. 100%	69,56%	75,33%	12,68%	72,33%

Fonte: Autor

Com relação ao índice *CrossBrackets*, nota-se, também, melhora gradual, com o índice caindo de 13,33%, no cenário 1, para 12,79%, no cenário 3, usando o etiquetador treinado com 90% do corpus. Com o etiquetador treinado com 100% do corpus, chega-se também à mesma conclusão, com o índice caindo de 13,12% para 12,68%. Quando a comparação é feita com os resultados obtidos pelas construções estáticas, nota-se uma melhora significativa, com o indicador F-score saindo de 28,59%, no cenário em que o etiquetador foi treinado com 90% do corpus, para 71,72%, no cenário 3, treinando o etiquetador da mesma forma; e de 29,45%, com o etiquetador treinado com 100% do corpus, para 72,33, no cenário 3, com o mesmo etiquetador. O índice *CrossBrackets*, no entanto apresentou piora no desempenho, com aumento de 0,53% para 12,79% e de 0,52% para 12,68%, respectivamente. Foram executados outros cenários de testes, em caráter exploratório, e os

resultados, assim como as configurações utilizadas, encontram-se disponíveis no APÊNDICE P.

Os resultados apresentados indicam melhora expressiva quando comparados com o método de análise de construções estáticas. No entanto, uma possibilidade a ser investigada é a combinação das duas técnicas, que é apresentada na seção seguinte.

10.3 Combinação de construções

O analisador de construções estáticas foi integrado ao analisador de construções dinâmicas, chamando este último se ele não encontrasse construções prontas para a sentença analisada. Os três cenários de testes do método de construções dinâmicas descritos na seção 10.2 foram repetidos para testar o uso combinado de construções. O etiquetador morfológico foi treinado com 90% do corpus para refletir apenas a situação real da ausência de palavras do corpus de treinamento no texto de análise. Além disso, foi criado outro cenário, com a parametrização abaixo, para testar sentenças curtas, de até dez *tokens*:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento;

e) Sentenças de comprimento de até dez *tokens.*

Os resultados estão apresentados na Tabela 12. Nota-se melhora em todos os cenários. Comparando-se com os resultados apresentados na Tabela 11, percebe-se que no cenário 1, o indicador F-score passou de 69,52% para 72,85%, um acréscimo de 4,79%, e o índice *CrossBrackets* passou de 13,33% para 11,63%, uma redução de 12,75%. Já no cenário 2, o indicador F-score passou de 70,51% para 74,10%, representando um acréscimo de 5,09%, e o índice *CrossBrackets* passou de 13,57% para 12,27%, correspondendo a uma redução de 9,57%.

Tabela 12. Comparativo usando construções dinâmicas e a combinação de estáticas e dinâmicas.

	Precisão	Cobertura	*CrossBrackets*	F-score
Cenário 1+ Etiq. 90%	65,34%	74,26%	13,33%	69,52%
Est+Din1+Et90%	70,04%	75,90%	11,63%	72,85%
Cenário 2+ Etiq. 90%	67,66%	73,62%	13,57%	70,51%
Est+Din2+Et90%	72,16%	76,15%	12,27%	74,10%
Cenário 3+ Etiq. 90%	69,06%	74,60%	12,79%	71,72%
Est+Din3+Etiq.90%	73,24%	77,29%	11,68%	75,21%
Est+Din3+Et90%<=10	77,76%	80,93%	8,59%	79,31%

Fonte: Autor

Finalmente, no cenário 3, o indicador F-score passou de 71,72% para 75,25%, um aumento de 4,92%, e o índice *CrossBrackets* passou de 12,79% para 11,68%, o que representa uma redução de 8,68%. Notam-se, ainda, ganhos maiores quando o analisador processa sentenças curtas, de até 10 *tokens*. Neste cenário, o indicador F-score chegou a 79,31%, representando um ganho de 5,45% quando comparado ao cenário anterior, em que não há restrição ao tamanho da sentença. O índice *CrossBrackets* caiu de 11,68% para 8,59%, representando uma redução de 26,46%.

Conclui-se que a combinação das técnicas produz melhores resultados do que quando usadas isoladamente e o impacto é mais significativo em sentenças menores, de até dez *tokens*, para as quais o analisador já obtém resultados próximos ao estado da arte, no qual o indicador F-score se situa entre 85% e 90% (Silva et al., 2010).

10.4 Comparativo com estado da arte

A Tabela 13 apresenta os resultados obtidos dos diferentes testes apresentados neste trabalho comparados com os resultados dos analisadores do estado da arte Bikel, Stanford e Berkeley, adaptados para o português (Silva et al., 2010).

Tabela 13. Comparativo de analisadores sintáticos

Analisador	F-score
Estático + Etiq 90%	28,59%
Cenário 1+ Etiq. 90%	69,52%
Est+Din1+Et90%	72,85%
Cenário 2+ Etiq. 90%	70,51%
Est+Din2+Et90%	74,10%
Cenário 3+ Etiq. 90%	71,72%
Est+Din3+Etiq.90%	75,21%
Est+Din3+Et90%<=10	79,31%
Bikel	84,97%
Stanford	88,07%
Berkeley	89,33%

Fonte: Autor

Observa-se a evolução das diferentes configurações e, para cadeias de até dez *tokens*, os resultados já se aproximam do estado da arte, que se situa entre F-score de 85% a 90%. É importante ressaltar que os resultados de (Silva et al., 2010) foram obtidos a partir de uma versão anterior do corpus CINTIL, composta por 1.204 sentenças, ao passo que a versão utilizada nesta tese é composta por 10.733 sentenças. Embora os corpora tenham sido produzidos pela mesma gramática, as diferenças entre as sentenças que os compõem podem interferir nos resultados, e a comparação entre os analisadores deve levar em conta esta ressalva.

11 CONCLUSÕES

Este capítulo apresenta as principais contribuições deste trabalho e diretrizes para trabalhos futuros.

11.1 Contribuições

As principais contribuições com o desenvolvimento deste trabalho são as seguintes:

a) Apresentação de um novo método para análise sintática do Português Brasileiro, tomando por base uma vertente recente ainda pouco explorada em Linguística Computacional que é a Gramática de Construções;

b) Resumo dos trabalhos mais significativos de análise sintática, apresentando técnicas, experimentos e resultados, procurando servir de referência para comparação com trabalhos realizados para a Língua Portuguesa;

c) Resumo de trinta anos da pesquisa para Língua Portuguesa, apresentando uma análise comparativa com trabalhos do estado-da-arte, majoritariamente em língua inglesa, procurando identificar lacunas e identificar ideias, técnicas e estratégias que possam auxiliar no desenvolvimento de novos trabalhos para Língua Portuguesa;

d) Aplicação de um dispositivo adaptativo proposto por (NETO, 2000), para controlar a parametrização do contexto de análise dos textos, através do chaveamento das funcionalidades de PLN;

e) Formalização do dispositivo adaptativo proposto por (NETO, 2000), com apresentação da definição, modelo de operação e comportamento de estados e transições a partir do consumo de consumo de uma cadeia;

f) Introdução de técnicas de preparação de textos para análise sintática, incluindo *tokenização*, etiquetagem morfológica e um algoritmo para etiquetagem morfológica;

g) Introdução de três modelos computacionais baseados em Gramática de Construções para análise sintática: Construções Estáticas, Construções Dinâmicas e Combinação de Construções Estáticas e Dinâmicas;

h) Introdução de técnicas para criação de Construções Estáticas a partir do corpus pré-anotados, com a apresentação de um algoritmo e geração de um conjunto de construções a partir do corpus CINTIL (COSTA; BRANCO, 2010) e de outro a partir do corpus Bosque (AFONSO et al., 2002);

i) Análise das características dos conjuntos de construções estáticas criadas a partir dos corpora CINTIL e Bosque, e um estudo comparativo das características das construções, identificando usos e limitações;

j) Apresentação de seis algoritmos para realizar a análise sintática por Construções Estáticas, outros seis para análise por Construções Dinâmicas e um último para análise por Combinação de Construções Estáticas e Dinâmicas;

k) Análise da complexidade algorítmica dos modelos computacionais propostos para análise sintática e comparação com a complexidade computacional de algoritmos clássicos;

l) Testes dos analisadores sintáticos propostos usando técnicas de avaliação padronizadas permitindo um estudo comparativo dos resultados obtidos com analisadores sintáticos do estado da arte;

m) Aplicação prática da proposta de identificação de dependência de contexto, apresentada por Iwai (2000) e Moraes (2006), através do desenvolvimento de dois modelos de *aprendizado de máquina* que atuam como transdutores, identificando restrições linguísticas das sentenças analisadas;

n) Aplicação prática do conceito de etapas preparatórias para análise sintática, apresentado por Miura (2019), através do desenvolvimento de um *tokenizador* e de técnicas de etiquetagem morfológica;

o) Aplicação prática do conceito de padrões oracionais em análise de constituintes, apresentado por Miura (2019), através do desenvolvimento da análise por Construções Estáticas;

p) Disponibilização, mediante solicitação, das seguintes ferramentas na plataforma de hospedagem de código fonte GitHub (PADOVANI, 2021):

- Programa para transformação do corpus CINTIL em um arquivo de sentenças anotadas preparado para desenvolvimento de modelos de análise sintática, junto com um arquivo com as sentenças anotadas do corpus CINTIL e outro com as sentenças anotadas de um subconjunto do corpus Bosque;
- Programas para analisar características das construções estáticas geradas a partir do Corpus CINTIL e do corpus Bosque;
- Etiquetador morfológico do Linguístico (PADOVANI, NETO, 2017) junto com o módulo de integração à biblioteca NLTK (BIRD; KLEIN; LOPER, 2016);
- Analisadores sintáticos que implementam os modelos computacionais propostos nesta tese – Construções Estáticas, Construções Dinâmicas e Combinação de Construções Estáticas e Dinâmicas;
- Programa para identificar restrições linguísticas em sentenças de corpus de treino, usando como referência os símbolos do lado direito das regras de produção e os símbolos do entorno;
- Dois modelos de *aprendizado de máquina*, treinados nas técnicas Naive Bayes e Random Forest, para inferir restrições linguísticas em sentenças ainda não vistas pelo analisador sintático;
- Modelo n-gramas para inferir símbolos do lado esquerdo das regras de produção a partir dos símbolos do lado direito e do histórico de etiquetas já aplicadas;
- Análises sintáticas e respectivos arquivos de referência (*gold standard)* usados em cada cenário de testes pra apuração das métricas de avaliação;
- Tabela de referência cruzada para a localização dos arquivos gerados nos testes e dos programas utilizados para gerá-los.

11.2 Trabalhos futuros

O trabalho apresentado procurou explorar uma forma ainda pouco estudada de análise sintática, através de construções. Conforme observado nos testes realizados, o desempenho do analisador apresentou melhora significativa nos resultados obtidos. No entanto, ainda existem diversas possibilidades de melhorias que podem direcionar pesquisas futuras, como, por exemplo:

a) Restrições linguísticas influenciam a forma como é feita a substituição de símbolos não terminais das sentenças, afetando o desempenho do analisador. No entanto, apenas uma situação foi objetivamente identificada e tratada em uma das configurações do analisador. Podem existir outras restrições. Portanto, uma possível extensão do trabalho poderia investigar como identificar restrições linguísticas que podem influenciar na substituição de símbolos e de que maneira o analisador poderia usar este conhecimento para melhorar seu desempenho;

b) Um autômato poderia ser criado para identificar as restrições às regras de produção a partir da leitura da sentença, de maneira que não apenas as regras de produção candidatas ficassem disponíveis ao término da varredura, mas também a informação de quais delas não tem restrições. Isto evitaria a leitura em uma tabela de apoio, como foi utilizada neste trabalho, e aumentaria a eficiência do reconhecimento. A adaptatividade também poderia ser explorada na construção deste autômato, no sentido de tentar reduzir a complexidade do algoritmo;

c) Outra possibilidade está relacionada aos critérios de escolha de regras de produção. Apenas parte dos parâmetros foi sistematicamente testada e melhorada, mas existem outras possibilidades que podem ser mais exploradas, tais como combinação de análises, uso de técnicas de aprendizado de máquina para escolha da regra para montagem da construção dinâmica, e uso de critérios linguísticos para filtrar regras inválidas no contexto;

d) A métrica *Crossbrackets* foi pouco aprofundada neste trabalho, consequentemente, seria interessante que novos trabalhos procurassem

explorar as possíveis causas do aumento deste indicador e possíveis formas de reduzi-lo;

e) Uma extensão natural deste trabalho seria usar os algoritmos desenvolvidos para criar construções dinâmicas na análise sintática da Gramática Livre de Contexto Adaptativa de Iwai (2000) e usar os mecanismos de identificação de dependência de contexto descritos por Moraes (2006) para representar as restrições linguísticas;

f) É possível também avaliar o uso das construções geradas pelo analisador sintático como entrada de dados para o *parser* de dependências proposto por Miura (2019). Em particular, os padrões oracionais usados pelo autor para criar as árvores de constituição podem ser encarados como um tipo de construção, consequentemente, o desenvolvimento de uma gramática de conversão seria de interesse para fazer a transposição das construções para padrões oracionais, funcionando como um tipo de tradução;

g) Uma linha de pesquisa complementar estaria relacionada ao uso do analisador sintático como ferramenta para o aprendizado de novas construções, que seriam posteriormente incorporadas à sua base de conhecimento, a partir da validação de linguistas, em um modelo supervisionado, provendo, assim, informações para melhoria contínua do desempenho das análises;

h) Ainda nesta linha de raciocínio, as novas construções dos textos analisados poderiam ser incorporadas à base de dados de construções estáticas em um modelo semissupervisionado, visando aumentar a cobertura de análise, enquanto não houver a possibilidade de validação dos resultados por um linguista;

i) O trabalho proposto envolveu a aplicação do método apenas para a Língua Portuguesa, no entanto, seria interessante avaliá-lo no processamento de textos de outras línguas, em particular o inglês, devido ao amplo material de pesquisa disponível, que daria subsídios para comparar concretamente os resultados.

REFERÊNCIAS BIBLIOGRÁFICAS

ABNEY S. Part-of-Speech Tagging and Partial Parsing. In: Young S., Bloothooft G. (eds) **Corpus-Based Methods in Language and Speech Processing. Text, Speech and Language Technology**, v.2, Springer, Dordrecht, 1997. http://link.springer.com/10.1007/978-94-017-1183-8_4

AFONSO, S. et al. Floresta sintá(c)tica: a treebank for Portuguese. In: INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION, LREC 2002, 3. **Proceedings**... 2002.

AGUSTINI, A.; GAMALLO, P.; LOPES, G. P. Assessment of Selection Restrictions Acquisition. In: **Advances in Artificial Intelligence**. [s.l.] Springer Berlin Heidelberg, 2002. p. 407–416.

AHO, A. V; ULLMAN, J. D. **The Theory of Parsing, Translation, and Compiling**. USA: Prentice-Hall, Inc., 1972.

ALENCAR, L. F. et al. JMorpher: A finite-state morphological *parser* in java for android. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2014.

ALUÍSIO, S. et al. An account of the challenge of tagging a reference corpus for brazilian Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings**... 2003.

ANCHIÊTA, R. T.; PARDO, T. A. S. A Rule-Based AMR *Parser* for Portuguese. In: IBERAMIA, **Proceedings**...2018.

ANCHIÊTA, R. T.; PARDO, T. A. S. Towards AMR-BR: A sembank for Brazilian Portuguese language. In: LREC 2018 - INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION,11. **Proceedings…** 2019.

APACHE. **Apache OpenNLP**.

ARMENTANO-OLLER, C. et al. Open-Source Portuguese–Spanish Machine Translation. In: **Lecture Notes in Computer Science**. [s.l.] Springer Berlin Heidelberg, 2006. p. 50–59.

AYCOCK, J.; HORSPOOL, R. N. Practical earley parsing. **Computer Journal**, 2002.

BAHDANAU, D.; CHO, K.; BENGIO, Y. **Neural Machine Translation by Jointly Learning to Align and Translate**, 2014. Disponível em: <http://arxiv.org/abs/1409.0473>. Acesso em: 22 dez. 2020.

BALTIN, M.; BRESNAN, J. The Mental Representation of Grammatical Relations.

Language, 1985.

BAPTISTA, J.; MAMEDE, N.; GOMES, F. Auxiliary verbs and verbal chains in European Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings...** 2010.

BAPTISTA, J.; MAMEDE, N.; MARKOV, I. Integrating verbal idioms into an NLP system. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2014.

BASILIO, M. Formação e Classes de Palavras no Português do Brasil.3ª. ed. (s.l.): Ed.Contexto, 2004.

BICK, E. THE PARSING SYSTEM " PALAVRAS " in a Constraint Grammar Framework. **Automatic Grammatical Analysis of**, 2000.

BICK, E. A constraint grammar based question answering system for Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2003.

BIKEL, D. M. Design of a multi-lingual, parallel-processing statistical parsing engine. In: HLT '02 INTERNATIONAL CONFERENCE ON HUMAN LANGUAGE TECHNOLOGY RESEARCH, 2., **Proceedings**...2002.

BIKEL, D. M. **Intricacies of collins' parsing model Computational Linguistics**, 2004.

BIRD, S.; KLEIN, E.; LOPER, E. **NLTK BookO'Reilly**, 2016.

BLACK, E. et al. A Procedure for Quantitatively Comparing the Syntactic Coverage of {E}nglish Grammars. In: SPEECH AND NATURAL LANGUAGE: PROCEEDINGS OF A WORKSHOP HELD AT PACIFIC GROVE, CALIFORNIA, {F}EBRUARY 19-22, 1991, **Proceedings…** 1991. Disponível em: <https://www.aclweb.org/anthology/H91-1060>. Acesso em: 22 dez. 2020.

BONFANTE, A.; Nunes, M. G.V. Parsing Probabilístico para o Português do Brasil. In: WORKSHOP DE TESES E DISSERTAÇÕES EM INTELIGÊNCIA ARTIFICIAL (WTDIA), 1. **Proceedings…**. Disponível em: <http://www.nilc.icmc.usp.br/nilc/index.php/publications>. Acesso em: 22 dez. 2020.

BOOS, R. et al. brWaC: A WaCky corpus for Brazilian Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2014.

BRANCO, A.; COSTA, F. A Computational Grammar for Deep Linguistic Processing of Portuguese: LX-Gram, version A.4.1, **Technical Report**, University of Lisbon, Department of Informatics, 2014.

BRANCO, A. et al. DeepBankPT and companion Portuguese treebanks in a multilingual collection of treebanks aligned with the penn treebank. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2014.

BRILL, E. Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach. In: WORKSHOP ON HUMAN LANGUAGE TECHNOLOGY, **Proceedings…** 1993.

BUCHHOLZ, S.; MARSI, E. CoNLL-X shared task on multilingual dependency parsing. In: CONFERENCE ON COMPUTATIONAL NATURAL LANGUAGE LEARNING, CONLL-X, 10. **Proceedings…** 2006.

CAI, S.; KNIGHT, K. Smatch: An evaluation metric for semantic feature structures. In: ACL 2013 - Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 51. **Proceedings…** 2013.

CALLISON-BURCH, C. Syntactic constraints on paraphrases extracted from parallel corpora. In: EMNLP 2008 - 2008 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, PROCEEDINGS OF THE CONFERENCE: A MEETING OF SIGDAT, A SPECIAL INTEREST GROUP OF THE ACL, **Proceedings…** 2008.

CARRERAS, X.; COLLINS, M.; KOO, T. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In: CONLL 2008 - CONFERENCE ON COMPUTATIONAL NATURAL LANGUAGE LEARNING, 12. **Proceedings…** 2008.

CETEM. **Cetem Folha. https://www.linguateca.pt/cetenfolha/index_info.html**.

CHARNIAK, E. A Maximum-Entropy-Inspired *Parser*. In: NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS CONFERENCE (NAACL' 2000), 1. **Proceedings…** 2000.

CHEN, D.; MANNING, C. D. A fast and accurate dependency *parser* using neural networks. In: EMNLP 2014 - 2014 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, **Proceedings…** 2014.

CHEN, W. et al. Improving Shift-Reduce Phrase-Structure Parsing with Constituent Boundary Information. **Computational Intelligence**, 2017.

CHOMSKY, N. On certain formal properties of grammars. **Information and Control**, 1959.

CLARK, A; FOX, C.; LAPPIN, S. **The Handbook of Computational Linguistics and Natural Language Processing.** Wiley-Blackwell. 2010, 802 p.

CLARIVATE. Disponível em: <https://clarivate.com/webofsciencegroup/solutions/web-of-science/>. Acesso em: 22 dez.2020.

COLLINS, M. Three generative, lexicalised models for statistical parsing. In: ACL '98/EACL '98: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS AND EIGHTH CONFERENCE OF THE EUROPEAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS,35., **Proceedings…** 1997.

COLLINS, M. Discriminative training methods for hidden Markov models. In: ACL '98/EACL '98: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS AND EIGHTH CONFERENCE OF THE EUROPEAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 35., **Proceedings…**2002.

COLLINS, M. Head-driven statistical models for natural language parsing**.** **Computational Linguistics**, dez. 2003.

COLLINS, M. **Probabilistic Context-Free Grammars (PCFGs)**. Disponível em: <http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf>. Acesso em: 22 dez. 2021.

COLLOBERT, R. Deep learning for efficient discriminative parsing. In: JOURNAL OF MACHINE LEARNING RESEARCH, **Proceedings…** 2011.

COLLOBERT, R. et al. Natural language processing (almost) from scratch. **Journal of Machine Learning Research**, 2011.

COLLOBERT, R.; WESTON, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 25. **Proceedings…** 2008.

COMPUTACIONAL, N. C. do N. I. de L. **Projeto AC/DC: corpo NILC/São Carlos**. Disponível em: <https://www.linguateca.pt/acesso/corpus.php?corpus=SAOCARLOS>. Acesso em: 22 dez. 2020.

CORREIA, J.; BAPTISTA, J.; MAMEDE, N. Syntax deep explorer. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2016.

COSTA, F. et al. Towards incremental parsing of natural language using recursive neural networks. **Applied Intelligence**, 2003.

COSTA, F.; BRANCO, A. LXGram: A deep linguistic processing grammar for Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2010.

COULTHARD, R. J. The application of corpus methodology to translation: the jpedparallel corpus and the pediatrics comparable corpus. Disponível em: <https://repositorio.ufsc.br/handle/123456789/102257>.

CUN, Y. L.; HUANG, F. J. Loss functions for discriminative training of energy-based models. In: AISTATS 2005 - INTERNATIONAL WORKSHOP ON ARTIFICIAL INTELLIGENCE AND STATISTICS, 10. **Proceedings…** 2005.

COSTA, P. B.; KEPLER, F. N. Semi-supervised parsing of Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** Springer Verlag, 2014.

DIAS-DA-SILVA, B. C.; MORAES, H. R. A construção de um Thesaurus eletrônico para o português do Brasil. ALFA: **Revista de Linguística**, v. 47, n. 2, 2003. Disponível em: <http://hdl.handle.net/11449/107537>. Acesso em: 22 dez. 2020.

DEVLIN, J. et al. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, 2019.

DOWTY, D. R.; WALL, R.; PETERS, S. **Introduction to Montague Semantics**. [s.l.] Reidel Pub., 1981.

DOZAT, T.; MANNING, C. D. Deep Biaffine Attention for Neural Dependency Parsing. **CoRR**, v. abs/1611.01734, 2016.

DYER, C. et al. Transition-based dependency parsing with stack long short-term memory. In: ACL-IJCNLP 2015 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS AND THE 7TH INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE ASIAN FEDERATION OF NATURAL LANGUAGE PROCESSING, 53., **Proceedings…** 2015.

EARLEY, J. An Efficient Context-Free Parsing Algorithm. **Communications of the ACM**, 1970.

ELMAN, J. L. Distributed Representations, Simple Recurrent Networks, And Grammatical Structure. **Machine Learning**, 1991.

ESCUDERO, G.; MARQUEZ, L.; RIGAU, G. A comparison between supervised learning algorithms for word sense disambiguation**.** In: CoNLL-2000, ACL, 31–6, **Proceedings…** 2000.

FELLBAUM, C. A semantic network of English: The mother of all WordNets. **Language Resources and Evaluation**, 1998.

FERNANDES, E. R.; RODRIGUES, I. M.; MILIDIÚ, R. L. Portuguese part-of-speech tagging with large margin structure learning. In: BRAZILIAN CONFERENCE ON INTELLIGENT SYSTEMS, BRACIS 2014, **Proceedings…** 2014.

FERRARI, L. **Introdução à linguística cognitiva**. [s.l.] Editora Contexto, 2011.

FINKEL, J. R.; KLEEMAN, A.; MANNING, C. D. Efficient, feature-based, conditional random field parsing. In: ACL-08: HLT - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, THE CONFERENCE, 46., **Proceedings…** 2008.

FONSECA, E. R.; ROSA, G. Mac-Morpho revisited: Towards robust part-of-Speech tagging. In: BRAZILIAN SYMPOSIUM IN INFORMATION AND HUMAN LANGUAGE TECHNOLOGY,9., **Proceedings…** 2013.

FRANCIS, N.; KUCERA, H. **Brown Corpus**.

GALVES, C. The Tycho Brahe Corpus of Historical Portuguese. **Linguistic Variation**, 2018.

GAMALLO, P. et al. LinguaKit: A Big Data-Based Multilingual Tool for Linguistic Analysis and Information Extraction. In: INTERNATIONAL CONFERENCE ON SOCIAL NETWORKS ANALYSIS, MANAGEMENT AND SECURITY, SNAMS, 5., **Proceedings…** 2018,

GAMALLO, P.; LOPES, G. P.; AGUSTINI, A. Clustering syntactic positions with similar semantic requirements. **Computational Linguistics**, 2005.

GILDEA, D. Corpus variation and *parser* performance. In: CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING (EMNLP), 167–20, **Proceedings…**2001.

GILDEA, D.; PALMER, M. The necessity of parsing for predicate argument recognition. In: ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, **Proceedings…** 2001.

GOLDBERG, Y.; NIVRE, J. A Dynamic Oracle for Arc-Eager Dependency Parsing. In: Mumbai, India. Mumbai, India: The COLING 2012 ORGANIZING COMMITTEE, **Proceedings…** 2012. Disponível em: <https://www.aclweb.org/anthology/C12-1059>. Acesso em: 22 dez. 2020.

GONÇALVES, T.; QUARESMA, P. Using linguistic information to classify Portuguese text documents. In: MEXICAN INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE - PROCEEDINGS OF THE SPECIAL SESSION, MICAI 2008, 7., **Proceedings…** 2008.

GRAMMAR, C. **Constraint Grammar**. Disponível em: <http://visl.sdu.dk/constraint_grammar.html>. Acesso em: 22 dez. 2020.

GUNJI, T.; POLLARD, C.; SAG, I. A. Head-Driven Phrase Structure Grammar. **Language**, 1996.

HARRIS, Z. **Mathematical Structures in Language1968**. [s.l: s.n.].

HARTMANN, N. S.; DURAN, M. S.; ALUÍSIO, S. M. Automatic semantic role labeling on non-revised syntactic trees of journalistic texts. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2016.

HE, H.; CHOI, J. D. Establishing Strong Baselines for the New Decade: Sequence Tagging, Syntactic and Semantic Parsing with BERT. In: Proceedings of the 33rd International Florida Artificial Intelligence Research Society Conference, **Proceedings**...2020.

HEMPHILL, C. T.; GODFREY, J. J.; DODDINGTON, G. R. The ATIS spoken language systems pilot corpus. In: HLT '90: WORKSHOP ON SPEECH AND NATURAL LANGUAGE, **Proceedings…** 1990.

HENDERSON, J. Neural network probability estimation for broad coverage parsing. In: EACL '03: CONFERENCE ON EUROPEAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 10., **Proceedings…** 2003.

HENDERSON, J. Discriminative training of a neural network statistical parser. In: ACL '04: ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 42., **Proceedings…** 2004.

HERNAULT, H. et al. HILDA: A Discourse *Parser* Using Support Vector Machine Classification. **Dialogue & Discourse**, 2010.

HINTON, G. E. Training products of experts by minimizing contrastive divergence. **Neural Computation**, 2002.

IWAI, M. K. **Um formalismo gramatical adaptativo para linguagens dependentes de contexto.** 2000. Tese (Doutorado). Universidade de São Paulo, São Paulo, 2000.

JACKENDOFF, R. **X' syntax: A study of phrase structure**. [s.l: s.n.]

JAF, S.; CALDER, C. Deep Learning for Natural Language Parsing. **IEEE Access**, 2019.

JAF, S.; RAMSAY, A. A hybrid approach to parsing natural languages. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN

ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2016.

JAWAHAR, G.; SAGOT, B.; SEDDAH, D. What Does {BERT} Learn about the Structure of Language? In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy. **Anais**... Florence, Italy: Association for Computational Linguistics, 2019.

JOSHI, A. K. An Introduction to Tree Adjoining Grammar. In: **Mathematics of Language**. [s.l: s.n.]

JULIA, R. M. S.; SEABRA, J. R.; SIQUEIRA, I. S. Intelligent parser that automatically generates semantic rules during syntactic and semantic analysis. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, **Proceedings…** 1995.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing. 2nd ed.** USA: Prentice-Hall, 2009.

KAKKONEN, T. **Framework and Resources for Natual Language *Parser* Evaluation.** Academic Dissertation. University of Joensuu. Computer Science. Finland, 2007.

KAPLAN, R. et al. Speed and accuracy in shallow and deep stochastic parsing. In: f HLT-NAACL, **Poceedings...** 2004.

KARLSSON, F. Constraint grammar as a framework for parsing running text. In: COLING '90: CONFERENCE ON COMPUTATIONAL LINGUISTICS, 13., **Proceedings...** 1990.

KLEIN, D.; MANNING, C. D. A generative constituent-context model for improved grammar induction. In: ACL '02: ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 40., **Proceedings…** 2001.

KLEIN, D.; MANNING, C. D. Accurate unlexicalized parsing. In: ACL '03: ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 41., **Proceedings…** 2003a.

KLEIN, D.; MANNING, C. D. Fast exact inference with a factored model for natural language parsing. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, **Proceedings…** 2003b.

KOEHN, P. Europarl : A Parallel Corpus for Statistical Machine Translation. **MT Summit**, 2005.

KOWALTOWSKI, T. et al. **Finite Automata and Efficient Lexicon Implementation**. [s.l: s.n.].

KUHN, M.; JOHNSON, K. **Applied predictive modeling** New York, NYSpringer, , 2013. . Disponível em: < https://link.springer.com/book/10.1007/978-1-4614-6849-3>. Acesso em: 3 mar. 2022.

LADEIRA, A.P. **Processamento de linguagem natural: caracterização da produção científica dos pesquisadores brasileiros**. 2010 Tese (Doutorado) - Universidade Federal de Minas Gerais, 2010.

LEES, R. B.; CHOMSKY, N. Syntactic Structures. **Language**, 1957.

LEÓN, F. S.; SERRANO, A. F. N. Development of a Spanish Version of the Xerox Tagger. **Communications**, 1995.

LIANG, P.; JORDAN, M. I. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 25., **Proceedings…** 2008.

LIMA, T.; COSTA-ABREU, M. A survey on automatic speech recognition systems for Portuguese language and its variations. **Computer Speech and Language**, v.62, 2020.

LIU, L. et al. Agreement on Target-bidirectional Neural Machine Translation. In: CONFERENCE OF THE NORTH {A}MERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, San Diego, California, 2016. **Proceedings...** San Diego, California: Association for Computational Linguistics, 2016. Disponível em: <https://www.aclweb.org/anthology/N16-1046>. Acesso em: 22 dez. 2020. Acesso em: 22 dez. 2020.

LIU, Y.; SUN, M. Contrastive unsupervised word alignment with non-local features. In: THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, **Proceedings…** 2015.

LXCENTER. **LXCenter**. Disponível em: <http://lxcenter.di.fc.ul.pt/services/en/LXServices*Parser*.html>. Acesso em: 22 dez. 2020.

MAAMOURI, M.; BIES, A. Developing an Arabic Treebank: Methods, Guidelines, Procedures, and Tools. In: WORKSHOP ON COMPUTATIONAL APPROACHES TO {A}RABIC SCRIPT-BASED LANGUAGES, Geneva, Switzerland. **Proceedings…** 2004. Geneva, Switzerland: COLING, 2004. Disponível em: <https://www.aclweb.org/anthology/W04-1602>. Acesso em: 22 dez. 2020.

Macao Special Administrative Region Government Portal. Disponível em: <http://www.gov.mo/>. Acesso em: 22 dez. 2020.

MANNING, C. et al. The Stanford CoreNLP Natural Language Processing Toolkit. In: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL

LINGUISTICS: SYSTEM DEMONSTRATIONS, 52., **Proceedings…** 2015.

MARCUS, M. P. Theory of Syntactic Recognition for Natural Language. **Energy Technology Review**, 1979.

MARCUS, M.; SANTORINI, B.; MARCINKIEWICZ, M. Building a Large Annotated Corpus of English: The Penn Treebank. **Computational linguistics - Association for Computational Linguistics (Print)**, 1993.

MARNEFFE, M.-C.; MACCARTNEY, B.; MANNING, C. D. Generating Typed Dependency Parses from Phrase Structure Trees. In: LREC, **Proceedings…** 2006. Disponível em: <http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf>. Acesso em: 22 dez. 2020.

MARQUES, N. M. C.; LOPES, J. G. P. Redes neuronais e um léxico na etiquetagem morfossintática para o estudo da subcategorização verbal. In: SARDINHA, T. B. (Ed.). **A Língua portuguesa no computador**. Campinas - SP - Brasil: Mercado de Letras, 2005. p. 71–89.

MARQUES, T.; BEULS, K. Evaluation strategies for computational construction grammars. In: COLING 2016 - INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS, TECHNICAL PAPERS, 26., **Proceedings…** 2016a.

MARQUES, T.; BEULS, K. A construction grammar approach for pronominal clitics in European Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2016b.

MARRAFA, P. et al. Gathering information from a relational lexical-conceptual database: A natural language question-answering system. (N. Callaos et al., Eds.) In: WORLD MULTI-CONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, VOL V, PROCEEDINGS: COMPUTER SCIENCE AND ENGINEERING, 8., **Proceedings…** 2004.

MARTAPA, L. The Penn Chinese TreeBank : Phrase structure annotation of a large corpus. **Natural Language Engineering** , v. 11 , n. 2 , p. 207 – 238, 2005.

MARTINS, A. et al. Turbo *Parser*s: Dependency Parsing by Approximate Variational Inference. In: CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, Cambridge, MA., 2010. **Proceedings…** Cambridge, MA: Association for Computational Linguistics, 2010. Disponível em: <https://www.aclweb.org/anthology/D10-1004>. Acesso em: 22 dez. 2020.

MARTINS, R.; NUNES, G.; HASEGAWA, R. Curupira: A functional *parser* for Brazilian Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2003.

MAZIERO, E. G.; HIRST, G.; PARDO, T. A. S. Adaptation of discourse parsing models for the Portuguese language. In: BRAZILIAN CONFERENCE ON INTELLIGENT SYSTEMS, BRACIS 2015, **Proceedings…** 2016.

MENCHETTI, S. et al. Wide coverage natural language processing using kernel methods and neural networks for structured data. In: PATTERN RECOGNITION LETTERS, **Proceedings...** 2005.

MENEZES, C.; NETO, J. J. Um Método para a Construção de Etiquetadores Morfológicos Aplicado a Língua Portuguesa, baseado em Autômatos Adaptativos. In: PROPOR, ENCONTRO PARA O PROCESSAMENTO COMPUTACIONAL DE PORTUGUÊS ESCRITO E FALADO, 5., **Anais**... 2000.

MIELENS, J.; SUN, L.; BALDRIDGE, J. Parse imputation for dependency annotations. In: ACL-IJCNLP 2015 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 53., AND INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE ASIAN FEDERATION OF NATURAL LANGUAGE PROCESSING, PROCEEDINGS OF THE CONFERENCE, 7., **Proceedings...** 2015.

MIURA, N.K. **Geração incremental de parsers dependentes de contexto para o português brasileiro**. 2019. 132 p. Tese (Doutorado) – Escola Politécnica da Universidade de São Paulo. São Paulo, 2019.

MOONEY, R. J. Learning for semantic parsing. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings...** 2007.

MORAES, M. **Alguns aspectos de tratamento de dependências de contexto em linguagem natural empregando tecnologia adaptativa**. 2006. Tese (Doutorado) - Universidade de São Paulo, 2006.

NEDERHOF, M.-J.; SATTA, G. Tabular Parsing. **ArXiv**, v. cs.CL/0404, 2004.

NETO, J. J. **Contribuições a metodologia de construção de compiladores**. 1993. Tese (Livre Docência). Universidade de São Paulo, 1993.

NETO, J. J. Adaptive automata for context-dependent languages. **ACM SIGPLAN Notices**, 1994.

NETO, J. J. Adaptive Rule-Driven Devices - General Formulation and Case Study. In: REVISED PAPERS FROM THE INTERNATIONAL CONFERENCE ON IMPLEMENTATION AND APPLICATION OF AUTOMATA, 6., BERLIN, HEIDELBERG. **Proceedings…** 2001, Berlin, Heidelberg: Springer-Verlag, 2001.

NETO, J. J. Solving Complex Problems Efficiently with Adaptive Automata. In:

Revised Papers of the Fifth International Conference on Implementation and Application of Automata, **Anais**...Springer-Verlag, 2000.

NIVRE, J. et al. The CoNLL 2007 shared task on dependency parsing. In: EMNLP-CoNLL 2007 - JOINT CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING AND COMPUTATIONAL NATURAL LANGUAGE LEARNING, **Proceedings…** 2007.

NIVRE, J. et al. Universal Dependencies 2.0 – CoNLL 2017 Shared Task Development and Test Data. **http://universaldependencies.org/conll17/**, 2017.

OLIVEIRA, F. et al. Query translation for cross-language information retrieval by parsing constraint synchronous grammar. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS, ICMLC 2007, 6., **Proceedings…** 2007.

OLIVEIRA, F.; WONG, F.; HONG, I.-S. Systematic Processing of Long Sentences in Rule Based Portuguese-Chinese Machine Translation. (A. Gelbukh, Ed.) In: COMPUTATIONAL LINGUISTICS AND INTELLIGENT TEXT PROCESSING, BERLIN, HEIDELBERG. **Proceedings…** 2010, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

OTERO, P.G.; GONZÁLEZ LÓPEZ, I. A grammatical formalism based on patterns of Part of Speech tags. **International Journal of Corpus Linguistics**, 2011.

OTHERO, G.; KENEDY, E. **Sintaxe, sintaxes: uma introdução**. [s.l.] Editora Contexto, 2015.

PADILHA, E. C.; VICARI, R. M. Morfologia da Língua Portuguesa com Máquinas de Estados Finitos. In: PROPOR, ENCONTRO PARA O PROCESSAMENTO COMPUTACIONAL DE PORTUGUÊS ESCRITO E FALADO, 5., **Anais...** 2000.

PADOVANI, D. Repositório de códigos fonte e dados. Disponível em : https://github.com/djpadovani/tese. Acesso em: 28 out. 2021.

PADOVANI, D.; NETO, J. J. Adaptive Automata Applied to Natural Language Processing. In: PROCEDIA COMPUTER SCIENCE, **Proceedings**...2017.

PADRO, L.; STANILOVSKY, E. FreeLing 3.0: TowardsWider multilinguality. In: INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION, LREC 2012, 8., **Proceedings…** 2012.

PALOMAR, M. et al. An algorithm for anaphora resolution in Spanish texts. **Computational Linguistics**, 2001.

PETROV, S. et al. Learning accurate, compact, and interpretable tree annotation. In: COLING/ACL 2006 - International Conference on Computational Linguistics, 21. and Annual Meeting of the Association for Computational Linguistics, 44., **Proceedings…** 2006.

QUARESMA, P. et al. Tagging and labelling Portuguese modal verbs. In:, **A** LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS). **Proceedings…** 2014.

QUARESMA, P.; PIMENTA RODRIGUES, I. A Natural Language Interface for Information Retrieval on Semantic Web Documents. In: **Advances in Web Intelligence**. [s.l.] Springer Berlin Heidelberg, 2003. p. 142–154.

RAJPURKAR, P. et al. {SQ}u{AD}: 100,000+ Questions for Machine Comprehension of Text. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, Texas. **Anais**... Austin, Texas: Association for Computational Linguistics, 2016.

RAMISCH, C. et al. A hybrid approach for multiword expression identification. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2010.

RICH, E.; KNIGHT, K. **Artificial Intelligence**. 2. ed. [s.l.] McGraw-Hill Higher Education, 1990.

ROSA, J. L. G. A connectionist thematic grid predictor for pre-parsed natural language sentences. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2007.

RUSH, A. M.; COLLINS, M. J. A Tutorial on Dual Decomposition and Lagrangian Relaxation for Inference in Natural Language Processing. **Journal of Artificial Intelligence Research**, v. 45, p. 305–362, 2012. Disponível em: <http://dx.doi.org/10.1613/jair.3680>. Acesso em: 22 dez. 2020.

SAKAI, I. Syntax in universal translation. In: 1961 INTERNATIONAL CONFERENCE ON MACHINE TRANSLATION OF LANGUAGES AND APPLIED LANGUAGE ANALYSIS, **Proceedings…** Teddington, England. II. London: Her Majesty's Stationery Office, 1961. pp. 593–608,

SAGAE, K.; LAVIE, A. *Parser* Combination by Reparsing. In: THE HUMAN LANGUAGE TECHNOLOGY CONFERENCE OF THE {NAACL}, COMPANION VOLUME: SHORT PAPERS, New York City, USA. **Proceedings…**New York City, USA: Association for Computational Linguistics, 2006. Disponível em: <https://www.aclweb.org/anthology/N06-2033>. Acesso em: 22 dez. 2020.

SALKIE, R. F. R. Palmer, Mood and modality. Cambridge: Cambridge University Press, 1986. **Journal of Linguistics**, 1988.

SAMUELSSON, C. Inductive Dependency Parsing Joakim Nivre (Växjö University)

Dordrecht: Springer (Text, speech, and language technology series, edited by Nancy Ide and Jean Véronis, volume 34), 2006, xi+216 pp; hardbound, ISBN 1-4020-4888-2,. **Computational Linguistics**, 2007.

SARDINHA, T. B. Ver a língua portuguesa no computador. In: **A língua portuguesa no computador.** Campinas - SP - Brasil: Mercado de Letras, 2005. p. 7–32.

SCHNEIDER, N. et al. A Framework for (Under)specifying Dependency Syntax without Overloading Annotators. In: LINGUISTIC ANNOTATION WORKSHOP AND INTEROPERABILITY WITH DISCOURSE, 7., Sofia, Bulgaria. **Proceedings…**2013. Sofia, Bulgaria: Association for Computational Linguistics, 2013. Disponível em: <https://www.aclweb.org/anthology/W13-2307>. Acesso em: 22 dez. 2020.

SCHUBERT, L. Computational Linguistics. In: ZALTA, E. N. (Ed.). **The {Stanford} Encyclopedia of Philosophy**. Spring 202 ed. [s.l.] Metaphysics Research Lab, Stanford University, 2020.

SCHWARZ, C. Automatic syntactic analysis of free text. **J. Am. Soc. Inf. Sci.**, v. 41, p. 408–417, 1990.

SHIEBER, S. M. **An introduction to unification-based approaches to grammar**: CSLI lecture notes ; no. 4.Stanford, CACenter for the Study of Language and Information, Stanford University, 1986.

SILVA, J. et al. Out-of-the-box robust parsing of Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2010.

SOAMES, S. et al. Generalized Phrase Structure Grammar. **The Philosophical Review**, 1989.

SOCHER, R. et al. Parsing with compositional vector grammars. In: ACL 2013 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 51., **Proceedings…** 2013.

SORICUT, R.; MARCU, D. Sentence level discourse parsing using syntactic and lexical information. In: NAACL '03: CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS ON HUMAN LANGUAGE TECHNOLOGY, **Proceedings…** 2003.

SPACY. **spaCyFeatures**, 2017.

STEEDMAN, M. Natural Language Processing. **ScholarlyCommons is the University of Pennsylvania - Technical Report CIS**, 1994. Disponível em: <https://repository.upenn.edu/cis_reports/323/>. Acesso em: 22 dez. 2020.

STEELS, L. Introducing Fluid Construction Grammar. [s.l: s.n.], 2011.

STEELS, L. Design Methods for Fluid Construction Grammar. [s.l: s.n.], 2012.

STEELS, L. L. Requirements for Computational Construction Grammars. In: {AAAI} SPRING SYMPOSIA, Stanford University, Palo Alto, California, USA, March 27-29, 2017, **Proceedings…** {AAAI} Press, 2017. Disponível em: <http://aaai.org/ocs/index.php/SSS/SSS17/paper/view/15319>.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to Sequence Learning with Neural Networks. (Z. Ghahramani et al., Eds.) In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, **Proceedings…** Curran Associates, Inc., 2014. Disponível em: <https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf>. Acesso em 22 dez. 2020.

TASKAR, B. et al. Max-margin parsing. In: CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING - EMNLP, 4., **Proceedings…** 2004.

TITOV, I.; HENDERSON, J. Porting statistical *parser*s with data-defined kernels. In: CONFERENCE ON COMPUTATIONAL NATURAL LANGUAGE LEARNING, CoNLL, 10., **Proceedings…** 2006.

VANHALTEREN, H.; et al. (eds). Constraint Grammar, a Language-Independent System for Parsing Unrestricted Text. **International Journal of Corpus Linguistics**, 1996.

VASWANI, A. et al. Attention Is All You Need. **CoRR**, v. abs/1706.03762, 2017. Disponível em: <http://arxiv.org/abs/1706.03762>. Acesso em: 01 mar. 2020.

VICKREY, D.; LIN, C. C. Y.; KOLLER, D. Non-local contrastive objectives. In: ICML 2010 - INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 27., **Proceedings…** 2010.

VINYALS, O. et al. Grammar as a foreign language. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, **Proceedings…** 2015.

VOUTILAINEN, A. NPtool, a detector of English noun phrases. **ACL Workshop On Very Large Corpora Academic And Industrial Perspectives**, 1995.

VOUTILAINEN, A.; HEIKKILA, J. An English constraint grammar (EngCG): a surface-syntactic *parser* of English. (U. Fries, G. Tottie, P. Schneider, Eds.) In: CREATING AND USING ENGLISH LANGUAGE CORPORA, Amsterdam. **Proceedings…** Amsterdam: Editions Rodopi, 1994.

WEISS, D. et al. Structured training for neural network transition-based parsing. In: ACL-IJCNLP 2015 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 53., AND INTERNATIONAL JOINT

CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE ASIAN FEDERATION OF NATURAL LANGUAGE PROCESSING, 7., **Proceedings…** 2015.

What is CETEMPublico - Linguateca. Disponível em: <https://www.linguateca.pt/cetempublico/whatisCETEMP.html>. Acesso em: 22 dez. 2020.

WILLIAMS, A.; NANGIA, N.; BOWMAN, S. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In: Proceedings of the 2018 Conference of the North {A}merican Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), New Orleans, Louisiana. **Anais**... New Orleans, Louisiana: Association for Computational Linguistics, 2018.

WING, B.; BALDRIDGE, J. Adaptation of data and models for probabilistic parsing of Portuguese. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** Springer Verlag, 2006.

WONG, F.; DONG, M.; HU, D. Machine translation using constraint-based synchronous grammar. **Tsinghua Science and Technology**, 2006.

YANG, L.-E. et al. Neural Parse Combination. **Journal of Computer Science and Technology**, v. 32, n. 4, p. 749–757, 2017. Disponível em: <https://doi.org/10.1007/s11390-017-1756-5>. Acesso em: 22 dez. 2020.

YOUNG, T. et al. **Recent trends in deep learning based natural language processing [Review Article]IEEE Computational Intelligence Magazine**, 2018.

YOUNGER, D. H. Recognition and parsing of context-free languages in time n3. **Information and Control**, 1967.

YUAN, Y.; JIANG, Y.; TU, K. Bidirectional Transition-Based Dependency Parsing. In: **AAAI Conference on Artificial Intelligence**, **Proceedings...** 2019.

ZAVAGLIA, C. O papel do léxico na elaboração de ontologias computacionais: do seu resgate à sua disponibilização. In: LINGUÍSTICA IN FOCUS - LÉXICO E MORFOFONOLOGIA: PERSPECTIVAS E ANÁLISES, **Anais**...EDUFU, 2006.

ZENG, X. et al. Lexicon expansion for latent variable grammars. **Pattern Recognition Letters**, 2014.

ZHANG, Y.; CLARK, S. Syntactic processing using the generalized perceptron and beam search. **Computational Linguistics**, 2011.

ZHOU, H. et al. A neural probabilistic structured-prediction model for transition-based dependency parsing. In: ACL-IJCNLP 2015 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 53., AND INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE ASIAN

FEDERATION OF NATURAL LANGUAGE PROCESSING, 7., **Proceedings…** 2015.

ZHU, M. et al. Fast and accurate shift-reduce constituent parsing. In: ACL 2013 - ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 51., **Proceedings…** 2013.

ZILIO, L. et al. Joining forces for Multiword expression identification. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), **Proceedings…** 2016.

APÊNDICE

Neste apêndice são apresentados trechos importantes dos códigos desenvolvidos na tese, assim como arquivos de apoio e estudos complementares.

APÊNDICE A – Autômato Adaptativo Margarida

Definição do autômato adaptativo Margarida:

Seja $M^t = (E^t, R^t, \Sigma, A^t, F^t)$ um autômato adaptativo definido da seguinte forma:

E^t é o conjunto de todos os n estados de M^t ($n \in \mathbb{N}$):

$E^t = \{ e^i \in \mathbb{N} \mid 0 \leq i \leq n \}$

R^t é o conjunto das regras de transição de M^t:

$R^t = \{[(e, \sigma): \alpha(e, \sigma) \rightarrow e] \}$, $e \in E^t$; $\sigma \in \Sigma^t$; $\alpha (e, \sigma) \in F^t$;

Σ é o conjunto de símbolos de entrada aceitos por M^t

$A^t \subseteq E^t$ é o conjunto de estados de aceitação de M^t;

F^t é o conjunto de funções adaptativas de M^t:

$F^t = \{$

$\alpha (x, w)$:

$\{ (y)^*: - [(x, w): \rightarrow x] + [(x, w): \rightarrow y, \beta (y, \sigma)] \}$, $w, \sigma \in \Sigma$; $x, y \in E^t$

$\beta (y, \sigma)$:

$\{ + [(y, \sigma): \alpha (y, \sigma) \rightarrow y] \}$, $\sigma \in \Sigma - \{w\}$; $y \in E^t$

}

Operação do autômato adaptativo Margarida:

Seja $\omega = \omega_0\, \omega_1 \ldots \omega_n$, $n \geq 0$ a cadeia de entrada a ser processada pelo autômato adaptativo.

Situação inicial:

ω_0 é o símbolo da cadeia a ser processado;

$t = 0$;

$M^0 = (E^0, R^0, \Sigma, A^0, F^0)$

Estado inicial = e^0;

Procedimento:

1. Buscar e aplicar a regra: $\{[(e^0, \omega_0): \alpha(e^0, \omega_0) \rightarrow e^0]\}$, $e^0 \in E^0$; $\omega_0 \in \Sigma$; $\alpha(e^0, \omega_0) \in F^0$;
2. Repetir até esgotar cadeia e o autômato adaptativo estar no estado de aceitação.

Teorema: Se e^{i+1} é um estado criado por um autômato adaptativo Margarida ao processar um elemento ω_i de uma cadeia, então as regras de transições de e^{i+1} são as mesmas do estado anterior e^i, com exceção da regra usada para processar ω_i.

Prova:

Vamos provar que para $i \geq 0$,

$$\{[(e^{i+1}, \sigma): \alpha(e^{i+1}, \sigma) \rightarrow e^{i+1}]\} = \{[(e^i, \sigma): \alpha(e^i, \sigma) \rightarrow e^i]\} - [(e^i, \omega_i): \rightarrow e^i, \beta(e^i, \sigma)],\ e^i, e^{i+1} \in E^{i+1}, \sigma \in \Sigma, \alpha(e^i, \sigma), \beta(e^i, \sigma) \in F^i,\ \alpha(e^{i+1}, \sigma) \in F^{i+1} \quad (1)$$

Base da Indução:

Em $i=0$, $\omega_i = \omega_0$, $t = 0$ e $M^t = M^0$:

$M^0 = (E^0, R^0, \Sigma, A^0, F^0)$, onde:

$E^0 = \{e^0\}$,

$R^0 = \{[(e^0, \sigma): \alpha(0,\sigma) \rightarrow e^0] , \sigma \in \Sigma\}$

Σ é o conjunto de símbolos de entrada aceitos por M^t

$A^0 = \{ e^0\}$

$F^0 = \{$

$\alpha (e^0, \omega_0): \{ (e^1)^*: - [(e^0, \omega_0): \alpha(e^0,\omega_0) \rightarrow e^0] + [(e^0, \omega_0): \rightarrow e^1, \beta (e^1, \sigma)]$

$\beta (e^1, \sigma): \{ + [(e^1, \sigma): \alpha (e^1, \sigma) \rightarrow e^1], \sigma \in \Sigma - \{\omega_0\}$ y $\in E^0$

$\}$

Após M^t processar ω_0, i=0, t = 1, $M^t = M^1$;

$M^1 = (E^1, R^1, \Sigma, A^1, F^1)$, onde:

$E^1 = \{e^0, e^1\}$, o estado 1 foi criado pela função adaptativa $\alpha (e^0, \omega_0)$ em M^o;

$\Sigma = \Sigma - \{\omega_0\}$, o elemento $\{\omega_0\}$ não faz parte de Σ, pois a função $\alpha (e^0, \omega_0)$ excluiu a regra de transição que o processava em M^0.

$R^1 = \{$

$[(e^0, \sigma): \alpha(e^0,\sigma) \rightarrow e^0] , \sigma \in \Sigma \cup$

$[(e^0, \omega_0): \rightarrow e^1, \beta (e^1,\sigma)], \sigma \in \Sigma \cup$

$[(e^1, \sigma): \alpha(e^1,\sigma) \rightarrow e^1] , \sigma \in \Sigma$

$\}$

$A^1 = \{ e^0, e^1\}$

$F^1 = \{$

$\alpha (e^1, \omega_1): \{ (e^2)^*: - [(e^1, \omega_1): \alpha(e^1,\omega_1) \rightarrow e^1] + [(e^1, \omega_1): \rightarrow e^2, \beta (e^2, \sigma)]$

$\beta (e^2, \sigma): \{ + [(e^2, \sigma): \rightarrow e^2], \sigma \in \Sigma - \{\omega_1\}$ y $\in E^1$

}

Para provar que a equação (1) é valida para i =0, precisamos mostrar que:

$$\{[(e^1, \sigma): \alpha(e^1, \sigma) \rightarrow e^1] \} = \{[(e^0, \sigma): \alpha(e^0, \sigma) \rightarrow e^0] \} - [(e^0, \omega_0): e^1, \beta (e^1,\sigma)], \ e^0, e^1 \in E^1, \sigma \in \Sigma, \ \alpha (e^0, \sigma) \in F^0, \alpha (e^1, \sigma) \in F^1$$

Observando R^1, notamos que as regras do estado e^0 são $[(e^0, \sigma): \alpha(e^0,\sigma) \rightarrow e^0] \cup [(e^0,\omega_0): \rightarrow e^1, \beta (e^1,\sigma)]$, $\sigma \in \Sigma$. Já as regras do estado e^1 são $[(e^1, \sigma): \alpha(e^1,\sigma) \rightarrow e^1]$, $\sigma \in \Sigma$.

Em R^1, $\Sigma = \Sigma - \{\omega_0\}$, portanto σ nunca assume o valor ω_0.

Logo, a diferença entre as regras dos estados e^1 e e^0 é a regra $[(e^0,\omega_0): \rightarrow e^1, \beta (e^1,\sigma)]$, provando que a equação 1 é valida para i=0.

Hipótese de indução:

$$\text{Se } \{[(e^{i+1}, \sigma): \alpha(e^{i+1}, \sigma) \rightarrow e^{i+1}] \} = \{[(e^i, \sigma): \alpha(e^i, \sigma) \rightarrow e^i] \} - [(e^i, \omega_i): \rightarrow e^{i+1}, \beta (e^{i+1},\sigma)], \ e^i, e^{i+1} \in E^{i+1}, \sigma \in \Sigma, \ \alpha(e^i, \sigma) \in F^i, \beta (e^{i+1},\sigma), \alpha (e^{i+1}, \sigma) \in F^{i+1} \quad (2)$$

É verdadeira para i=k, então ela é verdeira para i=k+1

Passo indutivo:

Assumindo que:

$$\{[(e^{k+1}, \sigma): \alpha(e^{k+1}, \sigma) \rightarrow e^{k+1}] \} = \{[(e^k, \sigma): \alpha(e^k, \sigma) \rightarrow e^k] \} - [(e^k, \omega_k): \rightarrow e^{k+1}, \beta (e^{k+1},\sigma)], \ e^k, e^{k+1} \in E^{k+1}, \sigma \in \Sigma, \ \alpha(e^k, \sigma) \in F^k, \beta (e^{k+1},\sigma), \alpha (e^{k+1}, \sigma) \in F^{k+1} \quad (3)$$

Precisamos provar que:

$$\{[(e^{k+2}, \sigma): \alpha(e^{k+2}, \sigma) \rightarrow e^{k+2}] \} = \{[(e^{k+1}, \sigma): \alpha(e^{k+1}, \sigma) \rightarrow e^{k+1}] \} - [(e^{k+1}, \omega_{k+1}): \rightarrow e^{k+2}, \beta (e^{k+2},\sigma)], \ e^{k+1}, e^{k+2} \in E^{k+12}, \sigma \in \Sigma, \ \alpha(e^{k+1}, \sigma) \in F^{k+1}, \beta (e^{k+2},\sigma), \alpha (e^{k+2}, \sigma) \in F^{k+2} \quad (4)$$

Vamos processar o elemento ω_{k+1} usando a definição do autômato adaptativo Margarida.

O estado do autômato adaptativo antes de processor ω_{k+1} é o seguinte:

$\omega_i = \omega_{k+1,}\ t = k+1,\ M^t = M^{k+1}$

$M^{k+1} = (E^{k+1}, R^{k+1}, \Sigma, A^{k+1}, F^{k+1})$,

onde:

$E^{k+1} = \{ e^i \in \mathbb{N} \mid 0 \leq i \leq k+1 \}$,

$R^{k+1} = \{[(e^{k+1},\sigma): \alpha(e^{k+1},\sigma) \rightarrow e^{k+1}] , \sigma \in \Sigma\}$

Σ é o conjunto de símbolos de entrada aceitos por M^t

$A^{k+1} = \{ e^i \in \mathbb{N} \mid 0 \leq i \leq k+1 \}$

$F^{k+1} = \{$

$\alpha (e^{k+1}, \omega_{k+1})$: { (e^{k+2})*: - [(e^{k+1}, ω_{k+1}): $\alpha(e^{k+1},\omega_{k+1}) \rightarrow e^{k+1}$] + [$(e^{k+1}, \omega_{k+1})$: $\rightarrow e^{k+2}$, $\beta (e^{k+2}, \sigma)$]

$\beta (e^{k+2}, \sigma)$: { + [(e^{k+2}, σ): $\alpha (e^{k+2}, \sigma) \rightarrow e^{k+2}$], $\sigma \in \Sigma - \{\omega_{k+1}\}$ y $\in E^{k+1}$

}

O estado do autômato adaptativo após de processor ω_{k+1}:

$\omega_i = \omega_{k+1,}\ t = k+2$ e $M^t = M^{k+2}$:

$M^{k+2} = (E^{k+2}, R^{k+2}, \Sigma, A^{k+2}, F^{k+2})$, onde:

$E^{k+2} = \{i \in \mathbb{N} \mid 0 \leq i \leq k+2\}$,

$R^{k+2} = \{$

$[(e^{k+1},\sigma): \alpha(e^{k+1},\sigma) \rightarrow e^{k+1}] , \sigma \in \Sigma \cup$

$[(e^{k+1},\omega_{k+1}): \rightarrow e^{k+2}, \beta (e^{k+2},\sigma)], \sigma \in \Sigma \cup$

$[(e^{k+2},\sigma): \alpha(e^{k+2},\sigma) \rightarrow e^{k+2}] , \sigma \in \Sigma$

}

$\Sigma = \Sigma - \{\omega_{k+1}\}$, o elemento $\{\omega_{k+1}\}$ não faz parte de Σ, pois a função $\alpha\ (e^{k+1}, \omega_{k+1})$ excluiu a regra de transição que o processava em M^{k+1}.

$A^{k+2} = \{ e^i \in \mathbb{N} \mid 0 \leq i \leq k+2 \}$

$F^{k+2} = \{$

$\alpha\ (e^{k+2}, \omega_{k+2})$: { (e^{k+3})*: - [(e^{k+2}, ω_{k+2}): $\alpha(e^{k+2},\omega_{k+2}) \rightarrow e^{k+2}$] + [$(e^{k+2}, \omega_{k+2})$: $\rightarrow e^{k+3}$, $\beta\ (e^{k+3}, \sigma)$]

$\beta\ (e^{k+3}, \sigma)$: { + [(e^{k+3}, σ): $\alpha\ (e^{k+3}, \sigma) \rightarrow e^{k+3}$], $\sigma \in \Sigma - \{\omega_{k+2}\}$ $e^{k+3} \in E^{k+3}$

}

As regras de R^{k+2} são:

= {

[(e^{k+1}, σ): $\alpha(e^{k+1},\sigma) \rightarrow e^{k+1}$] , $\sigma \in \Sigma \cup$

[(e^{k+1},ω_{k+1}): $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)$], $\sigma \in \Sigma \cup$

[(e^{k+2},σ): $\alpha(e^{k+2},\sigma) \rightarrow e^{k+2}$] , $\sigma \in \Sigma$

}

Como $\Sigma = \Sigma - \{\omega_{k+1}\}$, as regras de R^{k+2} podem ser reescritas da seguinte forma:

= {

[(e^{k+1}, σ): $\alpha(e^{k+1},\sigma) \rightarrow e^{k+1}$] , $\sigma \in \Sigma - \{\omega_{k+1}\} \cup$

[(e^{k+1},ω_{k+1}): $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)$], $\sigma \in \Sigma - \{\omega_{k+1}\} \cup$

[(e^{k+2}, σ): $\alpha(e^{k+2},\sigma) \rightarrow e^{k+2}$] , $\sigma \in \Sigma - \{\omega_{k+1}\}$

}

Nota-se que as regras dos estados e^{k+1} e e^{k+2} são as mesmas, com exceção da regra [(e^{k+1},ω_{k+1}): $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)$], $\sigma \in \Sigma - \{\omega_{k+1}\}$. Portanto,

$[(e^{k+1}, \sigma)$: $\alpha(e^{k+1},\sigma) \rightarrow e^{k+1}]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$, $\sigma \in \Sigma - \{\omega_{k+1}\}$ - $[(e^{k+2}, \sigma)$: $\alpha(e^{k+2},\sigma) \rightarrow e^{k+2}]$ =

$[(e^{k+1},\omega_{k+1})$: $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$

Ou

- $[(e^{k+2}, \sigma)$: $\alpha(e^{k+2},\sigma) \rightarrow e^{k+2}]$ = - $[(e^{k+1}, \sigma)$: $\alpha(e^{k+1},\sigma) \rightarrow e^{k+1}]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$, $\sigma \in \Sigma - \{\omega_{k+1}\}$ +

$[(e^{k+1},\omega_{k+1})$: $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$

Multiplicando a igualdade por -1, temos:

$[(e^{k+2},\sigma)$: $\alpha(e^{k+2},\sigma) \rightarrow e^{k+2}] = [(e^{k+1},\sigma)$: $\alpha(e^{k+1},\sigma) \rightarrow e^{k+1}]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$, $\sigma \in \Sigma - \{\omega_{k+1}\}$ – $[(e^{k+1},\omega_{k+1})$: $\rightarrow e^{k+2}$, $\beta\ (e^{k+2},\sigma)]$, $\sigma \in \Sigma - \{\omega_{k+1}\}$

O que é equivale à equação (4), como queríamos demonstrar.

APENDICE B - Gabaritos para a geração das regras do etiquetador de Brill

```
templates = [
        brill.Template(brill.Word([0]), brill.Word([1]), brill.Word([2])),
        brill.Template(brill.Word([-1]), brill.Word([0]), brill.Word([1])),
        brill.Template(brill.Word([0]), brill.Word([-1])),
        brill.Template(brill.Word([0]), brill.Word([1])),
        brill.Template(brill.Word([0]), brill.Word([2])),
        brill.Template(brill.Word([0]), brill.Word([-2])),
        brill.Template(brill.Word([1, 2])),
        brill.Template(brill.Word([-2, -1])),
        brill.Template(brill.Word([1, 2, 3])),
        brill.Template(brill.Word([-3, -2, -1])),
        brill.Template(brill.Word([0]), brill.Pos([2])),
        brill.Template(brill.Word([0]), brill.Pos([-2])),
        brill.Template(brill.Word([0]), brill.Pos([1])),
        brill.Template(brill.Word([0]), brill.Pos([-1])),
        brill.Template(brill.Word([0])),
        brill.Template(brill.Word([-2])),
        brill.Template(brill.Word([2])),
        brill.Template(brill.Word([1])),
        brill.Template(brill.Word([-1])),
        brill.Template(brill.Pos([-1]), brill.Pos([1])),
        brill.Template(brill.Pos([1]), brill.Pos([2])),
        brill.Template(brill.Pos([-1]), brill.Pos([-2])),
        brill.Template(brill.Pos([1])),
        brill.Template(brill.Pos([-1])),
        brill.Template(brill.Pos([-2])),
        brill.Template(brill.Pos([2])),
        brill.Template(brill.Pos([1, 2, 3])),
        brill.Template(brill.Pos([1, 2])),
        brill.Template(brill.Pos([-3, -2, -1])),
        brill.Template(brill.Pos([-2, -1])),
        brill.Template(brill.Pos([1]), brill.Word([0]), brill.Word([1])),
        brill.Template(brill.Pos([1]), brill.Word([0]), brill.Word([-1])),
        brill.Template(brill.Pos([-1]), brill.Word([-1]), brill.Word([0])),
        brill.Template(brill.Pos([-1]), brill.Word([0]), brill.Word([1])),
        brill.Template(brill.Pos([-2]), brill.Pos([-1])),
        brill.Template(brill.Pos([1]), brill.Pos([2])),
        brill.Template(brill.Pos([1]), brill.Pos([2]), brill.Word([1]))]
```

Fonte: Autor

APÊNDICE C – Etapa de preparação

Na etapa de preparação é realizada a divisão da sentença em *tokens*. Para isso, são utilizadas funções prontas disponíveis na biblioteca NLTK (BIRD; KLEIN; LOPER, 2016) aplicadas aos corpora CINTIL Treebank (COSTA; BRANCO, 2010) e Bosque (AFONSO et al., 2002). Com relação ao corpus Bosque, foi utilizado o módulo Floresta do nltk.corpus e, com relação ao Corpus CINTIL, foi utilizado o módulo BracketParseCorpusReader do nltk.corpus. A Figura 41 apresenta um exemplo das saídas geradas pelo *tokenizador* para o Corpus Bosque.

```
#8 CF2-1 «Confissões» chega a Portugal

import nltk

from nltk.corpus import floresta

twords =
floresta.tagged_words('C:\\nltk_data\\corpora\\floresta\\bosque\\Bosque_CF_8.0.PennTreebank.ptb')

twords[107:113]

[(u'\xab', u'\xab'),  (u'Confiss\xf5es', u'H:prop:Confiss\xf5es:F_S:'),  (u'\xbb', u'\xbb'),
(u'chega', u'MV:v-fin:chegar:PR_3S_IND:::'),  (u'a', u'H:prp:a::::'),  (u'Portugal',
u'H:prop:Portugal:M_S:::')]
```

Figura 41 – Exemplo das saídas do tokenizador para o Corpus Bosque

Fonte: Autor

APÊNDICE D – Fragmento do conjunto de regras usado pelo etiquetador de Brill

```
(Rule('012', 'ART', 'P', [(Word([0]),'a'), (Pos([1]),'V')]),
 Rule('019', 'CONJ', 'CARD', [(Pos([-1]),'CARD'), (Pos([1]),'CARD')]),
 Rule('013', 'REL', 'C', [(Word([0]),'que'), (Pos([-1]),'V')]),
 Rule('019', 'P', 'ADV', [(Pos([-1]),'ADV'), (Pos([1]),'CARD')]),
 Rule('003', 'P', 'PERCENT', [(Word([0]),'por'), (Word([1]),'cento')]),
 Rule('013', 'ART', 'P', [(Word([0]),'a'), (Pos([-1]),'N')]),
 Rule('007', 'P', 'N', [(Word([-2, -1]),'presidente')]),
 Rule('009', 'ART', 'N', [(Word([-3, -2, -1]),'presidente')]),
 Rule('013', 'ADV', 'A', [(Word([0]),'mesmo'), (Pos([-1]),'ART')]),
 Rule('020', 'ART', 'P', [(Pos([1]),'P'), (Pos([2]),'P')]),
 Rule('007', 'P', 'N', [(Word([-2, -1]),'C\xe2mara')]),
 Rule('020', 'P', 'ADV', [(Pos([1]),'ADV'), (Pos([2]),'PNT')]),
 Rule('020', 'ART', 'CL', [(Pos([1]),'V'), (Pos([2]),'PNT')]),
 Rule('000', 'P', 'CONJ', [(Word([0]),'de_'), (Word([1]),'o'), (Word([2]),'que')]),
 Rule('001', 'ART', 'CONJ', [(Word([-1]),'de_'), (Word([0]),'o'), (Word([1]),'que')]),
 Rule('019', 'REL', 'CONJ', [(Pos([-1]),'CONJ'), (Pos([1]),'ART')]),
 Rule('013', 'P', 'ORD', [(Word([0]),'segundo'), (Pos([-1]),'ART')]),
 Rule('001', 'ART', 'N', [(Word([-1]),'de_'), (Word([0]),'o'), (Word([1]),'Porto')]),
 Rule('002', 'P', 'N', [(Word([0]),'de_'), (Word([-1]),'Minist\xe9rio')]),
 Rule('002', 'P', 'N', [(Word([0]),'de_'), (Word([-1]),'Presidente')]),
 Rule('021', 'ART', 'N', [(Pos([-1]),'N'), (Pos([-2]),'N')]),
 Rule('009', 'P', 'N', [(Word([-3, -2, -1]),'Conselho')]),
 Rule('013', 'A', 'ADV', [(Word([0]),'outro'), (Pos([-1]),'P')]),
 Rule('009', 'P', 'N', [(Word([-3, -2, -1]),'Bolsa')]),
 Rule('009', 'P', 'N', [(Word([-3, -2, -1]),'Festival')]),
 Rule('012', 'ART', 'P', [(Word([0]),'a'), (Pos([1]),'ART')]),
 Rule('023', 'ART', 'A', [(Pos([-1]),'ART')]),
 Rule('013', 'N', 'A', [(Word([0]),'pol\xedtica'), (Pos([-1]),'N')]),
 Rule('013', 'N', 'ADV', [(Word([0]),'lado'), (Pos([-1]),'ADV')]),
 Rule('013', 'REL', 'C', [(Word([0]),'que'), (Pos([-1]),'ADV')]),
 Rule('036', 'P', 'ADV', [(Pos([1]),'ADV'), (Pos([2]),'ADV'), (Word([1]),'outro')]),
 Rule('000', 'P', 'N', [(Word([0]),'de_'), (Word([1]),'o'), (Word([2]),'Porto')]),
 Rule('009', 'P', 'N', [(Word([-3, -2, -1]),'Comiss\xe3o')]),
 Rule('021', 'ART', 'N', [(Pos([-1]),'N'), (Pos([-2]),'N')]),
 Rule('009', 'P', 'N', [(Word([-3, -2, -1]),'secret\xe1rio')]),
 Rule('013', 'A', 'ART', [(Word([0]),'outro'), (Pos([-1]),'V')]),
```

Fonte: Autor

APÊNDICE E – Etiquetas morfológicas.

Etiqueta	**Descrição**
A	Adjective
ADV	Adverb
ART	Article
C	Complementizer
CARD	Cardinal
CL	Clitic
CONJ	Conjunction
D	Determiner
DEM	Demonstrative
ITJ	Interjection
N	Noun
ORD	Ordinal
P	Preposition
PERCENT	Percentage
PNT	Punctuation
POSS	Possessive
PRS	Personal pronoun
QNT	Quantifier
REL	Relative pronoun
V	Verb

Fonte: Adaptado de (BRANCO; COSTA, 2014)

APÊNDICE F – Etiquetas sintáticas.

Etiqueta	Descrição
A_	Adjective sub-phrase constituent
ADV_	Adverb sub-phrase constituent
ADVP	Adverb phrase
AP	Adjective phrase
CARD_	Cardinal sub-phrase constituent
CONJ_	Conjunction sub-phrase constituent
CONJP	Conjunction phrase
CP	Complementizer phrase
ITJ	Interjection
N_	Nominal sub-phrase constituent
NP	Noun phrase
PERCENTP	Percentage phrase
POSS_	Possessive sub-phrase constituent
PP	Preposition phrase
QNT_	Quantifier sub-phrase constituent
S	Sentence
V_	Verb sub-phrase constituent
VP	Verb phrase

Fonte: Adaptado de (BRANCO; COSTA, 2014)

APÊNDICE G – Fragmento de construções sintáticas derivadas do corpus CINTIL.

```
[['(S (V token) (PNT token))', 2],
 ['(S (S (NP (DEM token) (N token)) (VP (V token) (PP (ADV token) (PP (P token)
(NP (N_ (N token) (A token))))))) (PNT token))',
  8],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (PP (ADV token) (PP (P token) (NP
(ART token) (N token)))))) (PNT token))',
  8],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (PP (P token) (NP (ART token) (N
token))))) (PNT token))',
  7],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (PP (ADV token) (PP (P token) (NP
(ART token) (N token)))))) (PNT token))',
  8],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (PP (P token) (V token)))) (PNT
token))',
  6],
 ['(S (S (NP (ART token) (N token)) (VP (V_ (V token) (ADV token)) (PP (P token) (V
token)))) (PNT token))',
  7],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (PP (P token) (VP (ADV token) (V
token))))) (PNT token))',
  7],
 ['(S (S (NP (ART token) (N token)) (V token)) (PNT token))', 4],
 ['(S (S (PP (P token) (NP (PRS token))) (S (NP (ART token) (N token)) (VP (V token)
(A token)))) (PNT token))',
  7],
 ['(S (S (PP (P token) (NP (PRS token))) (S (NP (ART token) (N token)) (VP (V token)
(A token)))) (PNT token))',
  7],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (NP (ART token) (N_ (N token) (A
token))))) (PNT token))',
  7],
 ['(S (S (NP (DEM token) (N token)) (VP (V token) (A token))) (PNT token))',
  5],
 ['(S (S (NP (ART token) (N token)) (VP (V token) (NP (ART token) (N_ (N token) (AP
(ADV token) (A token)))))) (PNT token))',
  8],
 ['(S (S (NP (DEM token) (N token)) (VP (V token) (AP (ADV token) (A token)))) (PNT
token))',
  6],
```

Fonte: Autor

APÊNDICE H – Fragmento de construções sintáticas ordenadas por número de *tokens*.

```
[['(S (NP (N token)))', 1],
 ['(S (V token) (PNT token))', 2],
 ['(S (V token) (NP (CL token)))', 2],
 ['(S (S (VP (A token))) (PNT token))', 2],
 ['(S (S (VP (V token))) (PNT token))', 2],
 ['(S (S (V token)) (PNT token))', 2],
 ['(S (NP (N token)) (PNT token))', 2],
 ['(S (NP (N token)) (V token))', 2],
 ['(S (NP (QNT token)) (PNT token))', 2],
 ['(S (NP (DEM token)) (PNT token))', 2],
 ['(S (S (VP (ADV token))) (PNT token))', 2],
 ['(S (NP (N_ (N token) (N token))))', 2],
 ['(S (S (V token) (NP (N token))) (PNT token))', 3],
 ['(S (S (V token) (NP (POSS token))) (PNT token))', 3],
 ['(S (S (V token) (NP (CARD token))) (PNT token))', 3],
 ['(S (S (VP (V token) (NP (N token)))) (PNT token))', 3],
 ['(S (S (NP (N token)) (V token)) (PNT token))', 3],
 ['(S (S (VP (V token) (A token))) (PNT token))', 3],
 ['(S (S (VP (V token) (NP (CL token)))) (PNT token))', 3],
 ['(S (S (NP (QNT token)) (VP (ADV token))) (PNT token))', 3],
 ['(S (NP (N token)) (VP (V token) (NP (CL token))))', 3],
 ['(S (NP (N_ (N token) (A token))) (PNT token))', 3],
 ['(S (NP (N_ (N S.) (N token))) (VP (V token) (ADV token)))', 3],
 ['(S (NP (N token)) (VP (V token) (NP (N token))))', 3],
 ['(S (NP (N_ (N token) (A token))) (V token))', 3],
 ['(S (S (VP (V token) (V token))) (PNT token))', 3],
 ['(S (NP (ART token) (N token)) (PNT token))', 3],
 ['(S (S (VP (ADV token) (V token))) (PNT token))', 3],
 ['(S (NP (QNT token) (N token)) (PNT token))', 3],
 ['(S (NP (N_ (A token) (N token))) (PNT token))', 3],
 ['(S (S (S (NP (N token))) (ADV token)) (PNT token))', 3],
 ['(S (S (VP (ADV_ (ADV token) (ADV token)))) (PNT token))', 3],
 ['(S (S (S (V token)) (PNT token)) (PNT token))', 3],
 ['(S (NP (ART token) (N token)) (V token))', 3],
 ['(S (NP (N token)) (VP (V token) (V token)))', 3],
 ['(S (NP (N_ (N token) (N token))) (V token))', 3],
 ['(S (S (NP (PRS token)) (V token)) (PNT token))', 3],
 ['(S (NP (CARD token) (N token)) (PNT token))', 3],
 ['(S (S (VP (V token) (ADV token))) (PNT token))', 3],
 ['(S (VP (V token) (NP (N token))) (PNT token))', 3],
 ['(S (S (NP (N token)) (VP (V token))) (PNT token))', 3],
```

Fonte: Autor

APÊNDICE I – Fragmento de construções sintáticas ordenadas por número de *tokens e* frequência.

```
[['(S (NP (N token)))', 1, 2],
 ['(S (S (V token)) (PNT token))', 2, 12],
 ['(S (S (VP (A token))) (PNT token))', 2, 9],
 ['(S (NP (N token)) (PNT token))', 2, 8],
 ['(S (NP (QNT token)) (PNT token))', 2, 3],
 ['(S (V token) (NP (CL token)))', 2, 3],
 ['(S (NP (DEM token)) (PNT token))', 2, 1],
 ['(S (NP (N token)) (V token))', 2, 1],
 ['(S (NP (N_ (N token) (N token))))', 2, 1],
 ['(S (S (VP (ADV token))) (PNT token))', 2, 1],
 ['(S (S (VP (V token))) (PNT token))', 2, 1],
 ['(S (V token) (PNT token))', 2, 1],
 ['(S (NP (N token)) (VP (V token) (NP (N token))))', 3, 31],
 ['(S (S (VP (ADV token) (V token))) (PNT token))', 3, 12],
 ['(S (S (VP (V token) (NP (N token)))) (PNT token))', 3, 12],
 ['(S (S (VP (V token) (A token))) (PNT token))', 3, 11],
 ['(S (S (V token) (NP (N token))) (PNT token))', 3, 8],
 ['(S (NP (ART token) (N token)) (PNT token))', 3, 7],
 ['(S (S (NP (PRS token)) (V token)) (PNT token))', 3, 6],
 ['(S (S (NP (N token)) (V token)) (PNT token))', 3, 5],
 ['(S (S (VP (V token) (ADV token))) (PNT token))', 3, 5],
 ['(S (S (VP (V token) (V token))) (PNT token))', 3, 3],
 ['(S (NP (N_ (N token) (A token))) (PNT token))', 3, 2],
 ['(S (NP (QNT token) (N token)) (PNT token))', 3, 2],
 ['(S (S (NP (QNT token)) (VP (ADV token))) (PNT token))', 3, 2],
 ['(S (S (S (NP (N token))) (ADV token)) (PNT token))', 3, 2],
 ['(S (S (V token) (NP (POSS token))) (PNT token))', 3, 2],
 ['(S (S (VP (ADV_ (ADV token) (ADV token)))) (PNT token))', 3, 2],
 ['(S (S (VP (V token) (NP (CL token)))) (PNT token))', 3, 2],
 ['(S (NP (ART token) (N token)) (V token))', 3, 1],
 ['(S (NP (CARD token) (N token)) (PNT token))', 3, 1],
 ['(S (NP (N token)) (VP (V token) (NP (CL token))))', 3, 1],
 ['(S (NP (N token)) (VP (V token) (V token)))', 3, 1],
 ['(S (NP (N_ (A token) (N token))) (PNT token))', 3, 1],
 ['(S (NP (N_ (N S.) (N token))) (VP (V token) (ADV token)))', 3, 1],
 ['(S (NP (N_ (N token) (A token))) (V token))', 3, 1],
 ['(S (NP (N_ (N token) (N token))) (PNT token))', 3, 1],
 ['(S (NP (N_ (N token) (N token))) (V token))', 3, 1],
 ['(S (S (ADV token) (S (VP (V token)))) (PNT token))', 3, 1],
 ['(S (S (NP (N token)) (VP (V token))) (PNT token))', 3, 1],
 ['(S (S (NP (QNT token)) (V token)) (PNT token))', 3, 1],
```

Fonte: Autor

APÊNDICE J – Fragmento de construções geradas pelo analisador sintático.

(S (S (NP (ART O) (N Manuel)) (VP (ADV hoje) (VP (V_ (ADV não) (V comprou)) (NP (ART um) (N livro))))) (PNT .))
(S (S (NP (ART O) (N Manuel)) (VP (V_ (V_ (ADV não) (V comprou)) (ADV hoje)) (NP (ART um) (N livro)))) (PNT .))
(S (S (NP (ART O) (N Manuel)) (VP (VP (V_ (ADV não) (V comprou)) (NP (ART um) (N livro))) (ADV hoje))) (PNT .))
(S (S (NP (ART O) (N Manuel)) (VP (V desapareceu) (PP (ADV perto) (PP (P de_) (NP (ART o) (N jardim)))))) (PNT .))
(S (S (NP (ART O) (N Manuel)) (VP (V desapareceu) (PP (P em_) (NP (ART o) (N jardim))))) (PNT .))
(S (S (VP (V Ruiu) (NP (ART um) (N_ (N prédio) (A velho))))) (PNT .))
(S (S (V Ruiu) (NP (ART um) (N_ (N prédio) (CP (NP (REL que)) (S (VP (V estava) (ADV ali))))))) (PNT .))
(S (S (VP (VP (V Ruiu) (NP (ART um) (N prédio))) (ADV ali))) (PNT .))
(S (S (S (V Ruiu) (NP (ART um) (N prédio))) (PP (P em_) (NP (ART a) (N cidade)))) (PNT .))
(S (S (VP (VP (V Ruiu) (NP (ART um) (N prédio))) (ADV hoje))) (PNT .))
(S (S (V Chegaram) (NP (ART outras) (N pessoas))) (PNT .))
(S (S (V Chegaram) (NP (ART outras) (N_ (CARD duas) (N pessoas)))) (PNT .))
(S(VAZIO))
(S(VAZIO))
(S (S (V Chegaram) (NP (ART outras) (N_ (A certas) (N pessoas)))) (PNT .))
(S (S (V Chegaram) (NP (ART outras) (N_ (ORD primeiras) (N pessoas)))) (PNT .))
(S (S (V Chegaram) (NP (ART outras) (N_ (N cartas) (POSS minhas)))) (PNT .))
(S(VAZIO))
(S (S (V Chegaram) (NP (ART as) (N_ (A outras) (N cartas)))) (PNT .))
(S(VAZIO))
(S (S (V Chegaram) (NP (D tais) (N pessoas))) (PNT .))
(S(VAZIO))
(S (S (V Desapareceu) (NP (ART o) (N_ (POSS meu) (N carro)))) (PNT .))
(S (S (V Desapareceu) (NP (DEM aquele) (N_ (POSS meu) (N carro)))) (PNT .))
(S (S (V Desapareceu) (NP (POSS meu) (N carro))) (PNT .))
(S (S (V Desapareceu) (NP (DEM aquele) (N_ (N carro) (POSS meu)))) (PNT .))
(S (S (V Desapareceu) (NP (ART um) (N_ (N carro) (POSS meu)))) (PNT .))
(S (S (V Desapareceram) (NP (ART os) (N_ (POSS meus) (N_ (CARD dois) (N carros))))) (PNT .))
(S (S (V Desapareceram) (NP (DEM aqueles) (N_ (POSS meus) (N_ (CARD dois) (N carros))))) (PNT .))
(S (S (V Desapareceram) (NP (DEM aqueles) (N_ (CARD dois) (N_ (N carros) (POSS meus))))) (PNT .))

Fonte: Autor

APÊNDICE K – Fragmento das regras de produção usadas pelo analisador sintático.

N_ -> N N N N N N N N N N N N [0.00012]
AP -> A CONJP [0.00645]
S -> PNT S PNT [0.01429]
S -> V_ AP [0.00021]
AP -> ADV_ A [0.00716]
V_ -> V_ NP [0.00870]
VP -> V CP [0.01065]
S -> V [0.00609]
PP -> PP CP [0.00007]
VP -> V_ [0.00676]
N_ -> N N N N N N N N N N [0.00029]
AP -> CONJ A [0.06877]
VP -> V ADV_ [0.00109]
NP -> PRS [0.01211]
VP -> PNT VP PNT [0.00178]
NP -> N NP [0.00023]
ADVP -> ADV_ ADV [0.02222]
CONJP -> CONJP CONJP [0.00167]
AP -> A AP [0.07808]
N_ -> CARD [0.00098]
AP -> A_ AP [0.00645]
S -> NP V [0.00871]
VP -> CONJP V [0.00020]
N_ -> N PNT [0.00127]
S -> VP PNT [0.00978]
VP -> V ADV [0.01093]
A_ -> ADV A_ [0.01739]
AP -> AP CP [0.00143]
NP -> NP QNT [0.00003]
NP -> QNT N_ [0.01137]
S -> S PNT [0.30826]
AP -> A CP [0.00358]
PP -> P S [0.00007]
V_ -> V NP [0.16225]
AP -> V [0.02937]
S -> PNT NP PNT [0.00060]
N_ -> CARD_ N_ [0.00017]
CONJP -> CONJ CARD [0.00333]
VP -> ADV VP [0.04384]
N_ -> N N N N N N [0.00334]
CP -> ITJ_ CP [0.00072]
ADVP -> ADVP PNT [0.15062]
AP -> CONJ AP [0.01576]

Fonte: Autor

APÊNDICE L – Fragmento da análise gerada através de construções dinâmicas em formato padrão

(S (S (PP (P Em) (NP (N Lisboa))) (S (S (VP (V existem) (S (NP (CARD_ (ADV apenas) (CARD 31))) (PNT ,)))) (PP (P_ (P com) (ADV_ (ADV cerca) (ADV de))) (NP (N_ (N 1200) (N crianças)))))) (PNT .))

(S (S (VP (V Comprei) (NP (CARD_ (ADV_ (ADV cerca) (ADV de)) (CARD_ (CARD vinte) (CARD mil) (CARD e) (CARD dois))) (N computadores)))) (PNT .))

(S (NP (ART A) (N angústia)) (VP (PP (P de_) (NP (ART os) (N_ (N pais) (PP (P perante) (NP (ART os) (N_ (N lobos) (A maus)))))))))

(S (S (CONJP (S (NP (ADV_ (ADV Tradicionalmente) (PNT ,)) (NP (ART a) (N Universidade))))) (S (PP (P de) (NP (N Lisboa))) (S (VP (VP (V diz) (CP (C que) (NP (ART a) (N culpa)))) (VP (V é) (PP (P de_) (NP (ART a) (N Câmara))))))) (PNT .))

Fonte: Autor

APÊNDICE M – Fragmento do corpus gerado através de construções dinâmicas em formato amigável

```
(S
  (S
    (S
      (S
        (NP
          (NP
            (ART O)
            (N_ (A único) (N_ (N acordo) (ADV aparentemente))))
          (A sólido))
        (V é))
      (PP (P sobre) (NP (ART o) (N comando))))
    (S
      (NP
        (NP
          (QNT_
            (AP (A central) (PP (P de_) (NP (ART as) (N forças))))))
        (A estratégicas))))
  (PNT .))

(S
  (NP
    (N_
      (CP
        (NP (N Clubes))
        (S
          (VP
            (V aprovaram)
            (NP
              (N_
                (N sorteio)
                (PP (P de_) (NP (ART os) (N árbitros))))))))))

(S
  (S
    (S (VP (ADV_ (PNT _) (ADV Não)) (V há)))
    (S (NP (N_ (N leis) (PP (P em_) (NP (DEM este) (N país)))))))
  (PNT ?))
```

Fonte: Autor

APÊNDICE N – Impacto do uso da frequência das regras de produção

A Tabela 14 apresenta o resultado do estudo feito para avaliar o impacto do uso da frequência das regras de produção nas escolhas feitas por um *parser* ideal. O estudo tomou por base o corpus CINTIL e considerou que o *parser* trabalha apenas com regras unárias, binárias, ternárias e quaternárias, priorizando, na escolha, as regras de maior abrangência. Isto fez com que parte do corpus fosse desconsiderada da análise, por utilizarem regras mais abrangentes que as quaternárias. A quantidade de sentenças desconsideradas, no entanto, foi pequena, 98, correspondendo a apenas 0,01% do total. Nota-se que, na grande maioria das vezes, o *parser* escolhe as regras que apresentam maior probabilidade, representado 91,40% do total. Em apenas 8,60% dos casos as regras escolhidas não eram as de maior probabilidade.

Tabela 14. Perfil das escolhas realizadas por um *parser* ideal

	Quantidade	Percentual
Escolhas c/menor probabilidade	10.203	8,60%
Escolhas c/maior probabilidade	118.666	91,40%
Sentenças usadas para avaliação	10.635	99,09%
Sentenças desconsideradas	98	0,01%
Sentenças do corpus	10.733	100,00%

Fonte: Autor

APÊNDICE O – Modelo de aprendizado de restrições linguísticas

A situação que caracteriza a restrição linguística é a que identifica um símbolo não terminal entre os símbolos não terminais que compõem o lado direito das regras de produção. Tomando por base a sentença abaixo, verifica-se que a regra de produção NP -> ART N não pode ser aplicada na posição assinalada em vermelho, visto que entre os símbolos não terminais ART e N encontra-se o símbolo N_. Por outro lado, a regra pode ser aplicada na posição indicada em negrito, pois não há nenhum símbolo entre os dois não terminais.

(S (S (NP) (VP (V) (NP (N_ (N) (PP (P) (NP (ART) (N_ (N) (PP (P) (NP **(ART) (N)**)))))))) (PNT))

No entanto, o analisador sintático não dispõe desta informação, pois a cadeia de análise é composta apenas pelos símbolos não terminais informados pelo etiquetador morfológico, os símbolos pré-terminais. Para o analisador sintático, o exemplo acima é representado da seguinte forma:

V N P ART N P ART N PNT

Ou seja, os símbolos não terminais ART e N aparecem duas vezes e o analisador sintático tem que decidir qual substituição deve ser feita. O problema acima pode ser resolvido de uma forma determinística, com a especificação de um linguista especialista na questão, ou através de métodos aproximados, como é o caso das técnicas de aprendizado de máquina. Neste trabalho, optou-se pelo método aproximado, devido à independência de um especialista no assunto e também pelo fato de que os métodos aproximados podem ser treinados através de um corpus pré-anotado.

O modelo a ser desenvolvido deveria utilizar apenas as informações que estão disponíveis para o analisador sintático, ou seja, as regras de produção e os símbolos não terminais que representam a sentença. Optou-se por usar o lado direito das regras de produção e os símbolos dos *tokens* do lado direito e esquerdo da regra, pois eles atendiam aos requisitos. No exemplo acima, o lado direito da regra NP -> ART N é ART N. Como a substituição pode ser realizada em duas posições diferentes da cadeia, existem dois pares de vizinhos. Na primeira posição, os *tokens*

vizinhos ao *tokens* do lado direito da regra são respectivamente P e P; já na segunda posição, os *tokens* vizinhos são P e PNT. Com esta informação o modelo deveria sinalizar ao analisador que a regra não poderia ser aplicada na primeira posição e poderia ser aplicada na segunda.

Para que o modelo fosse desenvolvido, foi desenvolvido um programa de computador que analisou o corpus CINTIL da mesma forma que o analisador de construções dinâmicas faz, selecionando inicialmente as regras unárias, binárias, ternárias e quaternárias, e, em seguida, identificando se o padrão (Símbolo) (Símbolo (Símbolo) era encontrado na cadeia analisada. Se fosse, a regra, assim como os símbolos dos *tokens* vizinhos eram identificados como possuidores de restrições e marcados com a etiqueta 1. Caso não fosse encontrado o padrão, a regra, junto com os símbolos dos *tokens vizinhos* eram identificados com a etiqueta 0.

Foram desenvolvidos dois modelos preditivos. O primeiro usando a técnica Naive Bayes e o segundo usando a técnica Random Forest. O motivo da escolha de dois modelos foi fazer um estudo comparativo do desempenho de cada modelo, de forma que pudesse ser selecionado o melhor. O algoritmo Naive Bayes foi escolhido por ser simples e pelos bons resultados que ele geralmente apresenta em problemas de processamento de linguagem natural. Já o algoritmo Random Forest foi escolhido por ser uma técnica de *ensemble* que reduz a possibilidade de *overfitting* e melhora a precisão.

O corpus preparado com as restrições foi dividido em 80% para treinamento e 20% para teses e os resultados finais estão apresentados na Tabela 15. Nota-se que os dois modelos apresentaram resultados muito bons, com acurácia geral acima de 98%. O modelo desenvolvido com o algoritmo Random Forest, no entanto teve desempenho superior na classificação de regras com restrições, apresentando F-score de 83%, enquanto o algoritmo Naive Bayes apresentou F-score de 76%. Dados os resultados, o modelo desenvolvido com Random Forest foi escolhido para informar ao analisador sintático as restrições sintáticas.

Tabela 15. Resultados dos modelos

	Precisão	Cobertura	F-score	Accuracy Score
Naive Bayes				
0 – Sem restrições	99,00%	99,00%	99,00%	
1 – Com restrições	73,00%	80,00%	76,00%	
Acurácia geral				98,38%
Random Forest				
0 – Sem restrições	99,00%	100,00%	99,00%	
1 – Com restrições	85,00%	80,00%	83,00%	
Acurácia geral				98,92%

Fonte: Autor

APÊNDICE P – Cenários de testes exploratórios

Nos cenários de testes exploratórios, o corpus CINTIL foi randomizado e dividido em 90% para treinamento e 10% para testes, e o etiquetador morfológico foi treinado em 100% do corpus. Cada cenário foi testado uma única vez, sem repetição (*holdout*).

No primeiro cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias e binárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento.

No segundo cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias e ternárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento.

No terceiro cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias, quaternárias e quinarias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento.

No quarto cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias e binárias;

b) Sentido de análise da esquerda para a direita;

c) Sem uso de técnica híbrida;

d) Desambiguação usando a regra mais frequente no corpus de treinamento.

No quinto cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Uso de técnica híbrida. Semente gerada de acordo com a regra de segunda maior abrangência e frequência;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento.

No sexto cenário, os testes foram realizados com a seguinte parametrização:

a) Critério de seleção de regras de produção selecionando regras unárias, binárias, ternárias e quaternárias;

b) Sentido de análise da esquerda para a direita;

c) Uso de técnica híbrida. Semente gerada de acordo com a regra de maior frequência;

d) Desambiguação usando a regra de maior abrangência, seguida pela frequência da regra no corpus de treinamento.

Os resultados estão apresentados na Tabela 16. O melhor resultado foi obtido com o cenário 3, quando foram utilizadas regras quinarias. Em seguida, vieram o cenário 6 e o cenário 2. Observa-se também que no cenário 4, em que apenas a frequência das regras de produção foi usada como critério de escolha de regras de produção, os resultados ficaram muito abaixo dos demais. Reforça-se o caráter exploratório

destes testes, visto que não houve repetição e que o etiquetador morfológico foi treinado com 100% do corpus.

Tabela 16. Comparativo dos cenários de 1 a 6

	Precisão	Cobertura	*CrossBrackets*	F-score
Cenário 1	63,85%	73,54%	13,62%	68,35%
Cenário 2	65,14%	74,22%	13,77%	69,39%
Cenário 3	66,07%	75,13%	12,68%	70,31%
Cenário 4	34,66%	60,19%	11,56%	43,99%
Cenário 5	60,39%	68,20%	10,65%	64,06%
Cenário 6	65,47%	73,93%	12,91%	69,44%

Fonte: Autor

GLOSSÁRIO

Ações Adaptativas: São chamadas, paramétricas ou não, de funções adaptativas.

Acurácia: Em aprendizado de máquina, acurácia é a medida do desempenho geral do modelo, indicando a razão entre o total de acertos e o total de entradas.

Algoritmo não determinístico: É aquele que pode exibir diferentes comportamentos quando processa a mesma entrada de dados.

Algoritmo determinístico: É aquele que exibe sempre o mesmo comportamento quando processa a mesma entrada de dados.

Ambiguidade: A ambiguidade de uma sentença em relação a uma gramática G é o número de maneiras distintas em que ela pode ser gerada por G a partir do símbolo inicial S.

Autômato não determinístico: É aquele que, para uma mesma entrada de dados, permite a transição para mais de um estado.

Autômato determinístico: É aquele que para uma mesma entrada, permite a transição para apenas um estado.

Backoff: Técnica de chamada sequencial de etiquetadores morfológicos em que, caso um etiquetador não encontre a etiqueta para determinado *token,* ele passa a execução para outro, de maneira que este seguinte aplique uma outra técnica de etiquetagem.

Backtrack: Havendo dois ou mais caminhos possíveis para o algoritmo percorrer, um deles é escolhido para prosseguir, e caso o caminho escolhido não tenha sucesso, o algoritmo desfaz o processamento

realizado e retorna ao último ponto de decisão, escolhendo um caminho ainda não experimentado.

Cobertura ou *Recall*: Fração de análises corretas obtidas pelo *parser* com relação a um corpus pré-anotado.

Corpus: Texto pré-anotado com informações sintáticas.

Crossing brackets: Comparação entre o posicionamento dos parênteses das análises geradas pelo *parser* com o posicionamento dos parênteses das análises do corpus.

Desambiguação: Trata-se de retirar o caráter ambíguo de algo, por exemplo, eliminando a possibilidade de caminhos múltiplos a partir de uma determinada situação.

Eficiência: Em análise sintática, é a capacidade de o *parser* analisar textos no menor tempo possível.

Fallback: Processo em que diversas técnicas de etiquetagem morfológica são aplicadas em sequência, sempre que com o resultado da etiquetagem não é possível identificar uma construção sintática.

Features: Em aprendizado de máquina, *features* são variáveis independentes que atuam como entradas do sistema.

F-score ou *F-measure*: Medida de acurácia obtida pela média harmônica entre precisão e cobertura.

Gramática não ambígua: É aquela que gera uma única análise sintática para uma mesma cadeia de entrada.

Gramática Ambígua: É aquela que pode gerar mais de uma análise sintática para a mesma cadeia de entrada.

Língua: É o conjunto organizado de elementos que possibilita a comunicação de um determinado grupo. Exemplo: Língua Portuguesa; Língua Brasileira de Sinais.

Linguagem: É o mecanismo usado para transmitir sentimentos, conceitos e ideias. Exemplo: Linguagem verbal (oral ou escrita); linguagem não verbal (gestos, símbolos, expressões corporais).

Parser: Em processamento de linguagem natural, *parser* é um programa que determina a estrutura sintática de um texto por meio da análise das palavras que o constitui e com base em uma gramática subjacente.

Parser determinístico: É aquele que não faz *backtrack* para analisar uma cadeia.

Parser não determinístico: É aquele que faz *backtrack* para analisar uma cadeia.

Precisão: Fração de análises corretas obtidas pelo *parser* com relação às análises por ele geradas.

Overfitting: Ocorre quando um modelo estatístico se ajusta muito bem aos dados de treinamento, mas se mostra ineficaz para prever novos resultados.

Robustez: Em análise sintática, é a capacidade de o *parser* analisar a maior quantidade possível das sentenças que compõem os textos.

Sintagma: É o nome que se dá às partes de uma oração, denominadas unidades sintáticas. Podem ser constituídos por uma ou mais palavras. Entre seus tipos, estão o sintagma nominal, o sintagma verbal, o sintagma adjetival, o sintagma preposicional e o sintagma adverbial.

Token: Sequência de caracteres em algum documento específico agrupados como uma unidade semântica útil para processamento.