Newton Kiyotaka Miura

Geração incremental de *parsers* dependentes de contexto para o português brasileiro

São Paulo
2019

Newton Kiyotaka Miura

Geração incremental de *parsers* dependentes de contexto para o português brasileiro

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

São Paulo
2019

Newton Kiyotaka Miura

Geração incremental de *parsers* dependentes de contexto para o português brasileiro

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

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. João José Neto

São Paulo
2019

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

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, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação na publicação

Miura, Newton Kiyotaka

Geração incremental de *parsers* dependentes de contexto para o português brasileiro / N. K. Miura – versão corr. – São Paulo, 2019.

132 p.

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

1. Automação das línguas naturais 2. Gramática (Processamento) 3. Sintaxe estrutural I. Universidade de São Paulo, Escola Politécnica, Departamento de Engenharia de Computação e Sistemas Digitais II. t.

AGRADECIMENTOS

À minha querida esposa Ana Cristina que sempre me apoiou e incentivou em todo o caminho que temos percorrido juntos em nossas vidas.

Aos meus filhos Mariana e Felipe, que juntos com a mãe, sempre me proporcionaram incondicional amor, carinho, apoio, incentivo, paciência e compreensão.

Aos meus pais Seiji e Emi, que iniciaram todo este processo e tanto se dedicaram para que eu sempre pudesse prosseguir adiante.

Ao professor João José Neto, que desde o primeiro momento em que nos conhecemos, antes de ser meu professor em disciplinas da graduação, sempre foi um orientador, não apenas acadêmico como foi neste programa de doutorado, mas como um amigo nos momentos alegres e também nos mais desafiadores.

Ao meu amigo Paulo Godoy que sempre acreditou na minha capacidade e com quem compartilhei muitas conquistas e desafios.

Ao meu amigo Paulo Roberto Massa Cereda pelo constante incentivo, amizade e pela generosidade em compartilhar o seu conhecimento comigo.

A todos os amigos da empresa Olos, que investe concretamente nas pessoas que lá trabalham e convivem.

Aos professores Ricardo Luis de Azevedo da Rocha e Jorge Kinoshita pelas conversas incentivadoras e estimulantes.

À Escola Politécnica da Universidade de São Paulo, ao Departamento de Engenharia de Computação e Sistemas Digitais e ao Laboratório de Linguagens e Técnicas Adaptativas, por possibilitarem a realização de pesquisa científica além de dar condições para que mais engenheiros possam contribuir para a construção de uma sociedade melhor.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Uma sentença em linguagem natural em formato textual é uma representação sucinta de um conjunto maior de informações contida numa mensagem que um ser humano deseja transmitir a outro. O entendimento de uma sentença pressupõe a obtenção dessas informações a partir da cadeia de caracteres que forma a sentença. Um dos objetivos do processamento computacional de linguagem natural é organizar e armazenar informações para utilizá-las na recuperação das informações subjacentes codificadas em texto. Esta tese apresenta um método para realizar este processo de decodificação de uma sentença, levantando um grafo que simultaneamente descreve a sua estrutura sintática dada por uma gramática e as relações de dependência entre seus elementos constituintes. A decodificação é realizada por um *parser* gerado automaticamente a partir da especificação de suas regras, permitindo a sua alteração de forma incremental. São apresentados técnicas de projeto e aspectos de implementação.

Palavras-chave: processamento automatizado de línguas naturais, gramática, sintaxe estrutural, dependência contextual

ABSTRACT

A sentence in natural language in text format is a reduced representation of a larger set of information contained in a message that a human wishes to transmit to another. The understanding of a sentence assumes that this information is obtained from the characters that form the sentence. One of the goals of natural language computational processing is to organize and store information for use in retrieving the underlying information encoded in text. This thesis presents a method to decode a sentence, obtaining a graph that simultaneously describes its grammatical phrase structure and the dependency relations between its constituents. The parser is automatically generated from the specification of its rules, allowing it to be incrementally modified. Project techniques and implementation aspects are presented.

Keywords: automated natural language processing, grammar, structural syntax, context dependency

LISTA DE FIGURAS

1.1 Evolução das regras da língua. 15

2.1 Árvore de dependência da sentença “O menino mora no campo.”. 28

3.1 Realidade x Modelo. 32
3.2 Sistema adaptativo. 33
3.3 Processamento de L. 34
3.4 Processamento de um subconjunto de L. 35
3.5 Alterações de gramáticas e reconhecedores. 35

4.1 Equivalência de descrições de linguagens do Tipo 3. 38
4.2 Equivalência de descrições de linguagens do Tipo 2. 39
4.3 Equivalência de descrições de linguagens do Tipo 1. 39
4.4 Visão geral do analisador - Parte 1. 40
4.5 Visão geral do analisador - Parte 2. 41
4.6 Visão geral da análise léxico morfológica. 43
4.7 Análise morfológica de uma sentença. 44
4.8 Entradas e saídas do subsistema de análise morfológica. 45
4.9 Análise da sentença livre de contexto. 46
4.10 Árvore sintática e de dependência. 48
4.11 Descrição parcial do autômato de pilha reconhecedor de sentenças. 49
4.12 Dependências parciais. 50
4.13 Árvore sintática do não terminal `PadVerb`. 51
4.14 Padrões de dependência - `PadVerb`. 51
4.15 Padrões de dependência - `SP` e `SS`. 51
4.16 Levantamento de padrões de dependência. 53
4.17 Tratamento adaptativo de sentenças em ordem inversa. 53

5.1 Notação de Wirth modificada. 56
5.2 Autômato finito determinístico W_M do analisador léxico para a notação de Wirth modificada. 56
5.3 Autômato de pilha estruturado G para reconhecimento de regras de produção de gramática na notação de Wirth modificada. 58
5.4 Visão esquemática da estrutura de dados para gramática. 59
5.5 Exemplo de estrutura de dados de gramática. 59
5.6 Visão geral da estrutura de dados para uma regra de produção. 60
5.7 Exemplo de estrutura de dados para uma regra de produção. 60
5.8 Relação entre elementos da expressão de uma regra com a lista de não terminais. 61
5.9 Estrutura de dados associadas a uma transição $trans_{out}$. 62
5.10 Ações semânticas associadas a uma transição $trans_{out}$. 63
5.11 Etapa 2 – Tipos das variáveis globais. 64
5.12 Eliminação de transições em vazio - Caso 1. 66
5.13 Eliminação de transições em vazio - Caso 2. 67
5.14 Exemplo de eliminação parcial de não-determinismo. 68

5.15 Simplificação de submáquina de reconhecimento de sintagma nominal. 69
5.16 Configuração parcial do autômato adaptativo coletor de nomes. 70
5.17 Autômato finito determinístico inicial L_{ini} do analisador léxico-morfológico. . . 72
5.18 Exemplo de identificação de padrões de relações de dependência. 82

6.1 Posições básicas da oração. 85
6.2 Fórmula geral para oração `S` na ordem direta. 86
6.3 Gramática livre de contexto para períodos simples. 87
6.4 Gramática livre de contexto de sintagmas básicos. 87
6.5 Gramática completa de Luft (2002) na notação de Wirth modificada. 88
6.6 Regra de produção e padrão de relações de dependência de `PadNom_2` 88
6.7 Exemplo de árvore sintática de saída de período simples. 91
6.8 Exemplo de árvore sintática de saída de período composto. 91
6.9 Exemplo de saída de relações de dependência de uma sentença de período simples. 92
6.10 Exemplo de saída de relações de dependência de uma sentença de período composto. 92

A.1 Etapa 1 - Cria nova submáquina. 104
A.2 Etapa 1 – Nova regra de produção. 104
A.3 Etapa 1 – Finaliza regra de produção. 105
A.4 Etapa 1 – Nova transição com não terminal. 105
A.5 Etapa 1 – Nova transição com terminal 105
A.6 Etapa 1 – Nova transição com metassímbolo ϵ 105
A.7 Etapa 2 – Cria nova submáquina. 106
A.8 Etapa 2 – Nova regra de produção. 106
A.9 Etapa 2 – Finaliza regra de produção. 107
A.10 Etapa 2 – Nova transição com não terminal, terminal ou ϵ. 107
A.11 Etapa 2 – Novo escopo. 107
A.12 Etapa 2 – Fecha escopo. 107
A.13 Etapa 2 – Adiciona opção. 108
A.14 Etapa 2 – Adiciona reverso. 108
A.15 Etapa 2 – Fecha escopo reverso. 108
A.16 Relações para auxílio na anotação. 109
A.17 Relações de dependência principais. 110
A.18 Relações de dependência nominal. 111
A.19 Relações de dependência secundárias. 112
A.20 Relações de dependência de coordenação. 113
A.21 Relações de dependência de palavras funcionais. 113
A.22 Relações de dependência principais entre orações. 114
A.23 Relações de dependência em expressões multipalavra. 115
A.24 Relação de dependência especial. 115
A.25 Relações de dependência fracas. 115
A.26 Regras de produção e padrões de relações de dependência - 1. 116
A.27 Regras de produção e padrões de relações de dependência - 2. 117
A.28 Regras de produção e padrões de relações de dependência - 3. 118
A.29 Regras de produção e padrões de relações de dependência - 4. 119
A.30 Regras de produção e padrões de relações de dependência - 5. 120
A.31 Regras de produção e padrões de relações de dependência - 6. 121

A.32 Regras de produção e padrões de relações de dependência - 7. 122
A.33 Exemplo de processamento - 1. 123
A.34 Exemplo de processamento - 2. 124
A.35 Exemplo de processamento - 3. 125
A.36 Exemplo de processamento - 4. 126
A.37 Exemplo de processamento - 5. 127
A.38 Exemplo de processamento - 6. 128
A.39 Exemplo de processamento - 7. 129
A.40 Exemplo de processamento - 8. 130
A.41 Exemplo de processamento - 9. 131
A.42 Exemplo de processamento - 10. 132

LISTA DE TABELAS

2.1 Classes de agrupamentos e membros constituintes envolvidas na comunicação. 23

5.1 Símbolos a serem consumidos nas transições do autômato finito determinístico W_M para análise de descrição gramatical em Wirth modificada. 56
5.2 Classes gramaticais do analisador léxico proposto para a notação de Wirth modificada. 57
5.3 Ações semânticas associadas às transições do autômato de pilha estruturado G (Figura 5.3). 58
5.4 Regras de mapeamento de rótulos e saídas. 60
5.5 Etapa 2 - Variáveis globais. 64
5.6 Condições para eliminação de transição em vazio. 65
5.7 Símbolos a serem consumidos nas transições do autômato finito determinístico L_{ini}. 72
5.8 Regras de mapeamento de rótulos e saídas. 73
5.9 Dados de relações de dependência associados às palavras da sentença de entrada. 75
5.10 Estrutura de dados de um padrão de dependência `DepPattern`. 76
5.11 Estrutura de dados de um elemento da lista de constituintes. 76
5.12 Exemplos de estruturas de dados de elementos da lista de constituintes. 77
5.13 Mapeamento da saída do transdutor livre de contexto para identificação de padrão de relações de dependências. 78

6.1 Classes gramaticais do corpus Bosque. 84
6.2 Classes gramaticais no corpus de Aires (2000). 85
6.3 Constituintes de sintagmas. 86
6.4 Não terminais da gramática. 89
6.5 Relações de dependências do português brasileiro. 90

LISTA DE ALGORITMOS

5.1 Ação semântica de tratamento de transição com símbolo terminal 78
5.2 Ação semântica de tratamento do rótulo ϵ 78
5.3 Ação semântica de tratamento de rótulo do tipo terminal 79
5.4 Ação semântica de tratamento do rótulo "[" 79
5.5 Ação semântica de tratamento do rótulo "]" 79
5.6 Ação semântica de tratamento de rótulo do tipo não terminal com recursão à esquerda na regra de produção 80
5.7 Ação semântica de tratamento de rótulo do tipo não terminal com recursão à direita na regra de produção 80
5.8 Ação semântica de tratamento de rótulo do tipo não terminal sem recursão na regra de produção 80
5.9 Tratamento de padrão de dependência associado a não terminal 81

SUMÁRIO

1 Introdução **13**
1.1 Motivação . . . 13
1.2 Justificativa . . . 16
1.3 Objetivos . . . 17
1.4 Organização do Trabalho . . . 17

2 Conceitos **18**
2.1 Definições Preliminares . . . 18
2.2 Processamento computacional de linguagem natural . . . 20
2.2.1 Ambiguidade e não determinismo em PLN . . . 20
2.3 Análise livre de contexto . . . 22
2.3.1 Construção de Compiladores . . . 23
2.4 Análise de dependências . . . 26
2.4.1 Dependências universais . . . 28
2.4.2 Dependências em português . . . 28

3 Proposta **31**
3.1 Introdução . . . 31
3.2 Modelo de processamento de linguagem natural . . . 31
3.2.1 Modelos . . . 32
3.2.2 Arquitetura Adaptativa . . . 34
3.3 Sumário . . . 36

4 Projeto **37**
4.1 Introdução . . . 37
4.2 Visão geral . . . 37
4.2.1 Organização hierárquica do reconhecimento de uma linguagem . . . 37
4.2.2 Construção de analisadores de uma linguagem . . . 40
4.3 Análise léxico morfológica . . . 43
4.3.1 Pré-processamento da cadeia de entrada . . . 43
4.3.2 Classificação da classe gramatical . . . 44
4.4 Análise livre de contexto . . . 46
4.4.1 Geração de um transdutor . . . 46
4.4.2 *Parsing* . . . 47
4.5 Análise de dependências de contexto . . . 47
4.5.1 Padrões de relações de dependência de contexto . . . 47
4.5.2 Levantamento de padrões . . . 52
4.5.3 Tratamento de ordem inversa . . . 52
4.6 Sumário . . . 54

5 Aspectos de implementação **55**
5.1 Introdução . . . 55

5.2 Geração de analisador sintático livre de contexto . . . 55
5.2.1 Analisador léxico para gramáticas em notação de Wirth modificada 55
5.2.2 Analisador sintático para gramáticas em notação de Wirth modificada 57
5.3 Construção do analisador sintático livre de contexto para português brasileiro 70
5.3.1 Analisador léxico para português brasileiro . . . 70
5.3.2 Analisador sintático para português brasileiro . . . 72
5.4 Análise de dependência de contexto . . . 75
5.5 Sumário . . . 82

6 Ensaios **83**
6.1 Introdução . . . 83
6.2 Caracterização dos ensaios . . . 83
6.2.1 Análise léxico-morfológica . . . 83
6.2.2 Análise livre de contexto . . . 84
6.2.3 Análise de dependências . . . 87
6.3 Resultados . . . 89
6.4 Sumário . . . 93

7 Conclusão **94**
7.1 Discussões . . . 94
7.2 Contribuições . . . 95
7.3 Trabalhos Futuros . . . 96

Referências bibliográficas **98**

A Apêndice **104**
A.1 Ações semânticas associadas ao reconhecedor de gramática em Wirth modificada - Etapas 1 e 2 . . . 104
A.1.1 Ações semânticas da Etapa 1 . . . 104
A.1.2 Ações semânticas da Etapa 2 . . . 106
A.2 Relações de dependência universais para o português . . . 109
A.3 Padrões de relações de dependência . . . 116
A.4 Exemplos de processamento . . . 123

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÃO

Ao longo do tempo pesquisadores têm sido motivados a criar sistemas automáticos que artificialmente reproduzam características do ser humano com o objetivo de substituí-lo na realização de atividades. Uma destas características humanas é a capacidade de se comunicar. De forma simplificada o processo de comunicação consiste na transmissão de uma mensagem de um emissor para um receptor. A mensagem, representada por meio de um código conhecido por ambas as partes, é enviada pelo emissor e trafega por um canal até chegar ao receptor, que usa seus sentidos para percebê-la. Um exemplo é uma conversa falada em que o emissor usa o seu sistema vocal para modular sons e emitir a mensagem numa língua que o interlocutor entenda. O receptor usa a audição para perceber os sons daquilo que foi falado pelo emissor e interpreta a mensagem. A automatização de partes desta comunicação humana pode ser realizada empregando-se o processamento computacional de linguagem natural (PLN). A pesquisa desenvolvida neste trabalho insere-se nesta área de PLN cuja aplicação permite que o ser humano possa interagir com sistemas automatizados utilizando a sua forma espontânea de comunicação normalmente empregado na interação com outros humanos.

O uso de PLN abre possibilidades para o desenvolvimento de inúmeras aplicações relevantes para a humanidade, que aumenta mais ainda a motivação dos pesquisadores desta área. Por exemplo, considerando a situação demográfica atual, em que há um aumento significativo da expectativa de vida combinado a um declínio do crescimento populacional, sistemas que auxiliam o idoso a manter a qualidade de vida mesmo que tenha a sua capacidade motora diminuída em decorrência de sua idade é um tipo de aplicação que se torna cada vez mais necessária. Aplicações deste tipo, que são sistemas eletromecânicos controlados por computadores, exigem uma interação humano-computador que demanda o PLN. É necessário o desenvolvimento de sistemas automatizados para analisar e interpretar sentenças proferidas em linguagem natural para que ações sejam executadas como resposta, seja na forma de algum movimento físico, ou o fornecimento de informações. Sistemas menos críticos de comunicação automatizada entre clientes e empresas estão já disponíveis atualmente. Em alguns atendimentos a clientes realizados por meio da troca de mensagens escritas, o representante da empresa é substituído por um "*chat bots*", que são programas que simulam conversas de um ser humano, e que vem sendo pesquisado há bastante tempo (WEIZENBAUM, 1966; WINOGRAD, 1972).

Outro fenômeno que temos observado é um grande aumento de textos disponibilizados

em formato digital graças à disseminação de equipamentos com capacidade de processamento computacional a um custo acessível para grupos populacionais cada vez maiores. Neste cenário, torna-se importante para essas pessoas que o processo de busca de informações relevantes seja realizado de forma automatizada em bases de informações cada vez maiores, que humanamente seria impossível de se analisar.

São inúmeras as aplicações de PLN como o reconhecimento e síntese de voz, tradução de uma linguagem para outra, busca de informações, sumarização/categorização de textos (JURAFSKY; MARTIN, 2008). Nos sistemas computacionais que implementam estas aplicações, em algum ponto a linguagem natural é representada por uma sequência de símbolos refletindo a língua escrita. Quando a informação não está neste formato textual, são utilizados sistemas para fazer conversões, como no reconhecimento de voz em que as ondas sonoras são transformadas em sinais elétricos, digitalizadas, convertidas para uma sequência de fonemas e depois para palavras em texto. Nesta tese a linguagem natural é tratada neste formato de texto.

No modelo de comunicação acima, o receptor analisa a mensagem para entender o seu significado. Uma forma de modelar a habilidade cognitiva humana é realizar a análise da mensagem, um texto, identificando a sua estrutura, os seus componentes e as regras que definem o relacionamento entre eles. Segundo Lieberman et al. (2007), a linguagem humana baseia-se em regras gramaticais que possibilitam a análise de textos por meio da sua decomposição em elementos menores. Desta forma, um texto pode ser decomposto em sentenças, que por sua vez são compostas por palavras, sendo que cada palavra obedece a leis de formação próprias e são organizadas entre si regidas por regras sintáticas. O significado de cada palavra e das construções sintáticas fornecem as informações que possibilitam o entendimento do texto de forma coerente como um todo. Em outras palavras, as informações morfológicas, sintáticas e semânticas contribuem para a compreensão da linguagem. O processamento destes elementos apresentam diversos desafios de pesquisa decorrentes de problemas originados pelas características inerentes de ambiguidade, não determinismos e das intrincadas dependências contextuais existentes entre os elementos componentes de uma sentença (ROCHA, 2007; BICK, 2000; JURAFSKY; MARTIN, 2008), conforme indicados no item 2.2.1.

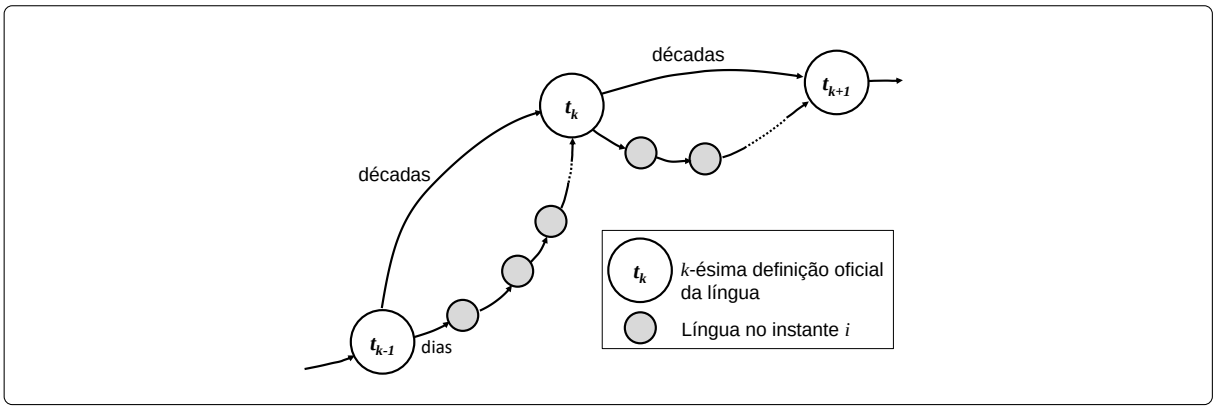
A análise de textos pode ser realizada usando técnicas derivadas de estudos sobre a equivalência de representação da descrição da linguagem por uma gramática, ou seja, por regras capazes de gerar as suas sentenças, ou por uma máquina que consiga determinar se uma sentença pertence ou não ao conjunto de sentenças da linguagem em questão (LEWIS; PAPADIMITRIOU, 1997).

A implementação de sistemas de PLN baseados neste modelo requer o levantamento de todas as regras gramaticais existentes na linguagem analisada para que elas sejam codificadas nos algoritmos. Além das dificuldades já citadas anteriormente, esse requisito adiciona o desafio de manter o modelo aderente ao caráter dinâmico das línguas vivas. A

linguagem utilizada cotidianamente pelas pessoas é constantemente alterada por diversos fatores como modismos, acontecimentos, influências culturais externas, entre outros (NOWAK; KOMAROVA; NIYOGI, 2002; LIEBERMAN et al., 2007).

Em relação à língua erudita, tomando como exemplo a língua Portuguesa no Brasil, podemos observar que, em intervalos de tempo de décadas, há uma definição das regras gramaticais por lei pelo governo. As setas da Figura 1.1 apresentam trajetórias da evolução da língua. O círculo hachurado representa a língua no instante i do tempo. Em intervalos de dias a língua vai evoluindo, até que nos momentos representados pelos círculos em branco ocorre uma definição oficial. Na figura, no momento t_k é feita uma definição das regras de linguagem que objetiva refletir as alterações sofridas desde a última vez em t_{k-1}.

Figura 1.1: Evolução das regras da língua.



Fonte: autor.

A questão do dinamismo é relevante para qualquer sistema de PLN que adota modelos estáticos da linguagem, incluindo aqueles baseados em regras, técnicas estatísticas (MANNING; SCHÜTZE, 1999; PEREIRA, 2000) e conexionistas (LECUN; BENGIO; HINTON, 2015). Nestes tipos de sistema, a língua é modelada a partir de uma representação estática obtida de uma amostragem da linguagem utilizada por uma população num período determinado de tempo. Em sistemas que utilizam técnicas probabilísticas ou conexionistas provenientes da área de inteligência artificial, os mecanismos de PLN são treinados com um corpus finito de amostras da linguagem utilizada em um período específico de tempo. Independente da modelagem uma crítica frequente ao PLN é relacionada ao fato de os sistemas pesquisados considerarem sempre a linguagem limitada.

Em suma, a obtenção de um modelo de PLN baseado em regras que possua mecanismos para se adaptar ao dinamismo de uma língua natural constitui a motivação deste trabalho.

1.2 JUSTIFICATIVA

Historicamente a pesquisa em PLN focou inicialmente no tratamento de texto, com grande ênfase na pesquisa de processamento simbólico considerando o estudo das regras de formação da linguagem. O processamento baseado em regras tem a vantagem de fornecer informações exatas de formação lógica das sentenças durante a análise, porém o custo de processamento exigido para a resoluções de aspectos não determinísticos dificulta o seu uso prático. Nas últimas décadas, houve maior ênfase nas técnicas probabilísticas (MANNING; SCHÜTZE, 1999; PEREIRA, 2000) e conexionistas (LECUN; BENGIO; HINTON, 2015) da área de inteligência artificial. Estas duas formas de abordagem obtiveram resultados bastante satisfatórios em critérios de precisão e tempo de processamento para aplicações industriais, porém não permitem a inferência de regras de formação no decorrer do processamento. Existem também pesquisas que adotam modelos híbridos, combinando essas tecnologias, e há autores constatando que a investigação científica deve continuar sem se limitar somente a uma única forma de modelagem (MANNING, 2015).

De maneira similar à análise de textos em linguagem natural baseado em regras, o estudo das linguagens formais (RAMOS; JOSÉ NETO; VEGA, 2009), que dão base para a construção de compiladores, fundamenta-se na descrição das regras de formação das cadeias que compõem as linguagens. A hierarquia de classes de linguagens proposta por Chomsky (RAMOS; JOSÉ NETO; VEGA, 2009) com as suas classes de gramáticas e reconhecedores equivalentes é utilizada na modelagem dos diversos módulos que compõem um compilador (JOSÉ NETO, 2016).

Usando como suporte teórico a hierarquia de Chomsky, identificam-se pontos em comum entre os tipos de linguagens natural e formal. A sintaxe das linguagens de programação em geral é descrita por meio de gramáticas derivadas do estudo de linguagens formais. Em linguagem natural é possível fazer a análise sintática de um conjunto de sentenças empregando-se gramáticas livre de contexto. Luft (2002), por exemplo, descreve a gramática da língua portuguesa culta num formato que facilita a descrição das construções sintática por meio de gramáticas livre de contexto.

Esta tese adota a modelagem do PLN baseada em regras e explora o uso dos formalismos (JOSÉ NETO, 1993) desenvolvidos para resolver problemas em compilação, dentre os quais a dependência de contexto. A validação de modelos para PLN foi realizada com exemplos da língua portuguesa, que é uma das línguas que apresentam esta característica (JÄGER; ROGERS, 2012; MARNEFFE et al., 2014). Esta tese apresenta-se como a continuidade de pesquisas anteriores realizadas no laboratório do qual o autor faz parte, usufruindo da sinergia com os demais pesquisadores no compartilhamento de técnicas e ferramentas desenvolvidas pelo grupo.

1.3 OBJETIVOS

O objetivo desta tese é obter um método para automatizar a construção de sistemas que possam abstrair informações a partir de uma sentença numa linguagem natural, o português brasileiro no padrão culto, e que possua mecanismos que permitam alterações no seu funcionamento de forma incremental.

As informações a serem abstraídas são a identificação dos atributos dos elementos constituintes de uma sentença em linguagem natural na forma de texto e as regras de relacionamento sintático e de dependência entre eles.

Nesta tese é proposta a geração de reconhecedores a partir de gramática livre de contexto associadas a padrões de dependência previamente levantados como o meio para obter um sistema de processamento de linguagem que seja alterável.

1.4 ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 apresenta os conceitos que dão base para a proposta desta tese, que é descrita no Capítulo 3. O projeto desta proposta e o método de implementação estão descritos nos Capítulos 4 e 5 respectivamente. Os ensaios para validar a proposta e os resultados obtidos são discutidos no Capítulo 6. Finalmente o Capítulo 7 apresenta a conclusão deste trabalho.

CAPÍTULO 2

CONCEITOS

Este capítulo contém definições e conceitos que auxiliam na descrição do sistema de processamento de linguagem natural proposto nesta tese.

2.1 DEFINIÇÕES PRELIMINARES

São descritas nesta seção os termos usados ao longo deste texto, que são interpretações derivadas de trabalhos de outros autores. São definições usadas para descrever o objeto principal de análise do sistema a ser proposto, que são sentenças de uma língua natural no seu formato textual.

Definição 1. Caractere c é a menor unidade de uma descrição textual de uma linguagem natural. No português brasileiro ele é um elemento do conjunto de símbolos constituído pelo alfabeto latino, sinais de pontuação e caracteres adicionais. No Brasil esta definição é estabelecida no Decreto Número 6.583, de 29 de setembro de 2008. Num sistema computacional, cada caractere é mapeado para um agrupamento único de unidades elementares de informação, o "byte", que por sua vez é formado por uma sequência de 8 elementos do "bit" ou dígito binário, que pode assumir os valores 0 e 1. As regras de agrupamentos de bytes são pré-definidos entre os usuários e projetistas de sistemas computacionais para que equipamentos e programas possam usar um mesmo mapeamento. Como exemplos de regras de agrupamento, podemos citar padrões industriais atualmente existentes como os códigos ASCII e UTF-8 (YERGEAU, 2003) que mapeiam estes caracteres para símbolos utilizados para descrever textos em diversas línguas naturais. □

Definição 2. Palavra w é uma sequência finita de caracteres (Definição 1) do ponto de vista computacional. Do ponto de vista linguístico esta representação possui uma ideia associada e constitui uma unidade da linguagem humana (ALMEIDA, 2005) correspondendo à *palavra gramatical* descrito por Bechara (2009). □

Definição 3. Sentença s é uma sequência finita de símbolos, cujo tipo pode ser abstraído em aderência a alguma regra pré-definida, que possua um significado para o projetista e ao usuário do sistema. O tipo de símbolo constituinte da sentença com a menor granularidade é o caractere (Definição 1) que pode assumir qualquer valor previsto na codificação adotada. Outros tipos de símbolos constituintes podem ser definidos a partir da segmentação da sequência de caracteres formando agrupamentos em conformidade com uma regra, resultando em exemplos como palavras (Definição 2), sinais de pontuação, números, o caractere espaço. Se a sentença s pertence a uma língua natural específica L, ela

corresponde a uma oração (BECHARA, 2009). Observe-se que a sentença pode não fazer parte da linguagem natural em questão. □

Definição 4. A classe gramatical de uma palavra (*Part of speech (POS)* em inglês) é a função sintática que ela pode assumir numa sentença (Definição 3). Uma mesma palavra pode assumir múltiplas classes, como a palavra "o", que pode ser classificada como um artigo definido ou um pronome pessoal na forma oblíqua, entre outras. □

Definição 5. Texto t é uma sequência de sentenças (Definição 3) que na estrutura gramatical estratificado de uma língua natural descrito por Bechara (2009) corresponde à camada de "texto". □

Definição 6. A língua é o conjunto de sinais, verbais ou não, e regras de como combiná-los, que determinadas comunidades utilizam para se comunicar e interagir (BECHARA, 2009). A língua natural é usada na comunicação entre humanos e historicamente se desenvolveu de forma natural decorrente de seu uso e repetição sem uma formalização prévia. Ela permite que um pensamento na forma de uma mensagem seja passada de maneira compreensível de uma pessoa a outra (SEVERO, 2014). □

Definição 7. A linguagem, na forma utilizada nesta tese, é um conjunto, finito ou infinito, de sentenças (Definição 3), ou seja, ela possui uma definição próxima àquela utilizada em linguagens formais (RAMOS; JOSÉ NETO; VEGA, 2009). Linguagem natural L, que é objeto de processamento dos modelos aqui propostos, é uma representação da língua portuguesa escrita.

Num entendimento mais amplo, a linguagem possui uma dimensão maior, e é entendido como a capacidade natural que os seres humanos possuem de comunicar os conteúdos de suas consciências como pensamentos, ideias, sentimentos por diversos meios, como palavras, gestos, sons, imagens, cores, expressões faciais, corporais. Esta faculdade mental distingue os humanos de outras espécies animais e possibilita a elaboração de um pensamento, organização do conhecimento e a interação com outros semelhantes (SEVERO, 2014).

Do ponto de vista linguístico, a língua (Definição 6) é uma das manifestações concretas da linguagem. □

Definição 8. Sintagma é uma sequência de palavras (Definição 2) que constitui uma unidade sintática é constituída de forma hierárquica (MIOTO; SILVA; VASCONCELLOS, 2013). A quantidade de itens que podem compô-lo é variável e a sua classe gramatical é definida por um núcleo, que é um de seus elementos. Por exemplo, um sintagma substantivo é uma sequência cujo núcleo é um substantivo e se comporta sintaticamente como ele. □

Definição 9. Árvore sintática de uma sentença (Definição 3) é a sua representação em forma de árvore, com informações sobre a sua constituição de acordo com a definição gramatical de linguagem à qual ela pertence (JOSÉ NETO, 2016; JURAFSKY; MARTIN, 2008). Ela contém na sua estrutura informações sobre as produções gramaticais executadas para gerar a sentença. Todas as folhas da árvore correspondem a símbolos terminais e os nós a símbolos não terminais da gramática. No processamento de linguagem natural as folhas correspondem a classe gramatical das palavras e os nós podem ser sintagmas ou uma estrutura intermediária de acordo com a gramática adotada. □

2.2 PROCESSAMENTO COMPUTACIONAL DE LINGUAGEM NATURAL

A pesquisa de sistemas de PLN tem um vasto histórico. Dentre as obras que fazem um levantamento da área de PLN estão o de Cambria e White (2014) e o livro de Jurafsky e Martin (2008) que cobre extensamente a área, além de linguística computacional e reconhecimento automático de fala.

A seguir é apresentado um dos desafios inerentes à realização de sistemas de PLN, prosseguindo com a apresentação de técnicas que podem ser aplicadas na análise de sentenças.

2.2.1 AMBIGUIDADE E NÃO DETERMINISMO EM PLN

A análise de uma sentença em linguagem natural pode resultar em mais de uma possível interpretação. Esta ambiguidade pode se manifestar de diversas formas (BICK, 2000; JOSÉ NETO, 2015; JURAFSKY; MARTIN, 2008). A lista abaixo apresenta aquelas existentes na língua portuguesa com alguns exemplos extraídos das referências citadas (BICK, 2000).

- Morfológica – Ocorre quando uma palavra admite mais de uma interpretação de classe gramatical num dado contexto. De acordo com as regras de formação da palavra as ambiguidades podem ser classificadas em:
  - Léxica – Podem ocorrer de 2 formas. No primeiro caso podem ser de mesma classe gramatical tendo formas básicas diferentes, como na palavra "foi", que é categorizada como verbo na terceira pessoa singular do pretérito tanto para a forma básica dos verbos "ir" e "ser". No segundo caso, podem ter a mesma forma básica, mas serem de classes gramaticais distintas, como em "complementar" que pode ser verbo ou adjetivo.
  - Flexiva – Ocorre em palavras que são de classes gramaticais diferentes resultantes da flexão de uma mesma forma básica. Por exemplo, a palavra "gostamos" é obtida da flexão do verbo "gostar", que pode ser categorizada no tempo presente ou no pretérito.

  - Léxico-flexiva – Ocorre em palavras com diferenças nas três características: nas formas básicas, nas classes gramaticais e nas flexões. A palavra "busca", por exemplo, pode ser originário das formas básicas "busca" e "buscar", e categorizada como substantivo, feminino, singular, ou verbo na terceira pessoa do singular do presente do modo indicativo.

- Sintática – Ocorre quando existe mais de uma alternativa de interpretação quanto à função exercida por cada termo nas orações. Podem ser classificadas nos tipos abaixo (TANIWAKI, 2001; BICK, 2000):
  - Identificação do constituinte:
    - Na forma sintática – Ocorre quando não é possível identificar qual elemento é modificado por um constituinte da sentença. Por exemplo, na frase "o homem com a bicicleta da China" tanto "homem" ou a "bicicleta" podem ser da "China").
    - Na função sintática – Ocorre quando um sintagma pode ser classificado em mais de uma função sintática. Na frase "um homem que ama toda mulher", "homem" pode ser o sujeito do verbo "amar", ou o objeto direto quando se interpreta que "toda mulher" é o sujeito.
  - Coesão
    - Anáfora – Ocorre quando há múltiplas interpretações para a relação de correferência entre uma palavra e a palavra ou expressão a qual ela se refere. Na frase "amava sua irmã", a "irmã" pode ser do falante ou do ouvinte, ou seja, não há como determinar a quem se refere o pronome "sua".
    - Coordenação – Decorre de múltiplas interpretações para uma relação entre duas palavras ou expressões. Por exemplo, na frase "homens e mulheres no Brasil" é possível interpretar ambos "homens e mulheres" no Brasil, ou apenas as "mulheres" no Brasil.

- Semântico -– A ambiguidade pode ocorrer no nível semântico e podem ter a seguinte classificação (BICK, 2000)
  - Léxica – Ocorre quando uma palavra possui múltiplos significados (polissemia) com as mesmas características morfológicas e sintáticas. Por exemplo, a palavra "fato", que é classificada como substantivo, singular, masculino, possui três significados: a vestimenta também chamada de terno, um rebanho de cabras, ou um acontecimento).
  - Unidades polilexicais – Ocorre quando palavras compostas podem ter significados diferentes. Por exemplo, "ter boas razões para" e "ter razão".
  - Escopo – Ocorre quando uma palavra pode ser interpretada de forma diferente dependendo do escopo no qual está inserido. Por exemplo, na frase "Não

compre três garrafas de vinho, compre licor!", não é possível determinar se a negação se aplica ao número de garrafas ou ao tipo de bebida.

– Papel temático – Ocorre quando não é possível determinar o papel temático de uma palavra. Na frase "o sacrifício da moça", a "moça" pode ser o agente do sacrifício ou o objeto.

Esta ambiguidade intrínseca à linguagem natural traz como consequência a necessidade de se analisar o conjunto de todas as possíveis interpretações de cada componente de um texto e o as combinações possíveis entre eles. A característica combinatória que se apresenta para o processo de aceitação do conjunto de possíveis interpretações acaba se refletindo num alto custo computacional para o tratamento das ambiguidades.

Num levantamento de ocorrência de ambiguidade morfológica na língua escrita portuguesa brasileira realizada por Bick (2000) contabilizou que 98,7% de um universo de 630.000 palavras do corpus Borba-Ramsey tem até 5 significados, sendo que os percentuais de palavras com 1 a 5 significados foram de 46,1%, 23,7%, 11,8%, 13,0% e 3,8% respectivamente. Considerando estes números, em sentenças curtas a quantidade limitada de alternativas torna viável o tratamento exaustivo de cada alternativa com um custo computacional tolerável. Apesar do custo potencialmente elevado que o processamento de sentenças maiores pode acarretar, a vantagem desse procedimento é a exatidão da análise, que permite a obter resultados absolutos e precisos. Entretanto, a técnica de análise exaustiva das alternativas não é viável para o caso geral devido à explosão combinatória resultante de todas as ambiguidades, suscitando o uso de métodos aproximados baseado em heurística ou estatística (MANNING; SCHÜTZE, 1999; LECUN; BENGIO; HINTON, 2015). Embora estas técnicas viabilizem o processo de análise reduzindo o custo computacional, devido às características intrínsecas destes métodos de aproximação, e por serem baseados em modelos construídos a partir de corpus, ou seja, que são diretamente influenciados pela qualidade das amostras, os resultados obtidos estão sujeitos a erros.

2.3 ANÁLISE LIVRE DE CONTEXTO

Do ponto de vista linguístico a gramática de uma língua pode ser entendida como o conjunto de regras que regem o seu uso. Estas regras podem ser deduzidas a partir da observação de uma linguagem comum falada por uma população e podem ser objeto de estudo científico visando a sua formalização. Havendo um consenso formado sobre estas regras, a sua divulgação e ensino aos usuários desta língua facilita o atingimento do objetivo principal do uso da língua que é a comunicação. Nesta tese, a gramática da língua portuguesa brasileira considerada é aquela do nível culto, ou aquela que é formalmente ensinada formalmente nas escolas.

Um aspecto tratado pela gramática é o modo como elementos da língua são combinados para formarem unidades maiores, ou seja, uma forma de descrever ilimitadas frases e

sentenças utilizando um conjunto limitado de regras. Considerando a comunicação escrita, segundo Luft (2002), numa análise partindo do todo até elementos indivisíveis, é possível classificar as unidades constituintes conforme descrito na Tabela 2.1. Em cada linha, é mostrado uma classe do lado esquerdo e os seus membros no lado direito. Cada membro é também uma classe do lado esquerdo na hierarquia imediatamente inferior, chegando-se até a classe mínima, o morfema, que não é divisível.

Tabela 2.1: Classes de agrupamentos e membros constituintes envolvidas na comunicação.

Agrupamento	Membros
Texto	Frase(s)
Frase	Sintagma(s)
Sintagma	Palavra(s)
Palavra	Morfema(s)
Morfema	

Fonte: adaptado de Luft (2002).

Na hierarquia acima, a frase ou sentença e o sintagma são objeto de estudo da sintaxe. Ou seja, a análise sintática tem por objetivo descrever como as palavras são combinadas para construir sintagmas, e as combinações destas estruturas constituintes para formar as frases.

Embora ainda não seja exatamente determinado como a linguagem natural pode ser mapeada dentro da hierarquia de Chomsky (HEINZ; HIGUERA; ZAANEN, 2015) aspectos da análise sintática podem ser modelados usando este suporte teórico.

Em particular, nesta tese consideraremos técnicas relacionadas a linguagens do tipo 3 - livre de contexto e tipo 2 - dependente de contexto.

Em relação ao tipo 3 iniciaremos o assunto tratando de técnicas disponíveis para linguagens formais.

2.3.1 CONSTRUÇÃO DE COMPILADORES

O compilador é um software que converte automaticamente textos escritos em uma linguagem de programação específica em um texto equivalente de instruções que possa ser executado num computador. A conversão é feita por um processo que tem como componente a análise do texto de entrada baseada na teoria de linguagens formais (JOSÉ NETO, 2016; AHO et al., 2007).

O processo de compilação é realizado em etapas, empregando fundamentos teóricos relacionados a linguagens do tipo 3 (regular) e 2 (livre de contexto) da hierarquia de Chomsky (RAMOS; JOSÉ NETO; VEGA, 2009) em módulos de análise léxica e sintática respectivamente. As linguagens de programação podem ser especificadas por meio de uma metalinguagem que descreve a gramática, ou a lei de formação de suas sentenças. Para

automatizar a construção dos analisadores, foram desenvolvidos algoritmos para a geração automática de reconhecedores para estas gramáticas. Por exemplo, para a análise léxica, há algoritmos para a obtenção do autômato finito a partir de uma gramática regular, e para a análise sintática, o autômato de pilha pode ser obtido a partir de uma gramática livre de contexto.

Entretanto, nas linguagens de programação de alto nível há também a necessidade de tratamento de dependência de contexto, que é uma característica não presente nas linguagens do tipo 2, por exemplo na verificação de tipos de dados. Nestes casos é comum a utilização de estruturas de dados e algoritmos auxiliares ao invés de um reconhecedor obtido de alguma definição gramatical (JOSÉ NETO, 2016).

Em contraposição a este cenário, José Neto (1993) apresenta técnicas para lidar com linguagens do tipo 1, 2 e 3 com um formalismo único. Nesse trabalho, os processos envolvidos na compilação são analisados e estruturados de uma forma que possam ser resolvidos com algoritmos que podem ser estendidos de modo incremental para tratarem aspectos progressivamente mais complexos. A construção do compilador de uma dada linguagem de programação é realizada a partir da especificação de sua gramática descrita numa metalinguagem proposta pelo autor e denominada de "notação de Wirth modificada", derivada da notação de Wirth (1977), capaz de expressar linguagens livres de contexto.

O autor também propõe o autômato de pilha estruturado e o algoritmo para obtê-lo automaticamente como reconhecedor da linguagem descrita nesta notação de Wirth modificada, e uma primeira extensão desse método para transformar o autômato resultante num transdutor capaz de gerar as árvores sintáticas relacionadas. A notação foi proposta pois permitia obter diretamente um autômato quase ótimo.

Uma extensão adicional que incorpora ações semânticas a este transdutor é apresentada por Ricchetti e José Neto (2005). Para o tratamento de dependências de contexto o autor apresenta o autômato adaptativo, que estende o autômato de pilha estruturado, dotando-o da capacidade de auto modificação. Na ocorrência de estruturas dependentes de contexto na cadeia de entrada a ser reconhecida, o autômato adaptativo altera a sua estrutura para que assuma uma nova configuração contemplando estados e transições que levem a estados de aceitação por uma trajetória que valide a dependência.

Segue abaixo a definição do autômato de pilha estruturado, conforme notação de José Neto (2016).

Definição 10. O autômato de pilha estruturado pode ser representado por uma máquina abstrata M no formato abaixo:

$$M = (Q, A, \Sigma, \Gamma, P, Z_0, q_0, F) \tag{2.1}$$

com

- A é o conjunto das submáquinas a_i descrita por:

$$a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \tag{2.2}$$

na qual,

- $Q_i \subseteq Q$ é o conjunto dos estados q_{ij} da submáquina a_i.
- $\Sigma_i \subseteq \Sigma$ é o conjunto dos símbolos de entrada da submáquina a_i.
- $E_i \subseteq Q_i$, $(E_i = q_{i0})$ é conjunto com um único estado de entrada da submáquina a_i.
- $S_i \subseteq Q_i$ é o conjunto de estados de saída da submáquina a_i.
- $P_i \subseteq P$ é o conjunto de transições que definem as regras de movimentação da submáquina a_i, cujos elementos podem assumir uma das três formas seguintes.
  - $\gamma q_{ij}\beta \rightarrow \gamma q_{ik}\beta'$, representam transições internas à submáquina a_i, do estado q_{ij} para q_{ik}, efetuadas em vazio, ou com *look-ahead*, ou consumindo algum símbolo da cadeia de entrada, mas sem modificar e nem consultar o conteúdo da pilha.
  - $\gamma q_{ij}\beta \rightarrow \gamma(i,k)q_{m0}\beta'$, representam as transições de chamada da submáquina a_m, a ser iniciada no seu estado de entrada q_{m0}, efetuadas em vazio, ou com *look-ahead*, sem consumir símbolos de entrada. O estado de retorno, q_{ik}, é armazenada na pilha na forma do par ordenado (i,k), após a execução deste tipo de transição.
  - $\gamma(m,n)q_{ij}\alpha \rightarrow \gamma q_{mn}\beta'$, representam transições de retorno para o estado q_{mn} registrado no topo da pilha, a partir de um estado de saída da submáquina a_i. Ao ser executada, esta transição finaliza a operação da submáquina a_i, retirando o par (m,n) da pilha, retornando para a submáquina a_m no estado de retorno q_{mn}.

  As seguintes convenções são utilizadas nas definições das transições acima:
  - $\gamma \in \Gamma^*$ é o conteúdo da pilha.
  - $(m,n) \in \Gamma$ é o par ordenado que representa a informação contida no elemento no topo da pilha antes de uma transição de retorno, que identifica o estado de retorno q_{mn} da submáquina a_m. Este elemento precisa ter sido empilhado previamente por alguma transição de chamada de submáquina.
  - $(i,j) \in \Gamma$ é o par ordenado que representa a informação a ser empilhada por uma transição de chamada de submáquina, executada na submáquina a_i. O par indica o estado de retorno q_{ij} após a finalização da execução da submáquina chamada.
  - $q_{xy} \in Q$, representa um estado específico simbolizado pelo índice y da submáquina a_x, da qual ele faz parte.
  - $\beta \subseteq \Sigma$, representa a cadeia de entrada antes da transição.
  - $\beta' \subseteq \Sigma$, representa a cadeia de entrada depois da transição.

- $Q = q_{ij} \in Q | a_i \in A$ é o conjunto finito de todos os estados do autômato, ou seja, é o conjunto-união de todos os conjuntos Q_i de estados das submáquinas a_i. Q_i constitui uma partição de Q, pois cada estado pertence somente a uma submáquina, ou seja, os conjuntos Q_i são disjuntos dois a dois.

- $\Sigma = \sigma \in \Sigma_i | a_i \in A$, corresponde ao conjunto finito não vazio dos símbolos de entrada do autômato M, que é o conjunto-união de todos os alfabetos de entrada Σ_i das submáquinas a_i.

- $Z_0 \in \Gamma$ é o símbolo inicial de pilha do autômato M, que indica que ela está vazia se ele estiver no topo.

- Γ é um conjunto finito não vazio de símbolos de pilha que constitui o alfabeto de pilha a ser armazenado pelo autômato M em sua memória auxiliar. Corresponde ao conjunto de todos os símbolos que representam estados de retorno das submáquinas a_i, ou $\Gamma = Z_0 \cup \gamma_{ij} = (i, j) | \gamma q_{ik} \alpha \rightarrow \gamma(i, j) q_{mn} \alpha \in P_i, a_i \in A$.

- $P = p \in P_i | a_i \in A$ é o conjunto de regras de transição do autômato M, que é representada pelo conjunto-união de todos os conjuntos P_i de transições que definem as regras de movimentação das submáquinas a_i. P_i constitui uma partição de P, pois cada transição pertence somente a uma submáquina, ou seja, os conjuntos P_i são disjuntos dois a dois.

- $q_0 = q_{00} \in E_0 \subseteq Q_0$, representa o estado inicial do autômato M, que corresponde ao estado inicial $q_{00} \in E_0$ da submáquina específica a_0 denominada submáquina inicial de M.

- $F \subseteq S_0$ corresponde ao conjunto de estados finais de aceitação de M e é um subconjunto de estados de saída da submáquina inicial a_0 de M.

□

2.4 ANÁLISE DE DEPENDÊNCIAS

Os textos em linguagem natural contêm elementos que se referenciam a outras partes dentro do próprio texto. As palavras podem referenciar outras numa mesma sentença, a outras sentenças e palavras distantes do texto, ou até mesmo a elementos não expressos explicitamente. Desta forma a interpretação de um texto requer a identificação de contextos e as relações de dependência existentes entre os elementos. Esta análise das regras de dependência de contexto é um problema de resolução complexa (ROCHA, 2007; BICK, 2000).

O estudo destas regras, ou da Gramática de Dependência (GD), *Dependency Grammar - DG* em inglês, é antigo e há indícios de que remonta da época medieval (GAIFMAN,

1965) e até mesmo centenas de anos antes da era cristã na gramática sânscrita (NIVRE, 2005). Muitos dos conceitos ainda hoje utilizados, como a dependência entre dois elementos, em que um deles é o determinante e o outro, o determinado, foi apresentado por Tesnière (2015) num livro publicado originalmente em francês em 1959. Tesniére é considerado por muitos como o pioneiro da teoria moderna da gramática de dependência (NIVRE, 2005), tanto que, embora tardiamente, a sua obra seminal foi traduzida recentemente em 2015 para a língua inglesa. Em comparação, o estudo da gramática da estrutura da frase (*Phrase Structure Grammar - PSG*) (FRASER, 1990), que é similar a análise sintática apresentada em obras sobre a gramática da língua portuguesa (LUFT, 2002; BECHARA, 2009; OTHERO, 2009), é mais recente e recebeu muitas contribuições de Chomsky.

São apresentadas abaixo definições relacionadas à GD.

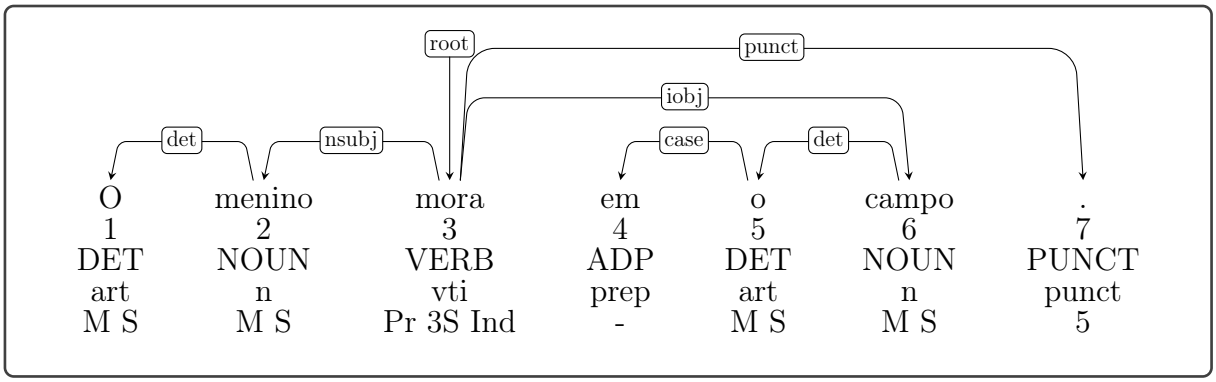
Definição 11. A gramática de dependência é baseada nas relações entre palavras, ou relações de dependência (KÜBLER; MCDONALD; NIVRE, 2009; MEL'ČUK, 1988). □

Definição 12. Relação de dependência entre duas palavras A e B, em que A governa B, ou B depende de A, grafadas com setas em diagramas (por exemplo, $A \rightarrow B$) refere-se a propriedades sintáticas ou semânticas existentes entre eles. Por exemplo, na frase "O menino comeu o bolo", há uma relação sintática de dois elementos que dependem do verbo "comeu", que são o sujeito "o menino" e o objeto direto "o bolo". No sujeito "o menino" desta mesma frase, o artigo (ou determinante) "o" depende em gênero (masculino) e número (singular) do substantivo "menino", que em sua semântica tem estes atributos. □

Definição 13. O diagrama, ou a árvore de dependências é uma forma gráfica de apresentar as relações de dependências existentes entre as palavras de uma sentença (MEL'ČUK, 1988). É uma árvore que contém somente símbolos terminais nos nós, e as conexões entre os seus elementos são rotulados com o tipo de relação de dependência existentes entre eles, e são ordenados indicando qual é o símbolo que é a cabeça da dependência e qual é o dependente. Cada elemento possui somente um elemento do qual ele é dependente, mas múltiplos outros podem depender dele. A Figura 2.1 apresenta um exemplo na notação adotada por demarneffe:2014. A figura está acrescida de informações adicionais colocadas nas linhas logo abaixo de cada palavra, que são a posição da palavra na sentença, iniciando com o elemento mais à esquerda, a classe gramatical da palavra segundo a classificação usada nas anotações das dependências universais, a classe gramatical adotada nas anotações do corpus em português e atributos adicionais. As relações de dependência utilizadas fazem parte do conjunto das relações universais de dependência levantadas para o português (RADEMAKER et al., 2017). As linhas abaixo da sentença representam para cada palavra o seu número na sequência da esquerda para direita iniciada em 1, o *POS tag* usado no corpus UD, o *POS tag* utilizado nas explicações acima, e atributos adicionais. Neste exemplo a palavra aglutinada "no" está decomposta em seus elementos

"em" e "o" que são palavras de classes gramaticais distintas com funções independentes tanto na estrutura sintática quanto nas relações de dependências. □

Figura 2.1: Árvore de dependência da sentença "O menino mora no campo.".



Fonte: autor.

2.4.1 DEPENDÊNCIAS UNIVERSAIS

Vários pesquisadores têm se dedicado a identificar relações de dependência de contexto comuns a diversas línguas naturais, incluindo o português (RADEMAKER et al., 2017), em torno do projeto chamado *Universal Dependencies* (UD) (MARNEFFE et al., 2014; KÜBLER; MCDONALD; NIVRE, 2009) que tem entre os seus objetivos desenvolver uma notação consistente de *treebank* entre as línguas, mas que permita extensões para tratar especificidades de cada uma delas. Considerando que inúmeras notações para as diversas línguas já haviam sido propostas antes desta iniciativa, e que estes corpora são usados por muitos *parsers* de dependências baseados em algoritmos de aprendizado de máquina, um questionamento que se faz é sobre a interferência da notação no desempenho do processo de *parsing*. Ao menos para aqueles baseados em transições, a notação adotada em UD apresentou um desempenho ligeiramente superior às outras (WISNIEWSKI; LACROIX, 2017).

No apêndice A.2 são apresentadas as relações universais de dependência mapeadas em (RADEMAKER et al., 2017).

2.4.2 DEPENDÊNCIAS EM PORTUGUÊS

Na língua portuguesa as relações de dependência entre elementos da frase tem como uma das consequências o estabelecimento de regras quanto aos atributos das palavras. A seguir são listados exemplos destas regras resultantes de dependências de contexto existentes na língua portuguesa (BECHARA, 2009).

Concordância

Sendo o português uma língua flexiva, palavras de determinadas classes gramaticais são constituídas de uma combinação de um significado léxico e morfológico, expressos respectivamente por um radical e as desinências e alternâncias. A flexão ocorre em função do papel que desempenham numa frase, e também de acordo com a relação com as outras palavras. Esta última situação é definida pela relação de dependência de contexto entre as palavras envolvidas, em que a palavra determinada flexiona de acordo com a determinante. A concordância refere-se a esta flexão de certos termos de acordo com atributos gramaticais de outros. Há duas formas de concordância:

- Concordância nominal – Neste caso a palavra determinante é um substantivo ou pronome, e as determinadas são o adjetivo, o pronome (adjetivo), o artigo, o numeral ou o particípio. As determinadas flexionam em gênero e número de acordo com estes atributos presentes na determinante a que se referem. Por exemplo, no sintagma substantivo "o menino esperto", o artigo "o" e o adjetivo "esperto" concordam com o substantivo "menino", que é o determinante.

- Concordância verbal – O verbo da oração é a palavra determinante. As palavras passíveis de flexão que compõe o sujeito (e às vezes o predicativo) são flexionados em número e pessoa de acordo com o verbo. Na oração "Nós estudamos matemática." o verbo "estudamos" está na terceira pessoa do plural, determinando que o agente "Nós" também esteja em concordância com esses mesmos atributos.

Regência

A regência estabelece uma relação de dependência entre um termo regente (determinante) e o regido (determinado).

Se o termo regente for um nome (substantivo, adjetivo, advérbio), ela é classificada como nominal, e o termo regido é um complemento nominal, que completa o significado do nome. Neste caso é estabelecida uma regra que determina a utilização de preposições para se completar o sentido das palavras. Por exemplo, na frase "A escola fica longe de casa.", o regente "longe" determina a necessidade da preposição "de" antes do complemento "casa" (regido).

Na regência verbal o regente é um verbo, que estabelece a relação com o seu elemento regido objeto direto e objeto indireto, e com o adjunto adverbial. Por exemplo, na frase "Os torcedores assistiram à vitória de seu time.", o regente é o verbo "assistir", e o complemento (objeto direto) é o substantivo "vitória". Neste caso, o verbo determina também a necessidade da preposição "a", antes do complemento.

Dependências contextuais distantes

Componentes de uma sentença podem se referir a elementos distantes, criando uma relação de dependência entre elementos em sentenças diferentes. Considerando algum conhecimento subentendido, fora até do texto, é possível fazer referência a conceitos totalmente ausentes.

A referência endófora, quando os elementos dependentes estão presentes no texto em questão, podem ser classificadas em dois tipos de acordo com a posição em que se encontra o elemento referenciado.

Na anáfora, a referência é feita para elementos que ocorreram anteriormente. No exemplo "Fui à festa. Lá, encontrei os meus amigos.", o advérbio "Lá" refere-se ao elemento "festa" da sentença anterior.

Na catáfora, a componente referenciada aparece depois da palavra que faz a referência. Em "O resultado foi esse: vencemos o jogo!", o pronome demonstrativo "esse" refere-se à sentença "vencemos o jogo" cuja localização é posterior a ela.

Como no exemplo anterior, os pronomes e demonstrativos em muitos casos são usados para se referirem a palavras e conceitos enunciados em pontos distantes no texto.

CAPÍTULO 3

PROPOSTA

3.1 INTRODUÇÃO

Neste capítulo é apresentado um modelo para processamento de linguagem natural com um mecanismo que permita alterá-lo de forma incremental e uma proposta de analisador que viabilize esta característica. Esta capacidade de adaptação tem por objetivo aproximar o sistema computacional ao comportamento dinâmico de uma língua natural viva.

3.2 MODELO DE PROCESSAMENTO DE LINGUAGEM NATURAL

Uma forma de modelar a linguagem natural é fazer uso de regras gramaticais (NOWAK; KOMAROVA; NIYOGI, 2002; LIEBERMAN et al., 2007) que possibilitam a análise de textos por meio da sua decomposição em elementos menores. Neste modelo um texto pode ser decomposto em sentenças, que por sua vez são compostas por palavras. Cada uma destas palavras obedece a leis de formação próprias, e são organizadas entre si regidas por regras sintáticas. O significado de cada palavra e das construções sintáticas fornecem as informações que possibilitam o entendimento do texto. Em outras palavras, as informações léxicas, sintáticas e semânticas são os requisitos para a compreensão da linguagem.

De maneira similar, o estudo das linguagens formais (RAMOS; JOSÉ NETO; VEGA, 2009), que dão base para a construção de compiladores, fundamenta-se na descrição das regras de formação das cadeias de símbolos que compõem as linguagens. A hierarquia de classes de linguagens proposta por Chomsky (RAMOS; JOSÉ NETO; VEGA, 2009) com as suas respectivas classes de gramáticas e reconhecedores equivalentes é comumente utilizada na modelagem dos diversos módulos que compõem um compilador (JOSÉ NETO, 2016).

Usando como suporte teórico a hierarquia de Chomsky, identificam-se pontos em comum entre os tipos de linguagens natural e formal. Por exemplo, é possível empregar gramáticas livre de contexto na análise sintática de sentenças. É comum definir a sintaxe das linguagens de programação por meio deste tipo de gramática (SEBESTA, 2012). Por outro lado, ela também pode ser usada para se fazer a descrição sintática de um subconjunto de sentenças da língua portuguesa culta (LUFT, 2002). A seguir é apresentado um modelo de processamento de linguagem natural que explora as características acima.

3.2.1 MODELOS

O ser humano possui uma habilidade inata de observar, analisar, e compreender fenômenos que ocorrem na natureza, o que tem sido fundamental para ele expandir os seus conhecimentos sobre o universo que o cerca.

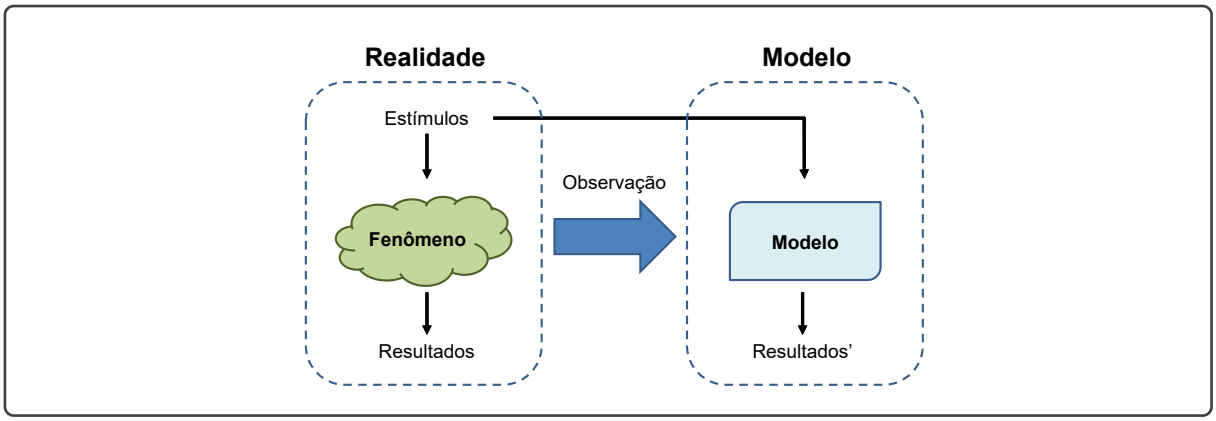
A análise pode ser feita em etapas, primeiramente observando o fenômeno de forma macroscópica prestando a atenção em informações facilmente identificáveis, e posteriormente focando em detalhes que em seu último nível atenda ao objetivo de se compreendê-lo.

Ao se executar este processo está se fazendo um modelo a partir da observação da realidade. Em outras palavras, um modelo é uma representação de um fenômeno real elaborado a partir de sua abstração.

A Figura 3.1 ilustra uma comparação considerando fenômenos onde se pode observar resultados causados por estímulos com modelos que tem por objetivo reproduzir resultados o mais próximo possível do real,

A comunicação falada entre seres humanos pode ser modelada desta forma. Uma pessoa A ao falar com outra B emite sinais sonoros, que representam os estímulos, provocando como resultado a interpretação por parte de B daquilo que A está falando. Um exemplo de aplicação deste tipo de modelagem são os sistemas de reconhecimento automático de fala contínua com vocabulários grandes (*Large Vocabulary Continuous Speech Recognition - LVCSR*), onde um computador capta a voz de uma pessoa por meio de um microfone e infere a sequência de palavras que os algoritmos estimam como a mais provável que a pessoa tenha falado (SAON; CHIEN, 2012).

Figura 3.1: Realidade x Modelo.



Fonte: autor.

O exemplo anterior representa um esforço de engenharia para a construção de sistemas automatizados com o objetivo de reproduzir de forma mais precisa a capacidade real do ser humano de se comunicar. Entretanto, as implementações do modelo ainda não conseguem produzir resultados com a precisão obtida pelo humano nas diversas condições encontradas na realidade (FURUI et al., 2012; HINTON et al., 2012). Em outras palavras, considerando por exemplo que o modelo na Figura3.1 é um sistema *LVCSR*, o resultado

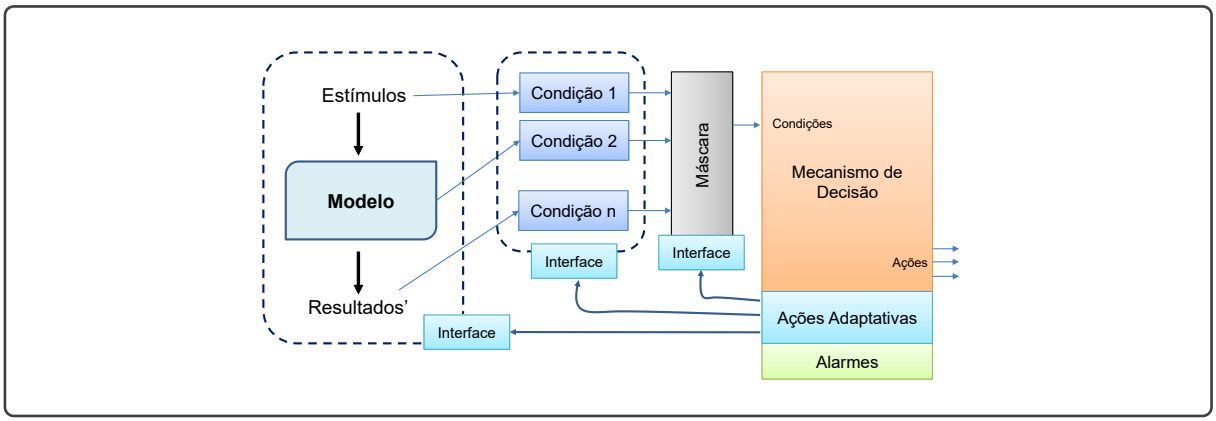
do processamento de reconhecimento, ou seja, a transcrição do áudio pode diferir do entendimento que um ser humano faz de uma sentença falada.

Outros problemas em engenharia podem ser também modelados desta forma. Os sistemas que tem por objetivo reconhecer padrões, sejam de movimentos, imagens, sinais sonoros, textos são exemplos desta classe de problemas. Os pesquisadores buscam elaborar modelos em que se consigam resultados cada vez mais próximos da realidade.

Este processo de melhoria pode ser decomposto em fases repetidas de forma cíclica. Uma primeira fase consiste em analisar os elementos disponíveis, ou seja, os estímulos, o próprio modelo e os resultados. Na próxima etapa, com base nas observações feitas, são tomadas decisões para se alterar o modelo, e eventualmente a forma de se analisar o novo sistema resultante. A fase seguinte é a execução do novo modelo para se realizar uma nova análise.

Para um conjunto de problemas computacionais (JOSÉ NETO, 2007), estas atividades podem ser automatizadas empregando-se técnicas adaptativas. A Figura 3.2 apresenta uma visão geral de um sistema deste tipo, onde o modelo é um dispositivo computacional dirigido por regras, e as ações adaptativas agem em suas regras para modificá-lo durante o funcionamento do sistema, alterando consequentemente a estrutura do próprio modelo. As mudanças podem ocorrer em decorrência de informações de condições obtidas em qualquer ponto do sistema durante a execução. Para isso o sistema é dotado de mecanismos para detectar alterações de estado e filtros são aplicados para que um conjunto de alterações sejam utilizados para ativar ações adaptativas. As ações por sua vez podem promover alterações no modelo, nos mecanismos de detecção, nos filtros, e até mesmo nas regras que regem as ações adaptativas (SILVA FILHO; ROCHA; GUIRALDELLI, 2013).

Figura 3.2: Sistema adaptativo.



Fonte: autor.

3.2.2 ARQUITETURA ADAPTATIVA

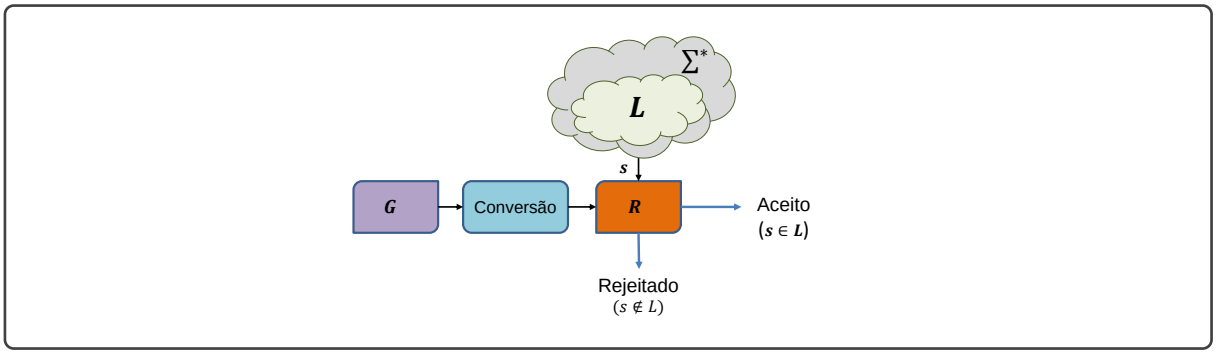
A seguir é descrita uma arquitetura conceitual para se fazer o processamento de linguagem natural.

Tomando como a realidade a ser modelada a interpretação do conjunto de sentenças válidas de uma língua natural observada num período de tempo delimitado em que não ocorram alterações na linguagem, considere-se um sistema hipotético de processamento de uma linguagem natural L. Sejam:

- Σ^* o conjunto de todas as sentenças, incluindo aquelas que não fazem parte da linguagem natural L.
- $s \in \Sigma^*$, uma sentença de Σ^*.
- L, onde $L \subset \Sigma^*$, o conjunto de todas as sentenças válidas de uma linguagem natural.
- G a gramática que descreve L.
- R o reconhecedor de L.

A Figura 3.3 apresenta o sistema hipotético ideal com um reconhecedor R obtido de G, aceitando sentenças s pertencentes L, e rejeitando as demais.

Figura 3.3: Processamento de L.



Fonte: autor.

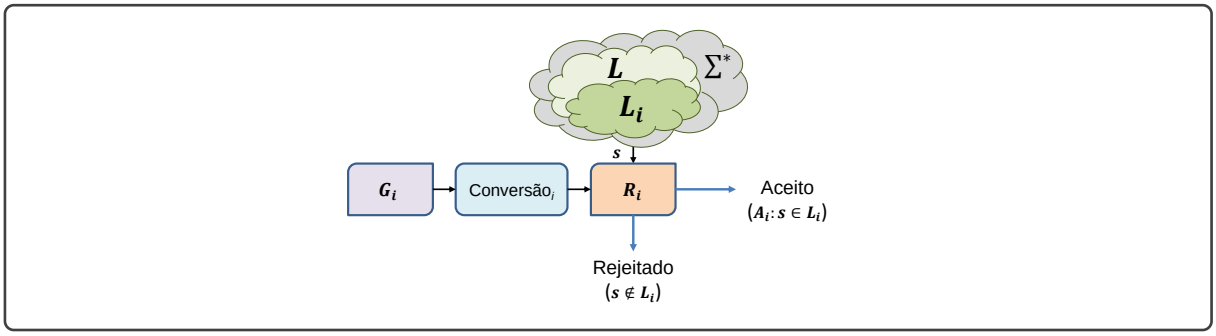
Um modelo aproximado da realidade conforme descrito na Figura 3.4 é um sistema com a capacidade de processar um subconjunto L_i com as seguintes características.

- $s_i \in L_i$, uma sentença de L_i.
- $\exists s_i \in L_i : s_i \in L$, R_i aceita um subconjunto de sentenças de L.
- G_i a gramática que descreve L_i.
- R_i o reconhecedor de L_i.

Sendo uma aproximação, as seguintes divergências podem ocorrer em relação ao modelo hipotético ideal de acordo com a precisão do modelo adotado.

- $\exists s_i \in L : s_i \notin L_i$, há sentenças válidas de L que R_i não aceita.
- $\exists s_i \in L_i : s_i \in (\Sigma^* - L)$, R_i também pode aceitar sentenças que não fazem parte da linguagem real L.

Figura 3.4: Processamento de um subconjunto de L.

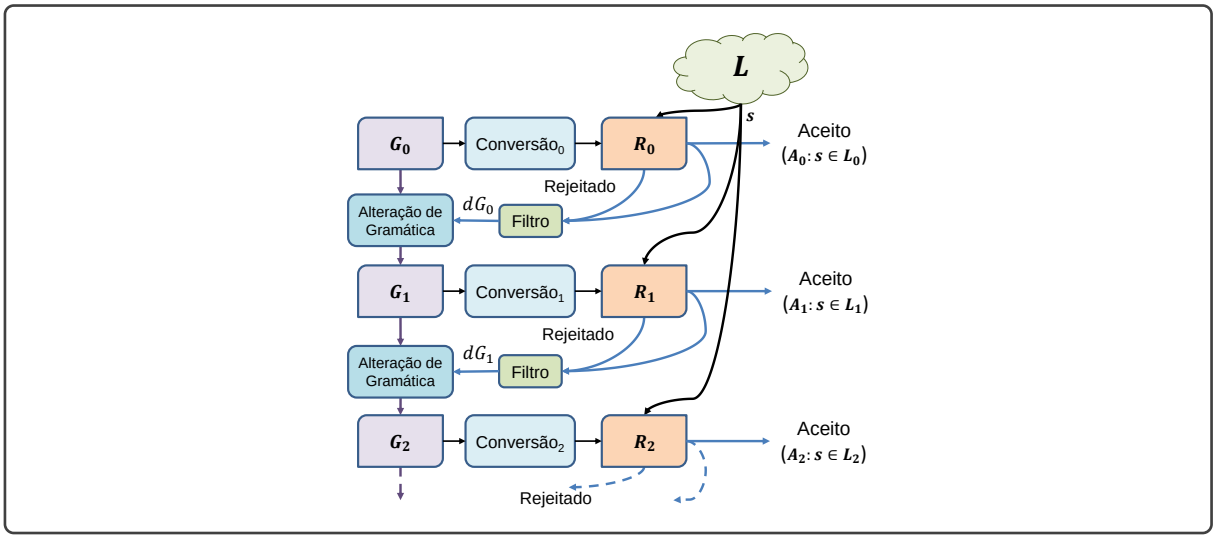


Fonte: autor.

Um modelo de aproximação pode ser obtido utilizando-se técnicas de construção de compiladores. Por exemplo, tomando-se como L_0 o subconjunto da língua portuguesa culta constituído de sentenças livre de contexto (LUFT, 2002), a partir da descrição das regras gramaticais em notação de Wirth (WIRTH, 1977), um reconhecedor R_0 correspondente pode ser obtido com algoritmo de geração de reconhecedores sintáticos apresentado em (JOSÉ NETO, 2016).

Partindo deste conceito inicial podem ser desenvolvidos mecanismos que possibilitem alterar a configuração inicial para que ela possa processar uma linguagem L_1 a partir de uma gramática G_1 obtida a partir de amostras de sentenças incorretamente rejeitadas por R_0 no reconhecimento de sentenças pertencentes a L, e por sentenças não pertencentes a L aceitas de forma incorreta. Em outras palavras $L(R_i) = L(G_{i-1}) \cup L(dG_{i-1})$, onde dG_i representa a gramática que descreve parte do conjunto das sentenças processadas incorretamente por G_i. Este mesmo processo poderia ser repetido para que de forma incremental seja possível obter reconhecedores que aceitem um subconjunto mais aderente à linguagem natural analisada, conforme ilustrado na Figura 3.5.

Figura 3.5: Alterações de gramáticas e reconhecedores.



Fonte: autor.

Desta arquitetura surgem os seguintes desafios:

i) Detectar o processamento incorreto do reconhecedor.

ii) Organizar as informações de erro para que possam ser utilizadas para alterar o modelo original.

iii) Obter uma arquitetura de processamento que permita fazer alterações em seu funcionamento a partir de alguma especificação de modificação.

Uma forma de detectar erros obviamente é inserir intervenção humana no processo para fazer a avaliação. Parte desta atividade pode ser automatizada utilizando por exemplo corpus anotado de sentenças pertencentes à linguagem original para detectar aquelas incorretamente rejeitadas. Por outro lado, o aceite incorreto de sentenças não válidas poderia ser analisado por um corpus formado de sentenças não pertencentes à linguagem geradas aleatoriamente por exemplo.

Gramáticas poderiam ser inferidas a partir dos conjuntos de sentenças processadas com erros (HEINZ; HIGUERA; ZAANEN, 2015) para representarem as alterações a serem aplicadas ao reconhecedor.

3.3 SUMÁRIO

Este capítulo apresentou os conceitos a serem incorporados no projeto e implementação de um mecanismo de processamento de linguagem natural que possa ser gerado de forma automatizada a partir de especificações que se aproximem da língua real de uma maneira incremental.

CAPÍTULO 4

PROJETO

4.1 INTRODUÇÃO

Neste capítulo é descrita a especificação de um projeto de um sistema de processamento de linguagem natural.

De forma similar à construção de compiladores, o sistema proposto faz o uso de múltiplas técnicas para atingir o seu objetivo, incluindo módulos baseados na teoria de linguagens formais, conjugado com algoritmos e estruturas de dados auxiliares.

A seguir é apresentada uma visão geral e os módulos que compõem o sistema.

4.2 VISÃO GERAL

Para a descrição do sistema consideremos que ele gera informações com nível maior de abstração a partir de uma sequência de símbolos, neste caso que pode formar uma sentença pertencente a uma língua natural.

A abstração pode ser interpretada como uma substituição de um conjunto de símbolos por outro. Por exemplo, numa dada língua natural, todas as palavras que morfologicamente são classificadas como substantivo podem ser abstraídas por um símbolo que represente esta classificação. Isso pressupõe a existência de um conhecimento prévio, ou de forma mais simples, uma regra que classifique cada tipo de palavra. As regras em si, podem ter sido criadas pelos usuários da língua, mas a identificação das regras de classificação é um dos focos de pesquisa da linguística. Em suma, o sistema em questão requer a incorporação de regras de abstração para poder atingir o seu objetivo. Consequentemente, do ponto de vista de momentos de execução, o funcionamento do sistema pode ser dividido em duas fases em alguns de seus módulos conforme será visto no decorrer deste capítulo. A primeira é justamente a fase de incorporação de regras de abstração, ou regras gramaticais, que prepara o sistema para a segunda fase que é o de obter uma nova representação para a cadeia de entrada, enriquecida com as informações contidas nestas regras. Estas fases determinam as entradas e saídas do sistema no decorrer de sua execução.

4.2.1 ORGANIZAÇÃO HIERÁRQUICA DO RECONHECIMENTO DE UMA LINGUAGEM

Consideremos um modelo no qual o processo de reconhecimento de uma linguagem L, formal ou natural, seja organizado de forma hierárquica.

Do ponto de vista de implementação, consideremos que a cadeia de entrada para este processo consiste de uma sentença qualquer $s \in L$ formada por uma sequência de símbolos.

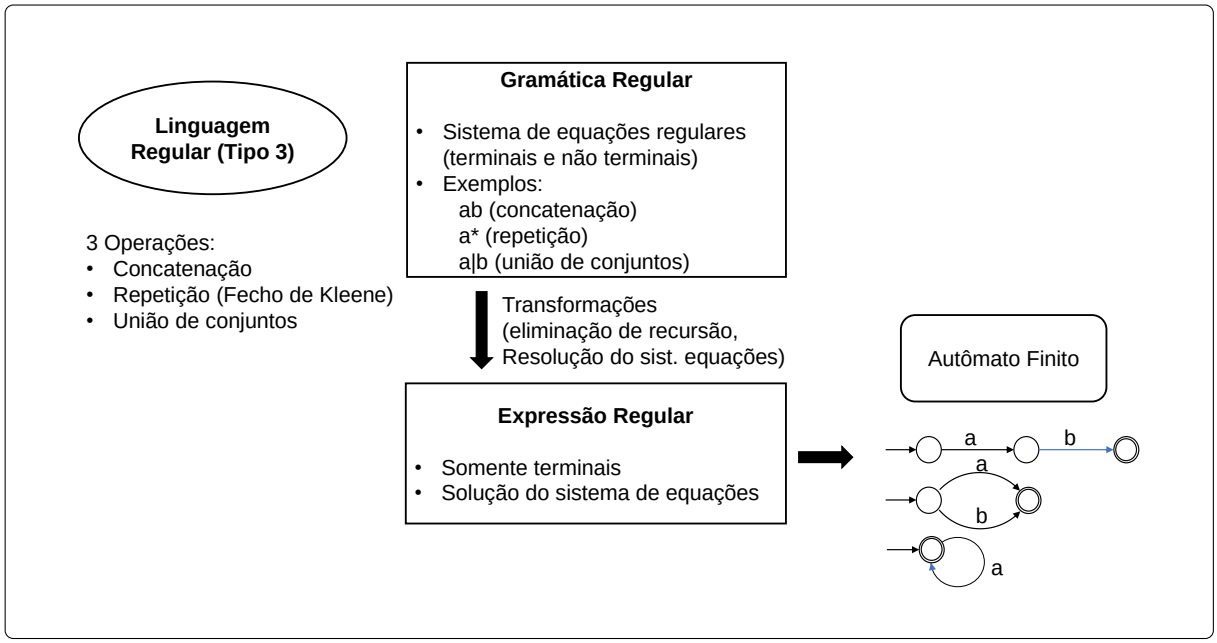
O tipo de símbolo pode ser abstraído em aderência a alguma regra pré-definida que tenha significado para quem está projetando o sistema. Por exemplo, a cadeia de entrada pode ser composta por bits 0 e 1, que podem ser agrupados para formarem uma estrutura de caractere c. Tomando como base a língua portuguesa, estas sequências de caracteres também podem ser reagrupados em subsequências para constituírem um outro tipo de símbolo palavra w considerando que alguns caracteres representam separadores para a formação das palavras como o "espaço" e os sinais de pontuação (ponto, virgula, entre outros).

Em cada nível de abstração i é processado uma linguagem específica L_i reconhecida por uma máquina R_i que aceita cadeias de símbolos pertencentes ao alfabeto Σ_i.

A equivalência entre representações de linguagens pode ser abordada didaticamente numa escala crescente de complexidade, com exemplos de implementação, conforme descrições esquematizadas nas Figuras 4.1, 4.2 e 4.3 respectivamente referentes a linguagens regulares, livres de contexto e dependentes de contexto.

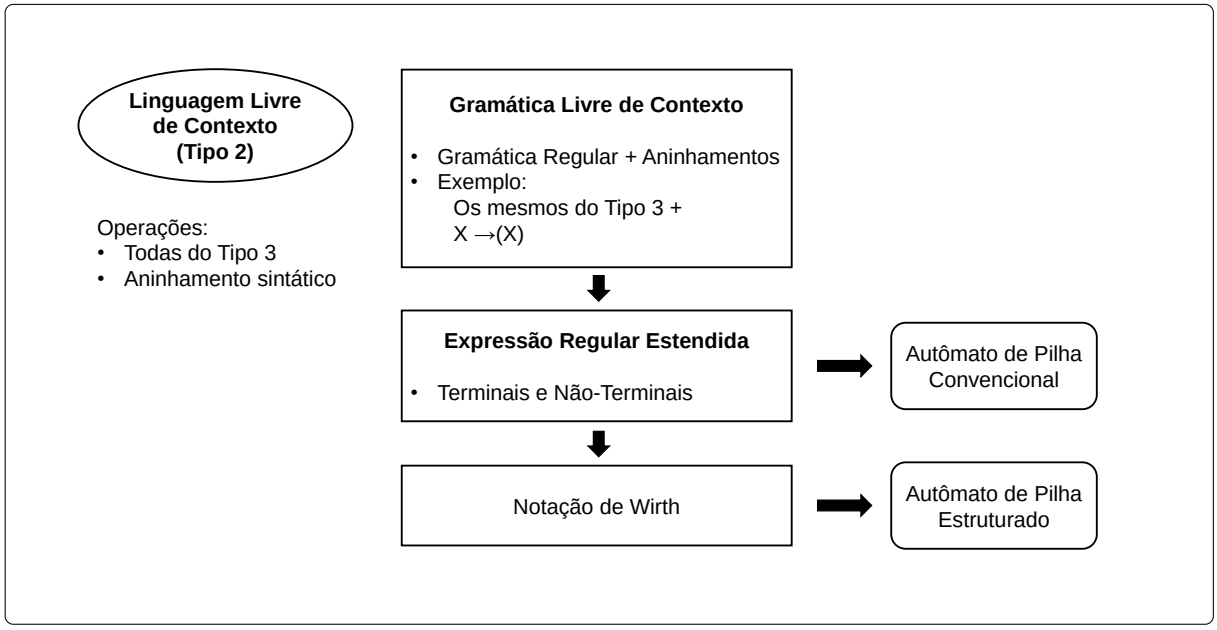
As linguagens dos tipos 2 e 3 são as mais bem pesquisadas (SIPSER, 2006; RAMOS; JOSÉ NETO; VEGA, 2009) e para as quais existem inúmeros algoritmos consolidados para a conversão entre representações. Dentre eles, o autômato de pilha estruturado obtido da conversão de gramática na notação de Wirth (JOSÉ NETO, 2016) apresenta uma característica adicional de facilitar a obtenção da árvore sintática no reconhecimento de sentenças da linguagem.

Figura 4.1: Equivalência de descrições de linguagens do Tipo 3.



Fonte: autor.

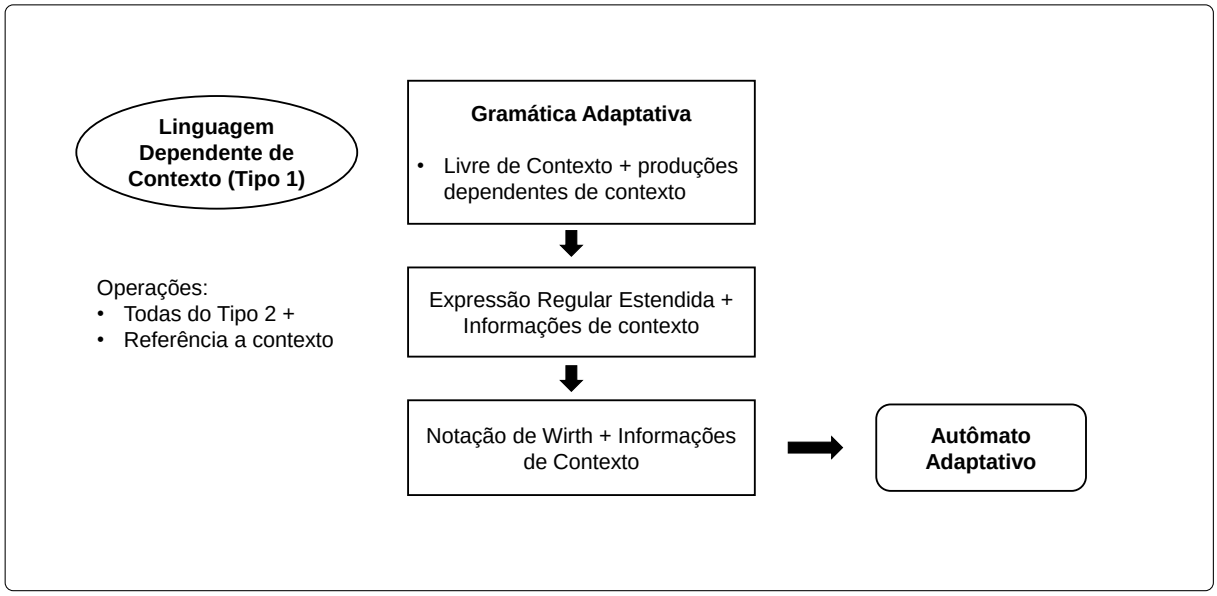
Figura 4.2: Equivalência de descrições de linguagens do Tipo 2.



Fonte: autor.

A Figura 4.3 sobre linguagem do tipo 1 faz referência ao processo de conversão da descrição em gramática adaptativa para o autômato adaptativo. Ele é baseado no algoritmo de conversão de descrição gramatical formal de linguagens livre de contexto na notação de Wirth para autômatos de pilha estruturados apresentado em (JOSÉ NETO, 2016).

Figura 4.3: Equivalência de descrições de linguagens do Tipo 1.



Fonte: autor.

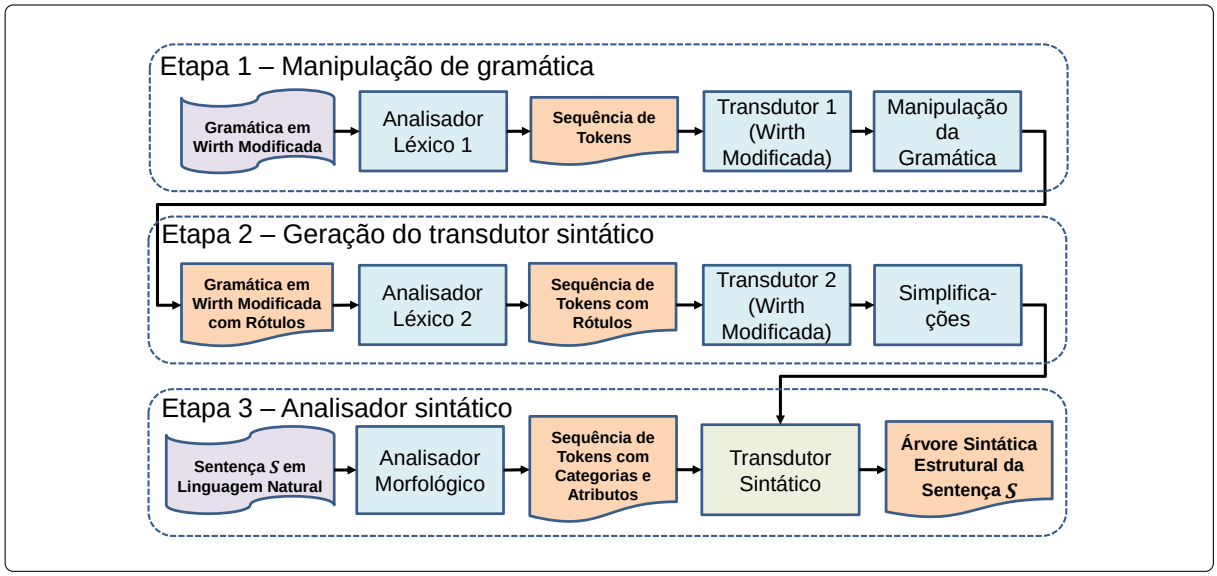
4.2.2 CONSTRUÇÃO DE ANALISADORES DE UMA LINGUAGEM

A organização hierárquica apresentada no item anterior pode ser usada sequencialmente para obter um analisador com automatização da construção de alguns módulos conforme apresentado a seguir.

O método de construção será ilustrado por um sistema que tem como especificação final um dispositivo que receba como entrada uma sentença e que, caso ela pertença à língua natural analisada, obtenha na saída a sua representação na forma árvore sintática livre de contexto com anotações de dependência.

As Figuras 4.4 e 4.5 apresentam a visão geral deste sistema com a indicação das etapas de processamento, os seus componentes principais de entrada, saída, módulos de processamento e as interligações. A Figura 4.4 detalha a obtenção de um transdutor sintático que gera a árvore sintática estrutural de uma sentença em linguagem natural que possa ser gerada por uma gramática livre de contexto. A Figura 4.5

Figura 4.4: Visão geral do analisador - Parte 1.



Fonte: autor.

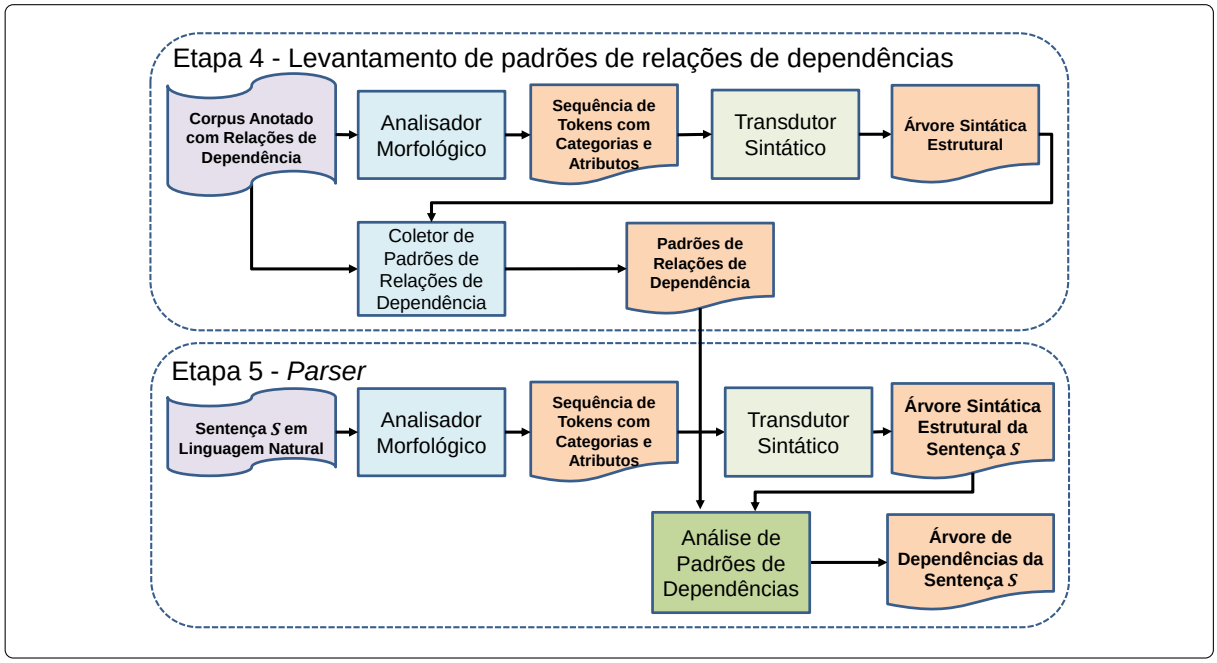
Foi adotada uma arquitetura bastante difundida (AGERRI et al., 2015) em PLN similar a uma linha de produção (*pipeline*) composta por uma sequência de módulos de processamento nos quais os dados de saída de um módulo é a entrada do seguinte. Considerando a sequência de símbolos de entrada de cada módulo como estímulos (ou eventos de entrada) em resposta aos quais ocorre um processamento que resulta em ações (ou eventos de saída), esta arquitetura pode ser interpretada como um encadeamento de motores de eventos (CEREDA, 2018).

Tendo como entrada uma linguagem formal livre de contexto, a metalinguagem usada para descrição de gramática na notação Wirth modificada, as Etapas 1 e 2 utilizam técnicas usadas para a construção de compiladores e transdutores sintáticos apresentadas no trabalho José Neto (1993).

O processamento em cada etapa é gerenciado pelos transdutores, ou seja, o comportamento é similar ao de um compilador dirigida por um analisador sintático (JOSÉ NETO, 2016; AHO et al., 2007).

A Etapa 3 realiza a análise sintática livre de contexto da linguagem natural descrita pela gramática processada nas etapas anteriores, adotando técnicas adicionais conforme será detalhado mais adiante neste capítulo. Este transdutor é a base para os passos seguintes apresentados na Figura 4.5 relacionados à análise de relações de dependências existentes dentro de uma sentença.

Figura 4.5: Visão geral do analisador - Parte 2.



Fonte: autor.

A seguir são descritas as 3 etapas da Figuras 4.4. As etapas 4 e 5 são descritas no item 4.5.

Etapa 1 - Manipulação de gramática

A Etapa 1 tem por objetivo converter a descrição textual de uma gramática livre de contexto na notação Wirth modificada para uma outra representação na forma de uma estrutura de dados que acrescenta informações adicionais de rótulos para a geração de um transdutor sintático (JOSÉ NETO, 1993).

O dado de entrada desta etapa é a gramática livre de contexto que descreve a estrutura de uma sentença de uma linguagem natural. A descrição deve definir as regras das possíveis sequências de sintagmas e palavras que formam uma sentença e as regras de constituição de cada sintagma.

O analisador léxico 1 aceita os símbolos usados na metalinguagem para a descrição de regras da notação de Wirth modificada, entregando uma sequência de *tokens* para o módulo transdutor seguinte.

O transdutor 1 é constituído por um autômato de pilha estruturado que reconhece o texto descritivo da linguagem com ações semânticas a serem executados em suas transições. De acordo com o algoritmo de José Neto (1993) os *tokens* recebidos são acrescidos de rótulos com informações sobre a regra de produção original da qual fazem parte. Ou seja, os símbolos do alfabeto de saída desta máquina de Mealy são constituídos por uma estrutura de dados composto pelo *token* obtido do analisador léxico 1 e de rótulos.

A saída do transdutor 1 deverá passar por processamento adicional para a manipulação de gramática original de forma a eliminar autorrecursões que não caracterizem aninhamento. As operações de agrupamento, substituição, fatoração, desfatoração das expressões devem ser realizadas conforme metodologia apresentada em (JOSÉ NETO, 1993). Note-se que como a gramática em questão descreve uma língua natural, ela pode ser não determinística, diferentemente de uma linguagem de programação.

Etapa 2 - Geração do transdutor sintático

Na Etapa 2, o dado de entrada é uma sequência de símbolos de uma linguagem cujo alfabeto são os possíveis *tokens* originadas da descrição gramatical acrescidas de rótulos, que é o resultado do processamento realizado com sucesso na saída da etapa 1. O analisador léxico 2 fornece estes *tokens* atendendo às solicitações feitas pelo algoritmo do transdutor 2.

O mesmo autômato de pilha estruturado utilizado no transdutor da Etapa 1 é usado também aqui. Entretanto as ações semânticas são diferentes nas transições e com outras nos estados.

O transdutor 2 gera na sua saída uma descrição de um autômato de pilha estruturado correspondente à linguagem livre de contexto descrita pela gramática em Wirth modificada aceita nesta etapa juntamente com informações a serem usadas na Etapa 3 seguinte.

O autômato resultante é não-determinístico em decorrência tanto do algoritmo de conversão como da própria gramática. Não-determinismos poderão ser parcialmente eliminados com aplicações de algoritmos clássicos (COOPER; TORCZON, 2014). Além disso, otimizações parciais poderão ser realizados por meio da minimização de estados (HOPCROFT, 1971). Estas manipulações que objetivam ganhos em tempo de execução no autômato resultante necessariamente devem manter as informações de rótulos que eventualmente estejam presentes nas transições e estados. Por exemplo, transições em que há ações semânticas para a geração de rótulos na cadeia de saída não podem ser eliminadas.

Etapa 3 - Análise sintática

Os módulos que constituem a cadeia de processamento nesta etapa também são inspirados naqueles usados em compiladores, porém a linguagem tratada é natural.

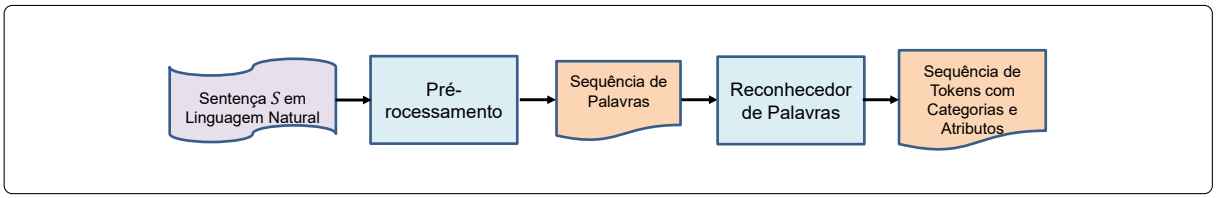
A cadeia de entrada desta etapa é uma sentença. Se ela for reconhecida como pertencente à linguagem natural em questão, a cadeia de saída será a sua descrição equivalente na forma de uma árvore sintática na notação proposta por José Neto (1993).

Os itens 4.3 a 4.4 descrevem os módulos constituintes desta etapa de processamento.

4.3 ANÁLISE LÉXICO MORFOLÓGICA

A análise morfológica tem por objetivo classificar cada palavra da sentença. O processamento inclui o pré-processamento do dado bruto, ou seja, da cadeia de caracteres que constitui a sentença de entrada.

Figura 4.6: Visão geral da análise léxico morfológica.



Fonte: autor.

4.3.1 PRÉ-PROCESSAMENTO DA CADEIA DE ENTRADA

O primeiro objetivo é identificar as palavras, ou *tokenize* na língua inglesa, que é uma tarefa cuja complexidade depende da língua na qual foi escrita a sentença. Por exemplo, na língua portuguesa, ou inglês, o processamento é facilitado pelo fato do caractere de espaço e de pontuações serem delimitadores de palavras. Em línguas sem delimitadores específicos como a japonesa, esta tarefa requer uma maior análise da sentença.

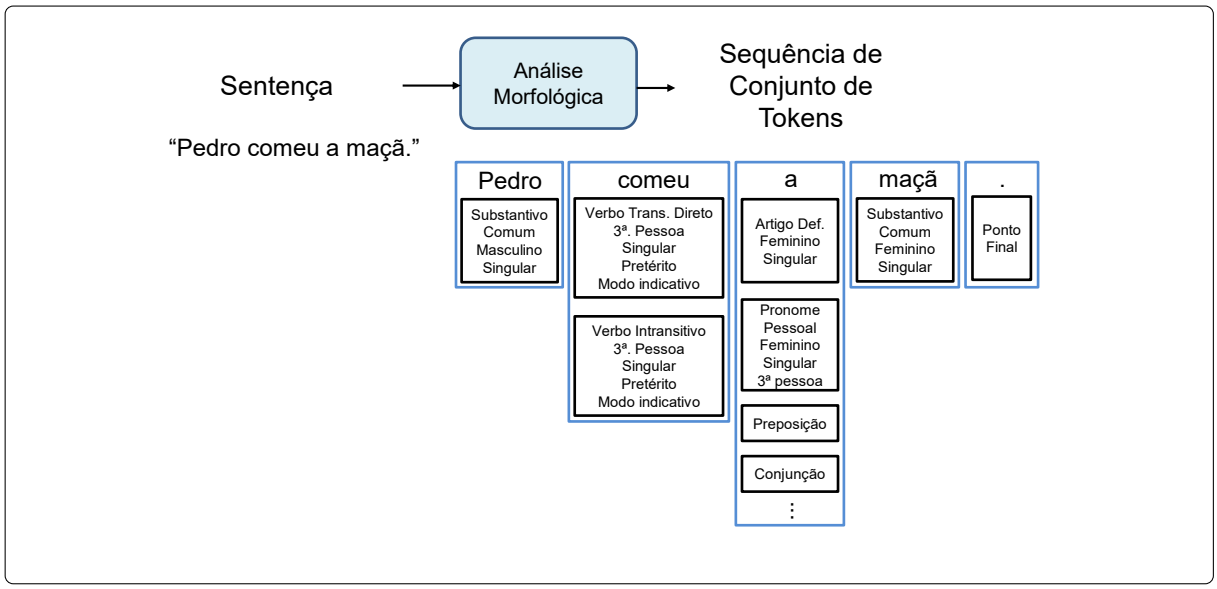
Muitos sistemas de codificação preveem uma quantidade maior de possibilidades de caracteres distintos do que é necessário para representar os caracteres usados numa língua natural. Uma das primeiras tarefas é validar os caracteres utilizados na sentença, adotando critérios prévios para o tratamento de cada um deles. Algumas alternativas possíveis são, ignorar o caractere suprimindo-o da cadeia de entrada ou substituir por uma outra sequência obedecendo alguma tabela de conversão (por exemplo, substituir o caractere "#" por "*hashtag*", ou "cerquilha").

Nesta fase pode-se identificar as possíveis combinações de palavras compostas e separar as palavras resultantes da aglutinação de outras que possuem classes gramaticais distintas. Um exemplo é a palavra "da", que pode ser decomposta na preposição "de" e no artigo definido "a".

4.3.2 CLASSIFICAÇÃO DA CLASSE GRAMATICAL

Após o processamento preliminar, inicia-se o processo de classificação das palavras, cujo objetivo é obter o conjunto de todas as possíveis classe gramatical (Definição 4) que cada uma pode assumir, juntamente com outros atributos associados. Na Figura 4.7 é apresentado um exemplo deste processamento, chamado de *POS tagging* em inglês, que considera a ambiguidade sintática existente em algumas palavras. Neste caso, podemos tomar a palavra "a", que pode ser um artigo definido no gênero feminino, no singular, ou uma preposição, entre outras.

Figura 4.7: Análise morfológica de uma sentença.



Fonte: autor.

A determinação da classe gramatical de cada palavra constitui em si um campo de estudo com vários algoritmos publicados (JURAFSKY; MARTIN, 2008).

Busca em dicionário

Uma forma intuitiva de implementação é a utilização de um dicionário, ou seja, as classes gramaticais de uma palavra seriam obtidas a partir da consulta a uma base de dados que conteria informações de cada palavra e os atributos associados. A organização dos dados deste dicionário e o método de busca seriam os fatores determinantes do desempenho de sua implementação. Um desafio é o espaço necessário para armazenar os dados de todas as possíveis palavras de uma língua.

Conjugador de verbos e flexionador de palavras

Uma forma de diminuir o tamanho deste dicionário é realizar a análise das palavras empregando as leis de formação existentes para algumas delas, como verbos regulares e a

flexão de alguns substantivos e adjetivos na língua portuguesa. Tomando o verbo regular "escrever" por exemplo, é possível obter todas as palavras resultantes da aplicação das regras da segunda conjugação (BECHARA, 2009), como "escrevi" que corresponde ao pretérito do indicativo na primeira pessoa do singular.

Padovani, Contier e José Neto (2018) aplica o autômato adaptativo "coletor de nomes" proposto por José Neto (1993) para construir um autômato finito que aceita palavras obtidas de um corpus constituído de sentenças cujas palavras estão anotadas com a respectiva classe gramatical, e também flexiona verbos, substantivos e adjetivos com comportamentos regulares. O autômato opera em dois modos distintos: no modo de aprendizado ele se automodifica incorporando as palavras coletadas do corpus e flexionando palavras; no modo de decodificação, recebe uma palavra como entrada, e se ela for aceita, retorna as classes gramaticais.

Outras formas de categorização de palavras

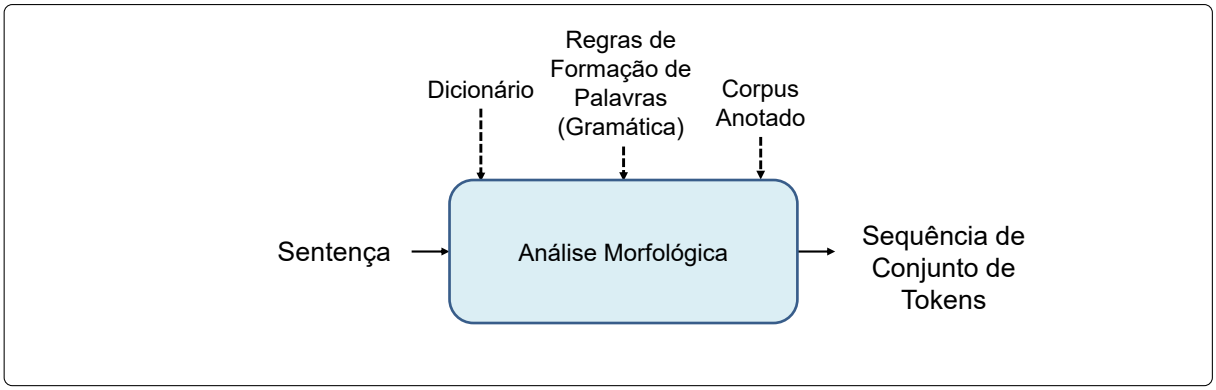
Dois métodos de categorização de palavras baseados em algoritmos de aprendizado de máquina frequentemente utilizados (JURAFSKY; MARTIN, 2008) são o *Hidden Markov Model* (HMM) e *Maximum Entropy Markov Model* (MEMM).

No HMM, a fase de aprendizado usa um corpus anotado que consiste de sequências de palavras, cada qual com a anotação da sua classe gramatical. Para cada palavra são estimadas as probabilidades de transição para as palavras subsequentes, considerando as suas classes gramaticais. Na fase de decodificação é utilizado o algoritmo de programação dinâmica Viterbi que calcula a sequência de classes que apresente a maior probabilidade de ocorrência baseado nas informações obtidas na fase de aprendizado.

Outro método de categorização é o *Maximum Entropy Markov Model* (MEMM).

A Figura 4.8 resume as entradas e saídas do subsistema de análise morfológica, incluindo os dados utilizados nas duas fases de operação: no aprendizado a na análise propriamente dita.

Figura 4.8: Entradas e saídas do subsistema de análise morfológica.

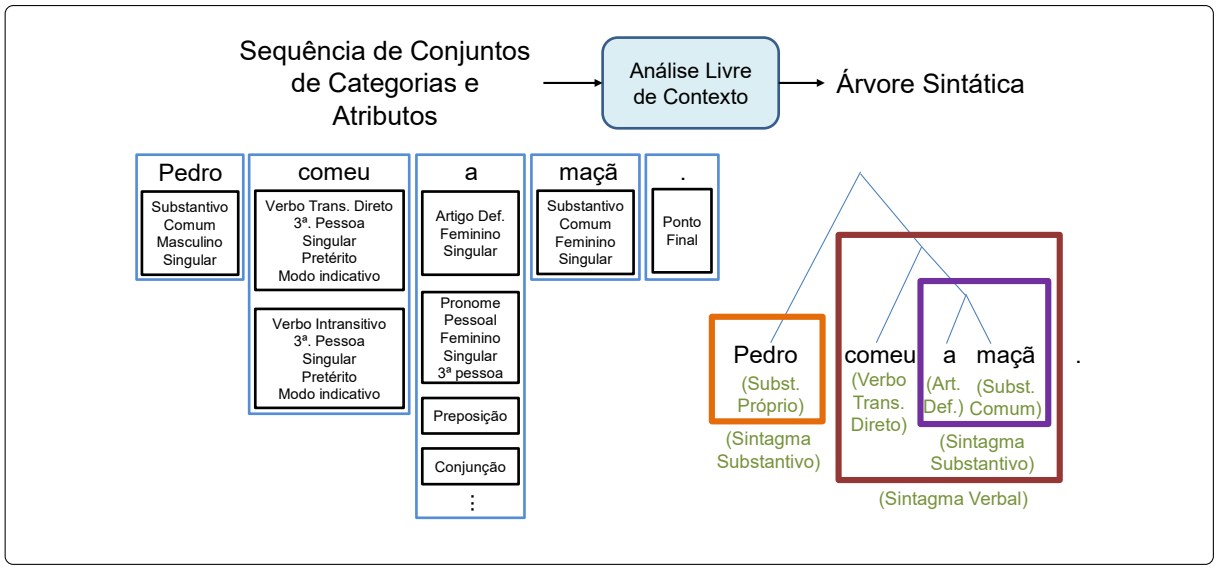


Fonte: autor.

4.4 ANÁLISE LIVRE DE CONTEXTO

Novamente usando a abstração da caixa-preta, conforme ilustrado na Figura 4.9, este analisador tem por objetivo obter na sua saída as possíveis árvores sintáticas (Definição 9) para a sua entrada que consiste na sequência de elementos gerados pelo analisador morfológico descrito no item 4.3 anterior.

Figura 4.9: Análise da sentença livre de contexto.



Fonte: autor.

De modo similar à análise morfológica, a análise livre de contexto opera em duas fases distintas.

Considerando que a análise será realizada usando regras de uma gramática do tipo livre de contexto na hierarquia de Chomsky, na primeira fase um reconhecedor equivalente é construído de forma automática. Ou seja, o usuário deste sistema deverá fornecer a gramática que gera a língua a ser analisada, que passará por um algoritmo que a converterá para um autômato de pilha. Este algoritmo é baseado na tecnologia para linguagens formais usada na construção de compiladores.

4.4.1 GERAÇÃO DE UM TRANSDUTOR

Neste trabalho é utilizada uma variante do algoritmo de obtenção de gramáticas na notação de Wirth modificada para autômatos de pilha estruturados (JOSÉ NETO; MAGALHÃES, 1981; JOSÉ NETO, 1993) desenvolvido por José Neto (2016).

A partir da gramática é obtido um transdutor que converte as sentenças de uma língua numa lista, que representa a sua árvore de derivação.

Entretanto, para o processamento de linguagem natural, existem duas diferenças em relação às técnicas de geração de compiladores.

A primeira é na gramática. Por descreverem a estrutura sintática da frase da linguagem natural, os símbolos não terminais da gramática representam sintagmas (Definição 8) e os símbolos terminais correspondem à classe gramatical das palavras.

A segunda diferença se refere ao *parsing*, conforme descrito no item a seguir.

4.4.2 *PARSING*

Uma diferença entre reconhecedores para linguagem natural e aqueles usados para a construção de compiladores é a admissão do não determinismo. Isso reflete a possibilidade de uma mesma palavra admitir mais de uma classificação gramatical, e a existência de regras de produção na gramática que permitem não terminais distintos e alternativos numa mesma sequência, e que podem ser iniciados por um mesmo terminal. Ou seja, para uma mesma cadeia de entrada o autômato resultante permitirá múltiplas trajetórias de aceitação ou rejeição, o que consequentemente implicará na necessidade de explorar todas as alternativas possíveis para o término do processamento.

Conforme descrito no item 2.2.1, as ambiguidades sintática e morfológica, sem contar a semântica, que não faz parte do escopo desta análise, causam uma explosão exponencial das combinações possíveis que requerem ser analisadas, impossibilitando a obtenção das possíveis árvores sintáticas de uma sentença sem usar alguma técnica para diminuir as alternativas.

Uma possível estratégia para lidar com esta questão é a adoção de algoritmo baseado em modelo probabilístico de sequências de palavras *N-gram* (JURAFSKY; MARTIN, 2008) para diminuir a quantidade de combinações a serem consideradas. Neste caso, um modelo de *N-gram* é construído a partir de amostras da língua natural de um corpus anotado, constituído de sequências válidas de classes de palavras. Após a análise morfológica, as sequências de classes de palavras com probabilidade nula, ou abaixo de um limiar definido pelo usuário, seriam eliminadas evitando-se o processamento desnecessário.

4.5 ANÁLISE DE DEPENDÊNCIAS DE CONTEXTO

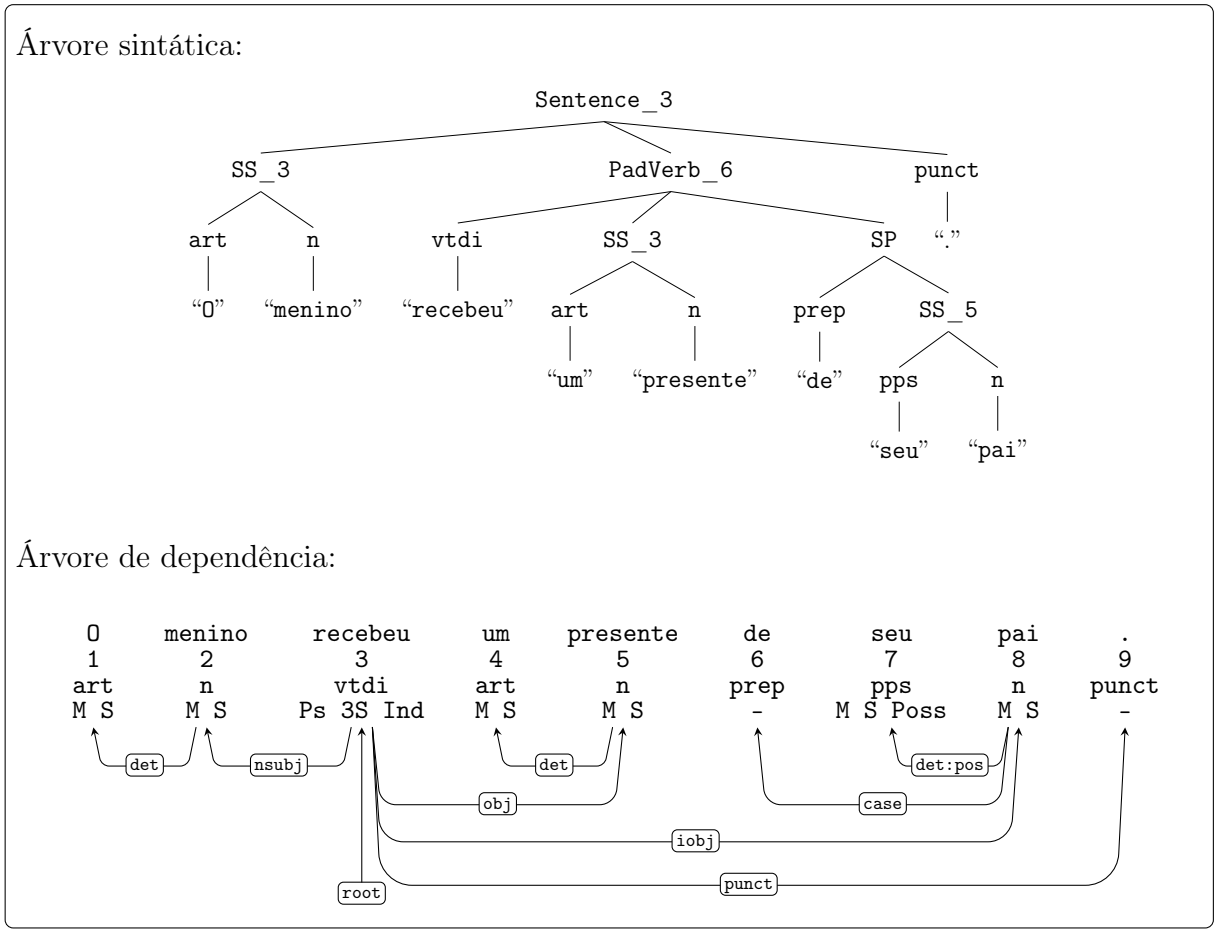
A análise das relações de dependência ocorre durante a obtenção da árvore sintática estrutural (item 4.4.2 anterior). Na versão implementada são consideradas somente sentenças na ordem direta cuja árvore de dependência é do tipo projetivo (HERINGER, 1993). O processo é realizado por meio de aproximações sucessivas conforme a identificação das regras de produção da gramática livre de contexto no decorrer da execução do autômato de pilha estruturado usado no transdutor sintático.

4.5.1 PADRÕES DE RELAÇÕES DE DEPENDÊNCIA DE CONTEXTO

Para ilustrar o processo consideremos a análise da sentença “O menino recebeu um presente de seu pai.” cuja árvore sintática e de dependência estão apresentadas na Figura

4.10. A gramática utilizada está descrita nas Figuras 6.3 e 6.4 do Capítulo 6.

Figura 4.10: Árvore sintática e de dependência.

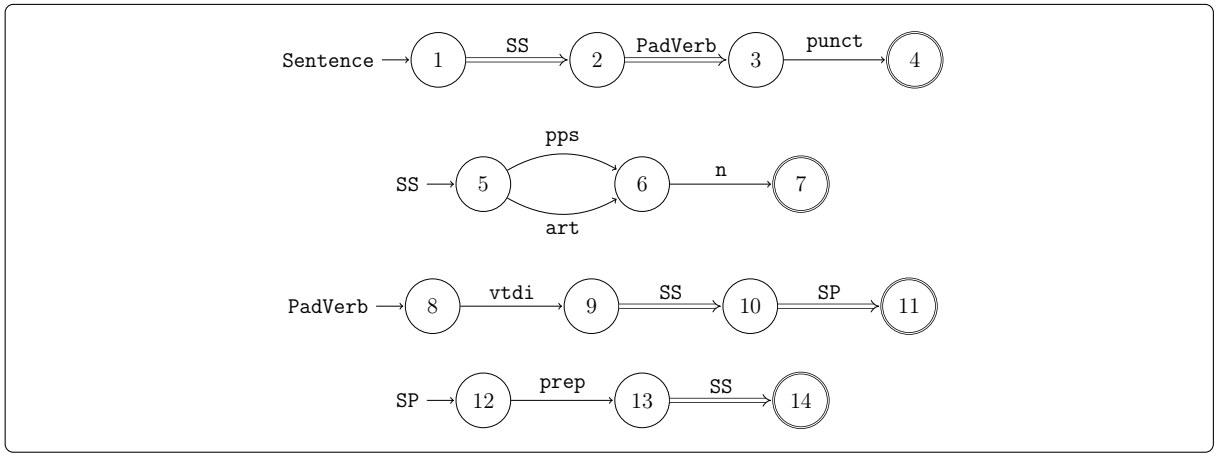


Fonte: autor.

O *parsing* da sentença é realizada da esquerda para a direita em relação à cadeia de entrada pelo autômato obtido a partir da gramática. A Figura 4.11 apresenta partes deste autômato relevantes para a análise do exemplo.

Considerando a forma como é construído o autômato, para se poder consumir o primeiro símbolo da cadeia, a palavra “O” neste exemplo, é percorrida uma trajetória que corresponde às produções gramaticais do maior nível hierárquico para as inferiores. `Sentence` é a primeira regra de produção, e em seguida é percorrido o trajeto numa submáquina que correspondente à regra de produção `SS`, que do ponto de vista da linguagem natural representa as regras de formação de um sintagma substantivo. Nesta submáquina, a subcadeia “O menino” provoca uma sequência de transições que culmina com o reconhecimento com sucesso do sintagma substantivo. Após a finalização da execução desta regra de produção é realizada a análise de relações de dependências considerando os padrões previamente levantados que podem ocorrer dentro deste sintagma. O início e o final do percurso correspondente a uma regra específica de produção é determinado pelo mesmo algoritmo de geração de rótulos a ser detalhado no item 5.3.2. O padrão de dependência

Figura 4.11: Descrição parcial do autômato de pilha reconhecedor de sentenças.



Fonte: autor.

que corresponde a este caso é o primeiro dos três tipos listados para `SS` na Figura 4.12. Esta figura apresenta individualmente os padrões para cada sintagma existente na sentença exemplo, indicando as relações existentes entre os elementos constituintes que estão no mesmo nível hierárquico, ou seja, que se conectam ao mesmo nó da árvore sintática. A subárvore estrutural correspondente ao nó `PadVerb` da Figura 4.13 ilustra esta situação na qual as folhas correspondentes a terminas e não terminais se conectam diretamente a esse nó.

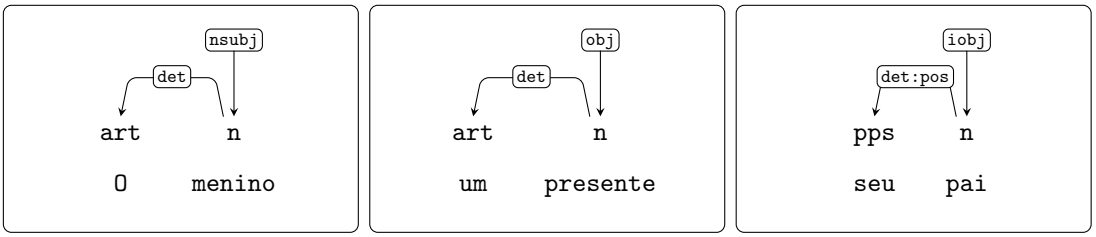
A Figura 4.14 apresenta outros padrões de exemplo para `PadVerb` adotando um formato gráfico que facilita a visualização de todas as informações que devem constar na especificação de um padrão de relação de dependência. Nesta figura, embora todos os padrões tenham a mesma relação com os elementos externos, cada um difere em relação aos tipos de verbo, os complementos, e a sequência dos constituintes, que é definida pela gramática livre de contexto.

A Figura 4.15 apresenta os padrões dos sintagmas `SP` (item (a)) e `SS` (itens (b) e (c)).

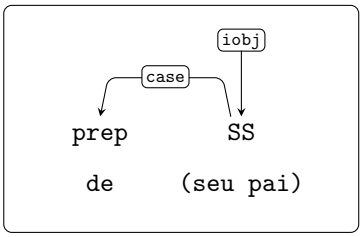
Nestes exemplos observa-se que, assim como em cada elemento constituinte, os padrões correspondentes a sintagmas tem uma relação na qual é dependente de somente um ente externo, representado por seta vertical de entrada nas figuras, e podem ser o determinante de nenhum ou de múltiplos padrões externos. A seta que indica a dependência de algum elemento externo é sempre direcionada a um único constituinte interno, que corresponde ao núcleo do sintagma de forma aderente à teoria X-barra (MIOTO; SILVA; VASCONCELLOS, 2013). No sintagma preposicional `SP` o núcleo é um sintagma substantivo. Nos sintagmas substantivos `SS` deste exemplo, o núcleo é um substantivo. Este comportamento é observável também nos padrões `Sentence` e `PadVerb` nos quais os núcleos são os verbos.

Figura 4.12: Dependências parciais.

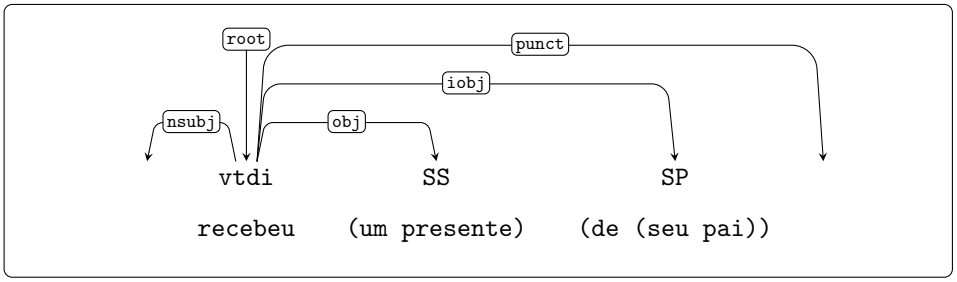
Sintagma Substantivo (`SS`):



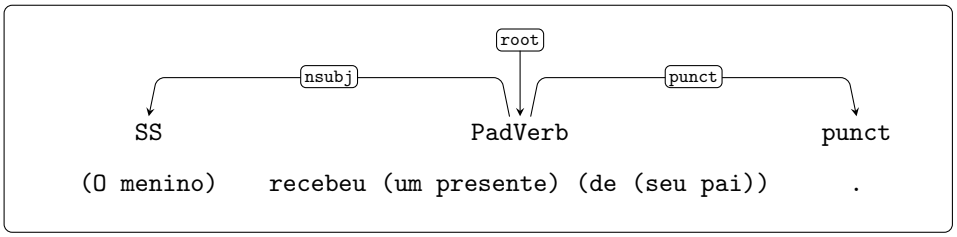
Sintagma Preposicional (`SP`):



Padrão Verbal (`PadVerb`):

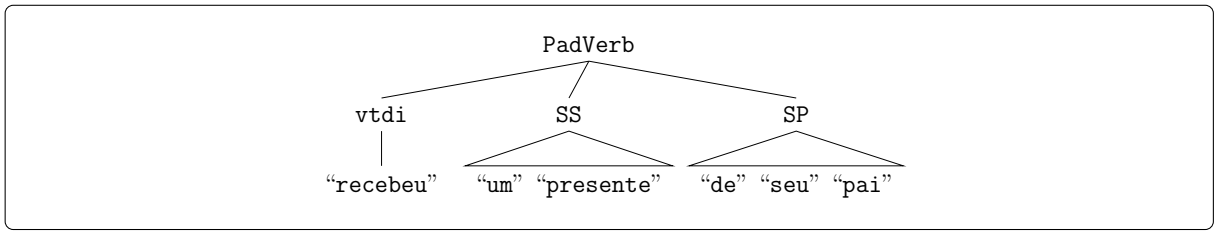


Sentença (`Sentence`):



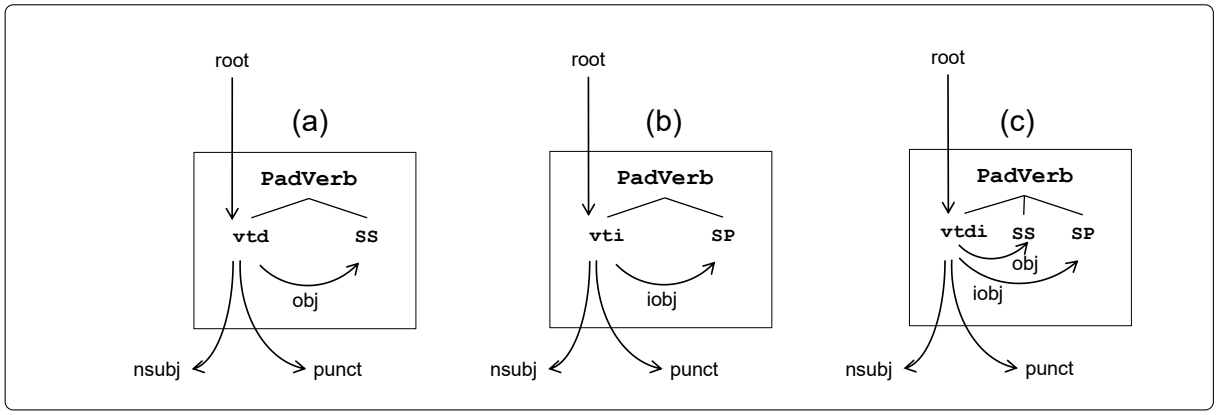
Fonte: autor.

Figura 4.13: Árvore sintática do não terminal `PadVerb`.



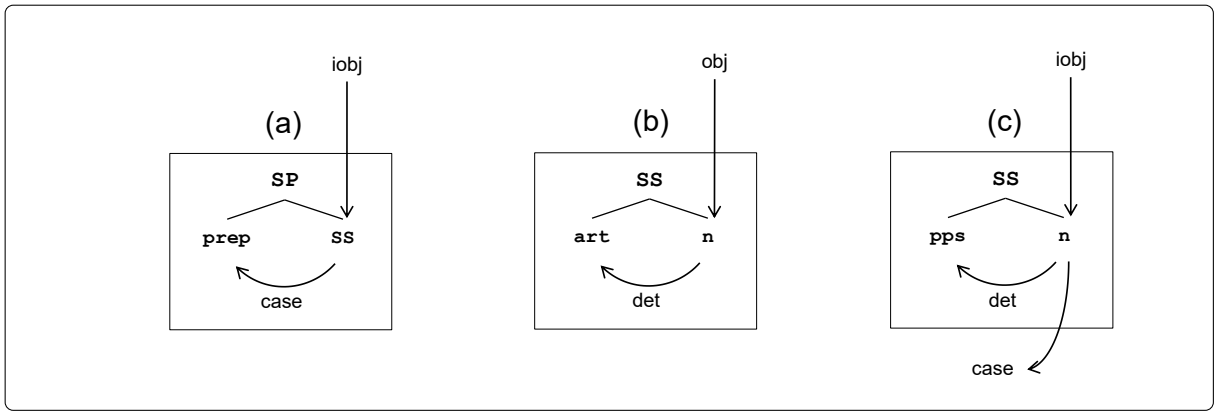
Fonte: autor.

Figura 4.14: Padrões de dependência - `PadVerb`.



Fonte: autor.

Figura 4.15: Padrões de dependência - `SP` e `SS`.



Fonte: autor.

4.5.2 LEVANTAMENTO DE PADRÕES

A identificação de padrões de relações de dependências pode ser parcialmente automatizada por meio da utilização do transdutor livre de contexto (4.4.1) e de corpus anotado. Para cada sentença do corpus deverá ser adotado o procedimento descrito a seguir.

A sentença deverá ser submetido ao transdutor que gerará informações sobre a estrutura sintática com a identificação da regra de produção da gramática que foi aplicada em cada nó da árvore e os dados associados a cada terminal como no exemplo da Figura 4.10.

Cada relação de dependência ocorre sempre entre duas palavras que são representadas por duas folhas na árvore sintática. A informação sobre esta relação de dependência deverá ser registrada na estrutura de dados do padrão de dependência associado a cada não terminal (representado pelos nós da árvore) que esteja no caminho que interliga as duas palavras passando pelo menor ancestral comum (*lowest common ancestor*) (CORMEN; LEISERSON; RIVEST, 1990). A Figura 4.16 apresenta um exemplo para a relação de dependência (identificador `acl:relcl`) entre o substantivo "casa" e a oração subordinada adjetiva "que eu vi" cujo núcleo é o verbo "vi". Esta informação será preenchida nas estruturas de dados dos padrões de dependência associados aos não terminais $\texttt{SS}_{10}$ e $\texttt{OSAR}_{6}$ que estão no percurso da árvore entre as palavras "vi" e "casa" passando pelo nó $\texttt{SS}_{10}$ que é o menor ancestral comum.

4.5.3 TRATAMENTO DE ORDEM INVERSA

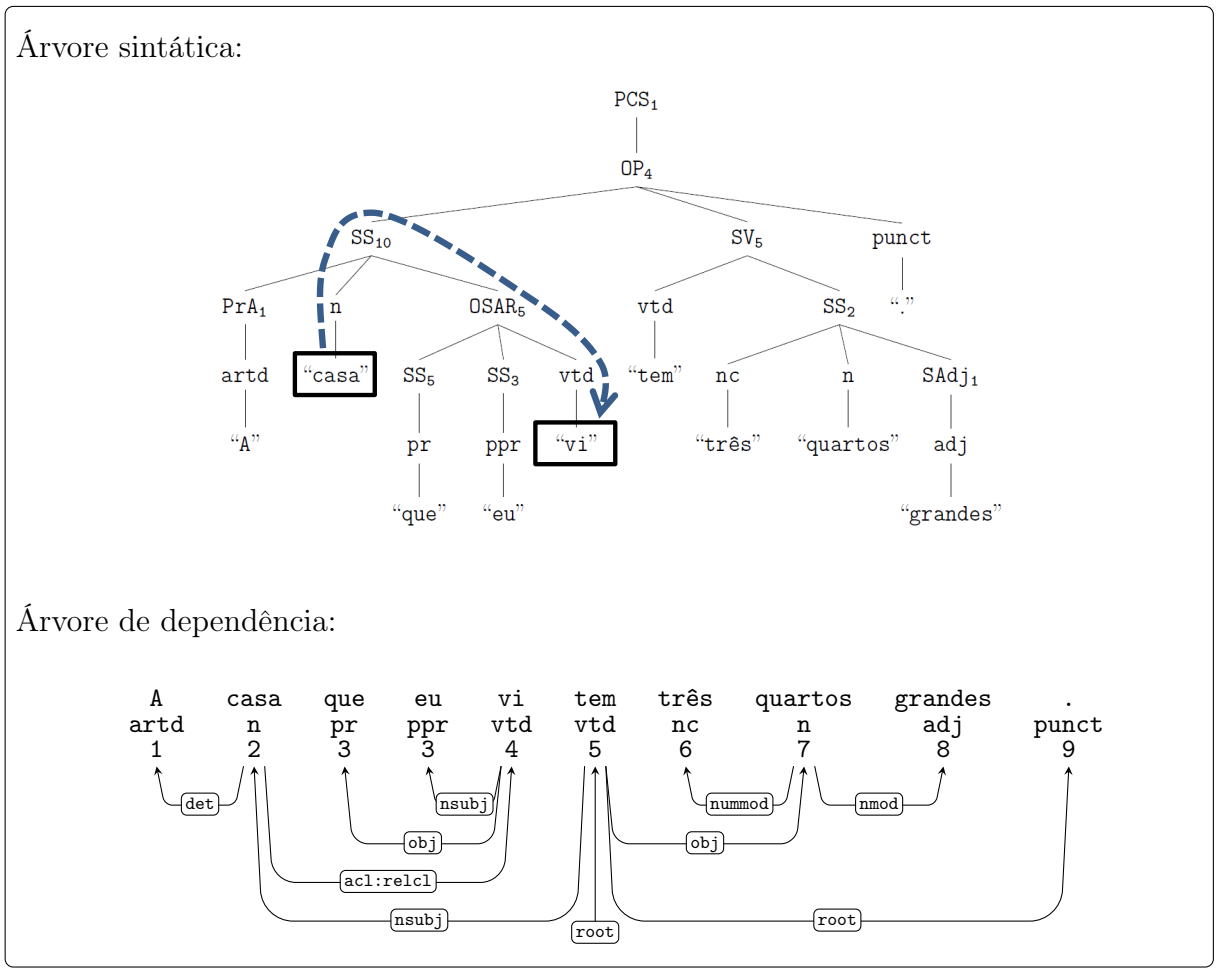
A língua portuguesa permite a construção sintática de sentenças na qual a disposição de seus termos é inversa à sua colocação usual direta (BECHARA, 2009), principalmente para enfatizar algum termo oracional. Estas sentenças na ordem direta ou inversa, apesar de terem disposições diferentes de seus termos, mantém as relações de dependência entre eles.

Considerando esta característica, a partir dos padrões de dependências na ordem direta, realizando operações de movimentação dos elementos constituintes é possível obter os padrões com ordem inversa.

A Figura 4.17 apresenta uma estratégia para tratar a análise de sentenças em ordem inversa usando técnica adaptativa (JOSÉ NETO, 1993) durante a Etapa 1. As regras de produção da gramática que consideram os elementos constituintes na ordem direta poderão ser manipulados de forma a inverter os termos gerando novas regras, e os respectivos novos padrões de relações de dependências a partir daquele existentes.

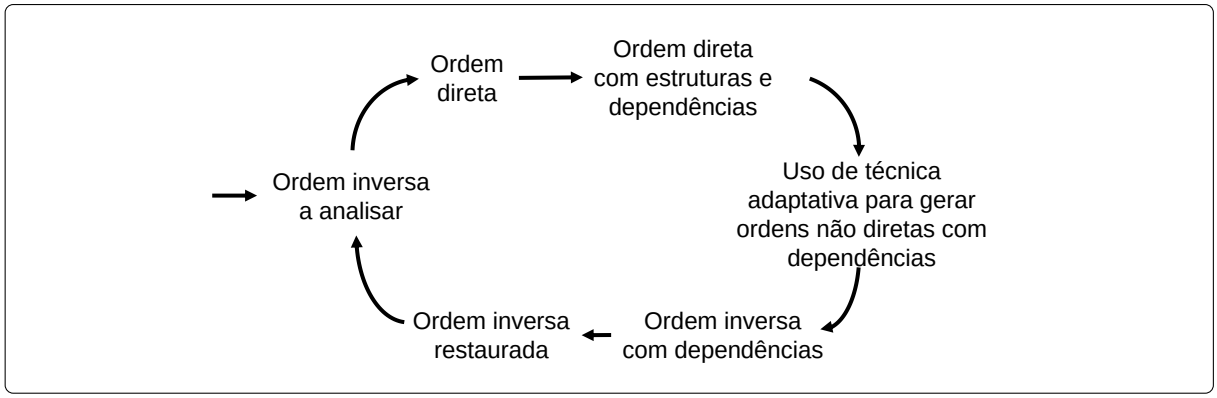
A implementação requer um levantamento das regras de inversão permitidas pela língua portuguesa e uma forma de expressá-las para serem utilizadas no algoritmo adaptativo. O desenvolvimento desta estratégia não está contemplada nesta tese e se constitui num dos temas para a continuidade da pesquisa apresentada.

Figura 4.16: Levantamento de padrões de dependência.



Fonte: autor.

Figura 4.17: Tratamento adaptativo de sentenças em ordem inversa.



Fonte: autor.

4.6 SUMÁRIO

Nesta seção foram detalhados aspectos de projeto do conjunto de sistemas para realizar a análise de sentenças em português brasileiro com o objetivo de se obter ao final do processamento uma árvore com a descrição estrutural da sintaxe da sentença de entrada e as relações de dependência entre as palavras constituintes.

CAPÍTULO 5

ASPECTOS DE IMPLEMENTAÇÃO

5.1 INTRODUÇÃO

Neste capítulo são apresentados aspectos observados durante a implementação do método proposto.

Os sistemas propostos nesta tese foram implementados por meio de módulos de software escritos nas linguagens de programação Java e Python. A decisão da escolha por essas linguagens deveu-se ao fato de oferecerem bibliotecas que implementam rotinas básicas necessárias para a construção dos sistemas, e pela existência de códigos fontes disponibilizados publicamente por outros pesquisadores (BIRD; KLEIN; LOPER, 2009; CEREDA; JOSÉ NETO, 2016; CEREDA, 2018)[1] que serviram de base para o desenvolvimento dos ensaios.

5.2 GERAÇÃO DE ANALISADOR SINTÁTICO LIVRE DE CONTEXTO

Neste item são apresentados aspectos de implementação da construção de um analisador sintático livre de contexto para linguagem natural relacionados às Etapas 1 e 2 do Item 4.2.2 do Capítulo 4.

Ambas as etapas são compostas por módulos baseados no processamento da metalinguagem de descrição de gramáticas livre de contexto na notação de Wirth modificada (JOSÉ NETO, 1993), ou seja, a sua construção segue os mesmos fundamentos usados para a construção de compiladores para linguagens de programação.

A Figura 5.1 apresenta a notação usando ela mesma para a sua descrição.

A seguir são apresentados aspectos relevantes da implementação das 2 etapas.

5.2.1 ANALISADOR LÉXICO PARA GRAMÁTICAS EM NOTAÇÃO DE WIRTH MODIFICADA

Etapa 1

O analisador léxico desta etapa foi construído para gerar uma sequência de *tokens* categorizados a partir da descrição textual S_{wm} de uma gramática na notação Wirth modificada. Ele é baseado no autômato finito determinístico W_M descrito na Figura 5.2 que analisa cadeias $w \in \Sigma^*$.

[1] Disponíveis em <https://github.com/cereda/wsn2spa> e <https://www.nltk.org> Acesso em: 2 de janeiro de 2019.

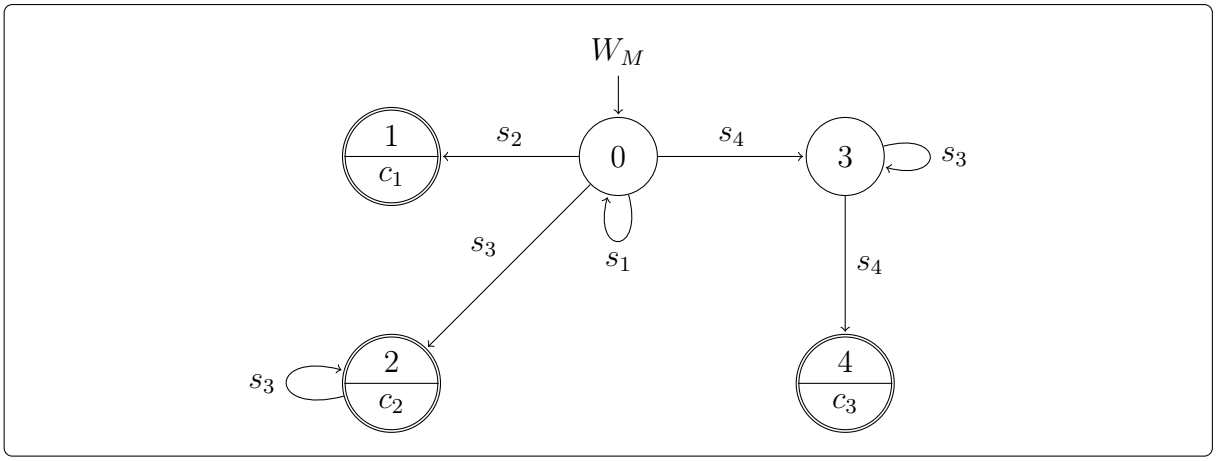
Figura 5.1: Notação de Wirth modificada.

```
syntax = { production } .
production = identifier "=" expression "." .
expression = ( term \ "|" ) .
term = factor ( factor \ ϵ ) .
factor = identifier |
        literal |
        "ϵ" |
        "(" expression ( "\" expression | ϵ ) ")" .
literal = """ ( character \ character ) """ .
```

Fonte: adaptado de José Neto (1993).

O analisador léxico fornece um *token* a cada acionamento do analisador sintático suportando também a operação de empilhamento de elementos, e operação de consulta quanto a existência de mais símbolos na cadeia de entrada.

Figura 5.2: Autômato finito determinístico W_M do analisador léxico para a notação de Wirth modificada.



Fonte: autor.

A Tabela 5.1 descreve o símbolo $\sigma \in \Sigma$ a ser consumido nas transições.

Tabela 5.1: Símbolos a serem consumidos nas transições do autômato finito determinístico W_M para análise de descrição gramatical em Wirth modificada.

Rótulo	Descrição
s_1	$\sigma \in \{\text{␣}, \texttt{\t}, \texttt{\n}\}$
s_2	$\sigma \in \{$ =, (,), |, \, ϵ. $\}$
s_3	$\sigma \in \{$ letras e dígitos $\}$
s_4	$\sigma =$ "

Fonte: autor.

A Tabela 5.2 apresenta as classes gramaticais nas quais a cadeia w pode ser classificada quando ela é aceita e o estado corrente corresponder aos estados 1, 2 e 4. Na figura, na parte inferior destes estados está indicada a respectiva classe obtida, que é um dos componentes da estrutura de dados que representa o *token*.

Tabela 5.2: Classes gramaticais do analisador léxico proposto para a notação de Wirth modificada.

Classe	Descrição	Significado
c_1	$\sigma \in \Sigma$	metassímbolo
c_2	*nonterm*	não terminal
c_3	*terminal*	símbolo terminal

Fonte: autor.

Etapa 2

A Etapa 2 é iniciada somente quando a gramática original está correta e consequentemente foi aceita pelo reconhecedor sintático correspondente na Etapa 1.

O analisador léxico desta etapa utiliza os *tokens* com anotações de rotulação gerados na etapa anterior, ou seja, ele não necessita realizar o processo de gerá-los. São suportadas as operações de fornecer um *token* a cada acionamento do analisador sintático, a operação de empilhamento de elemento e de consulta.

5.2.2 ANALISADOR SINTÁTICO PARA GRAMÁTICAS EM NOTAÇÃO DE WIRTH MODIFICADA

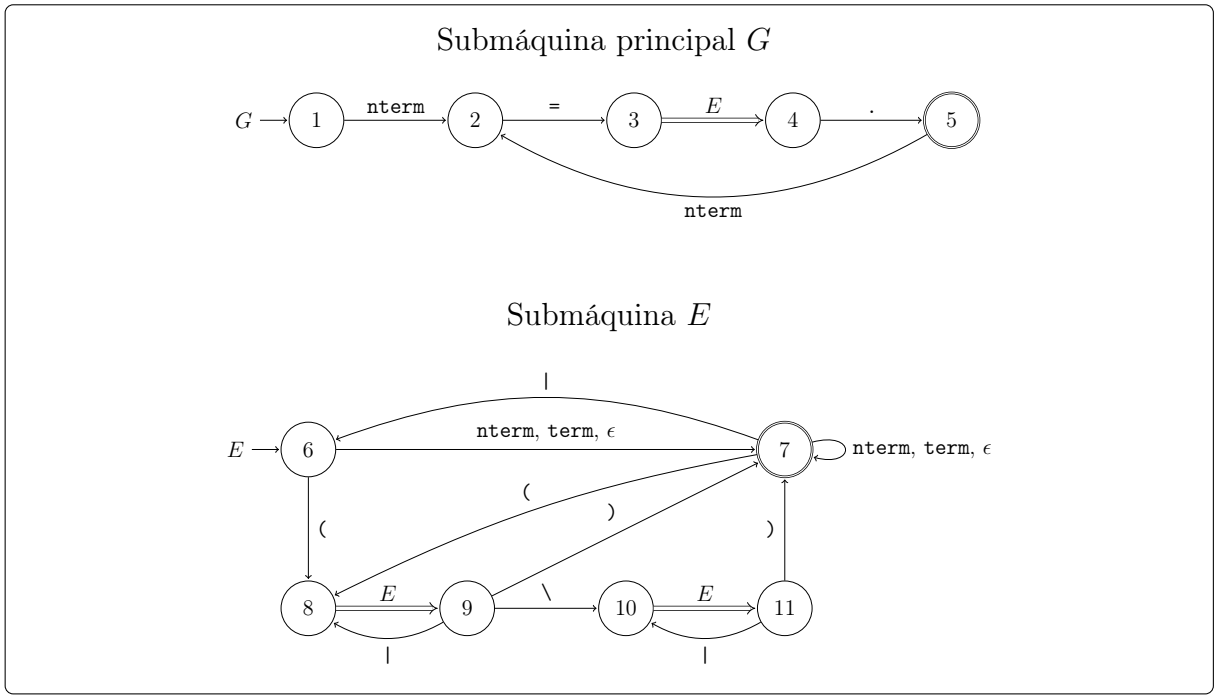
O primeiro passo do algoritmo de José Neto (1993) para geração de analisador sintático a partir de uma gramática na notação de Wirth modificada é a preparação da gramática para adicionar rótulos que tem a função de auxiliar a preservação de informações das regras de produção da gramática original.

Na Etapa 1 é realizado este processo de rotulação que analisa cada sentença que compõe a gramática de entrada utilizando-se para tal um reconhecedor da metalinguagem de Wirth modificada descrito na Figura 5.1 e implementado por um autômato de pilha estruturado G descrito na Figura 5.3. Observe que as expressões `identifier`, `literal` usadas na definição da notação correspondem aos símbolos `nterm` e `term` no autômato.

Este mesmo autômato é usado na segunda etapa, porém com outras ações semânticas para gerar um novo autômato de pilha estruturado que reconhece a linguagem definida pela gramática de entrada.

As ações semânticas associadas às transições do autômato da Figura 5.3 estão indicadas na Tabela 5.3. A penúltima coluna aponta para as figuras relacionadas à ações da Etapa 1 e a última aponta para aquelas da Etapa 2.

Figura 5.3: Autômato de pilha estruturado G para reconhecimento de regras de produção de gramática na notação de Wirth modificada.



Fonte: autor.

Tabela 5.3: Ações semânticas associadas às transições do autômato de pilha estruturado G (Figura 5.3).

Transição	Descrição	Ações Semânticas	
		Etapa 1	Etapa 2
$(1, \texttt{nterm}) \rightarrow 2$	nova submáquina	A.1	A.7
$(2, \texttt{=}) \rightarrow 3$	nova produção	A.2	A.8
$(4, \texttt{.}) \rightarrow 5$	fechamento de produção	A.3	A.9
$(5, \texttt{nterm}) \rightarrow 2$	nova submáquina	A.1	A.7
$(6, \texttt{nterm}) \rightarrow 7$	nova transição com não terminal	A.4	A.10
$(6, \texttt{term}) \rightarrow 7$	nova transição com terminal	A.5	A.10
$(6, \epsilon) \rightarrow 7$	nova transição com ϵ	A.6	A.10
$(6, \texttt{(}) \rightarrow 8$	novo escopo	A.4	A.11
$(7, \texttt{nterm}) \rightarrow 7$	nova transição com não terminal	A.4	A.10
$(7, \texttt{term}) \rightarrow 7$	nova transição com terminal	A.5	A.10
$(7, \epsilon) \rightarrow 7$	nova transição com ϵ	A.6	A.10
$(7, \texttt{\|}) \rightarrow 6$	adição de opção	A.4	A.13
$(7, \texttt{(}) \rightarrow 8$	novo escopo	A.4	A.11
$(9, \texttt{)}) \rightarrow 7$	fechamento de escopo	A.4	A.12
$(9, \texttt{\|}) \rightarrow 8$	adição de opção	A.4	A.13
$(9, \texttt{\textbackslash}) \rightarrow 10$	adiciona reverso	A.4	A.14
$(11, \texttt{)}) \rightarrow 7$	fechamento de escopo reverso	A.4	A.15
$(11, \texttt{\|}) \rightarrow 10$	adição de opção	A.4	A.13

Fonte: autor.

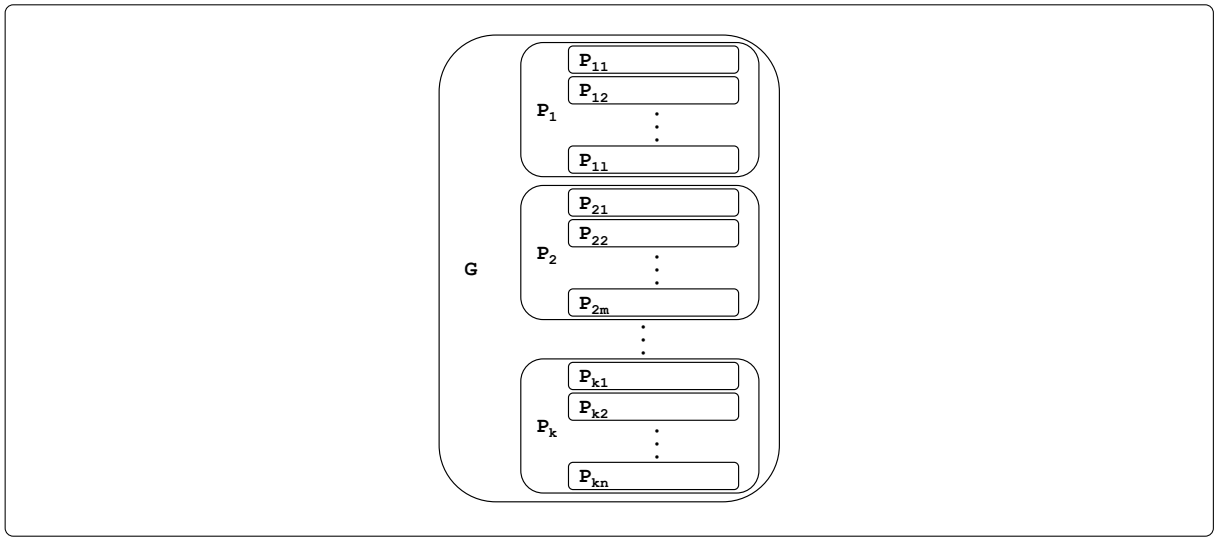
Embora sejam executadas nas mesmas transições as ações semânticas das Etapas 1 e 2 são distintas e possuem conjuntos de variáveis e estrutura de dados auxiliares independentes. Os *tokens* que estas rotinas recebem como parâmetro de entrada também diferem quanto aos atributos adicionais que contêm.

Etapa 1

Na Etapa 1 o transdutor tem por objetivo construir uma estrutura de dados que permita a manipulação das regras de produção para preparar a gramática para a Etapa 2 e para gerar uma sequência de novos *tokens* para essa fase seguinte de processamento.

A Figura 5.4 apresenta uma descrição esquemática de uma estrutura de uma gramática `G`, formada por uma lista de conjuntos P_1 a P_k de regras de produção. Cada conjunto P_i é formada por uma lista ordenada de regras associadas a um único não terminal.

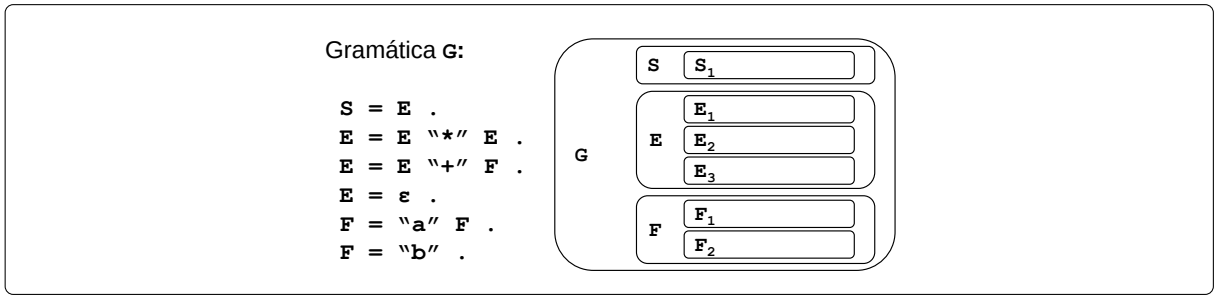
Figura 5.4: Visão esquemática da estrutura de dados para gramática.



Fonte: autor.

A Figura 5.5 ilustra a estrutura para a gramática exemplo `G`.

Figura 5.5: Exemplo de estrutura de dados de gramática.

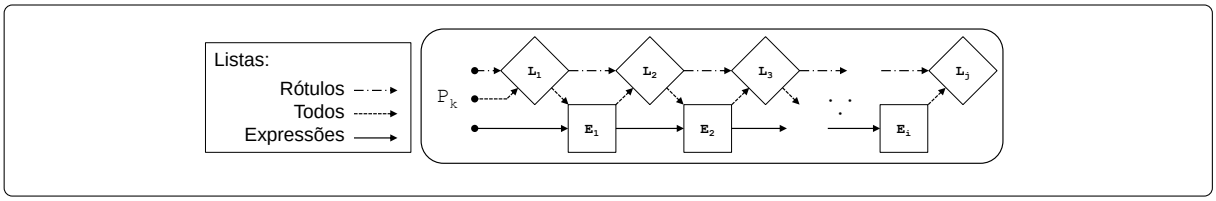


Fonte: autor.

Cada regra de produção individual da estrutura da Figura 5.4 é formada por 3 listas conforme esquema da Figura 5.6. Uma lista de elementos da expressão é formada a partir

dos *tokens* $\mathtt{E_1}$ a $\mathtt{E_i}$ extraídos pelo analisador léxico. Outra lista é formada pelos rótulos $\mathtt{L_1}$ a $\mathtt{L_j}$ obtidos a partir dos elementos anteriores seguindo o algoritmo proposto por José Neto (1993), e que podem ser dos tipos descritos na Figura 5.4. Os elementos das duas listas são interligadas numa terceira lista mista, intercalando elementos dos rótulos e da expressão.

Figura 5.6: Visão geral da estrutura de dados para uma regra de produção.



Fonte: autor.

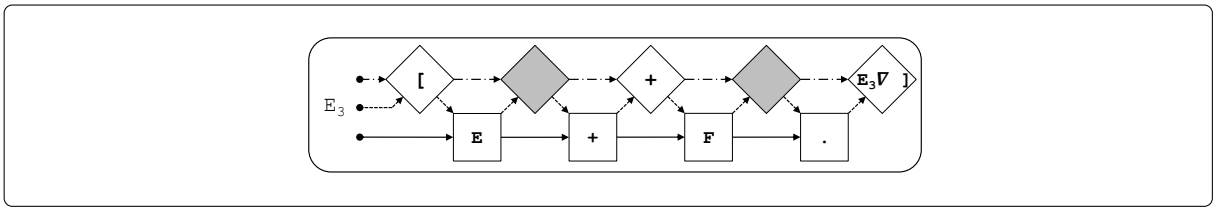
Tabela 5.4: Regras de mapeamento de rótulos e saídas.

Rótulo	Descrição
ϵ	Cadeia vazia.
σ	Terminal.
$X_i\nabla$	Não terminal X_i com autorrecursão à esquerda.
X_i	Não terminal X_i sem autorrecursão.
∇X_i	Não terminal X_i com autorrecursão à direita.
[	Início de regra de produção.
]	Fim de regra de produção.

Fonte: adaptado de José Neto (1993).

A Figura 5.7 apresenta a estrutura acima descrita para a regra $\mathtt{E_3}$ da gramática exemplo.

Figura 5.7: Exemplo de estrutura de dados para uma regra de produção.

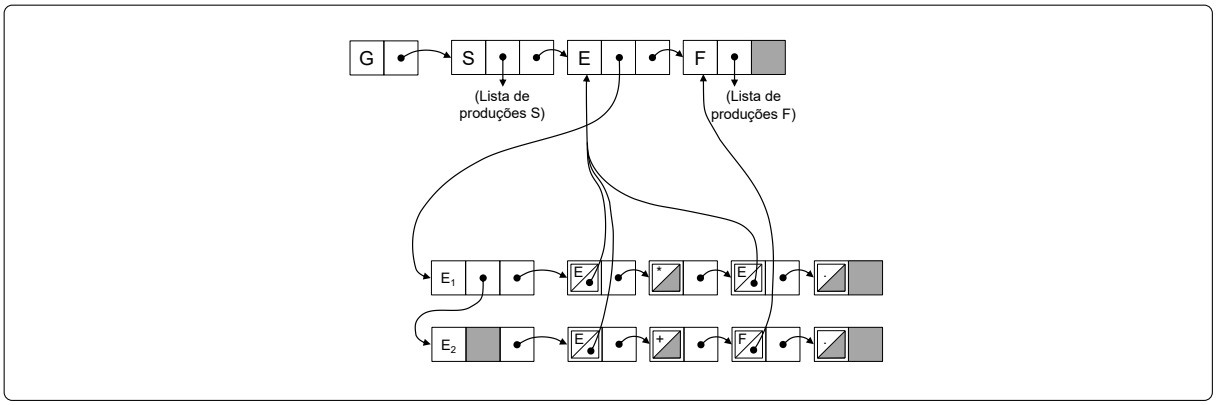


Fonte: autor.

Os elementos que representam não terminais na lista de uma expressão são construídos de forma que possam referenciar a estrutura de dados que permite acessar todas as outras instâncias do mesmo não terminal. A Figura 5.8 apresenta essa relação para as regras $\mathtt{E_1}$ e $\mathtt{E_2}$ da gramática exemplo através de um desenho que utiliza a representação de ponteiros para lista ligada.

Os elementos individuais da lista de rótulos por sua vez são também representados por uma lista, para facilitar operações como de concatenação na manipulação de regras. O último elemento da lista de rótulos do exemplo da Figura 5.7 é formado por uma lista de dois rótulos "$\texttt{E}_3\nabla$" e "`]`". Na implementação, a lista interligando os elementos da expressão e dos rótulos, e as informações referentes a existência de autorrecursão são obtidas após o processamento de todas as regras, através de uma varredura da estrutura de dados criada.

Figura 5.8: Relação entre elementos da expressão de uma regra com a lista de não terminais.



Fonte: autor.

As Figuras A.1 a A.6 apresentam os trechos relevantes do código em Java das funções que implementam as ações semânticas para gerar a estrutura de dados. Todas as ações recebem como parâmetro de entrada o *token* consumido na transição que contém informações quanto ao seu tipo `token.type`, que corresponde à classe gramatical listada na Tabela 5.2, e o seu valor `token.value`, que é a palavra presente na cadeia de entrada. As ações compartilham duas variáveis globais: `currentNterm` e `currentProduction` que são respectivamente o não terminal corrente para o qual está sendo gerada a estrutura de dados, e a estrutura de dados da regra de produção para a qual estão sendo geradas as listas de elementos da expressão e rótulos.

Para o registro dos elementos das listas as funções abaixo são usadas:

- `registerExpressionToken(Token token)`: adiciona o *token* na lista de elementos da expressão da regra de produção corrente e na lista mista.
- `registerLabelToken(String value)`: adiciona o rótulo `value` na lista de rótulos e na lista mista.

Na ação descrita na Figura A.3, que é invocada ao final de uma expressão de produção, no último elemento do tipo rótulo são inseridos na lista os rótulos `"]"`e o identificador do não terminal cuja expressão está sendo finalizada. Por exemplo, o último elemento de rótulo da produção $\texttt{E}_3$ conterá a lista `{"`$\texttt{E}_3$`", "]"}`.

Etapa 2

A cadeia de entrada nesta etapa é composta por uma sequência de *tokens* obtidos a partir da estrutura de dados gerado na etapa anterior (Figura 5.4) fornecido pelo analisador léxico desta Etapa 2. Cada *token* contém informações sobre a sua classe gramatical e a palavra na gramática original a partir da qual foi obtida a sua classificação (Figura 5.2) além de informações sobre rótulos.

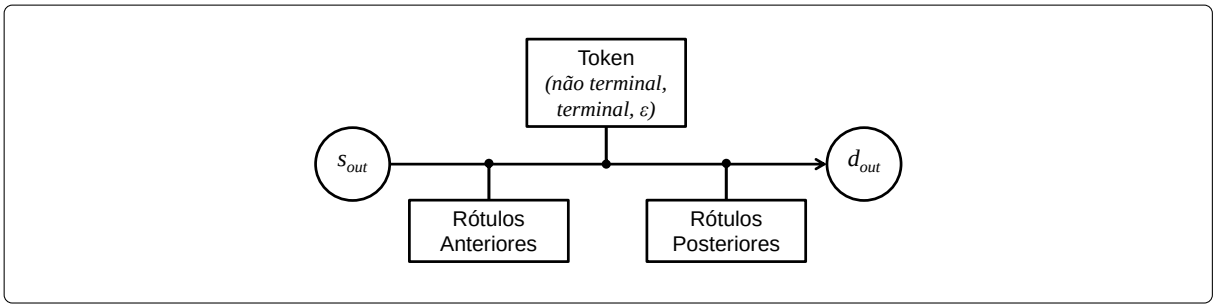
Nesta etapa o transdutor baseado no autômato da Figura 5.3 ao reconhecer com sucesso a cadeia de entrada que descreve uma gramática na notação de Wirth modificada, gera na sua saída uma estrutura de dados a ser posteriormente utilizada na construção de um autômato de pilha estruturado que reconhece a linguagem por ela definida.

Para cada não terminal existente na gramática o algoritmo gera uma estrutura de dados que representa um autômato finito não determinístico, não simplificado, com um único estado de saída. Esta topologia é obtida com o uso de transições em vazio, que numa fase posterior de processamento são eliminadas explorando a característica de se ter somente um estado de saída. Para facilitar a descrição das estruturas de dados de saída e ações associadas às mesmas, elas serão nomeadas com o sufixo *out* nos parágrafos a seguir.

A estrutura de dados de saída que descreve uma transição $trans_{out}$ é composta por informações a serem utilizadas pelas ações semânticas do transdutor em tempo de execução na análise de uma sentença. Os dados são organizados de modo que sejam processados por ações semânticas executadas numa sequência pré-determinada.

A Figura 5.9 esquematiza as principais estruturas de dados associadas a uma transição $trans_{out}$ do estado de origem s_{out} para o destino d_{out}. A Figura 5.10 apresenta ações semânticas associadas, cujas execuções devem ser realizadas sequencialmente da direita para a esquerda.

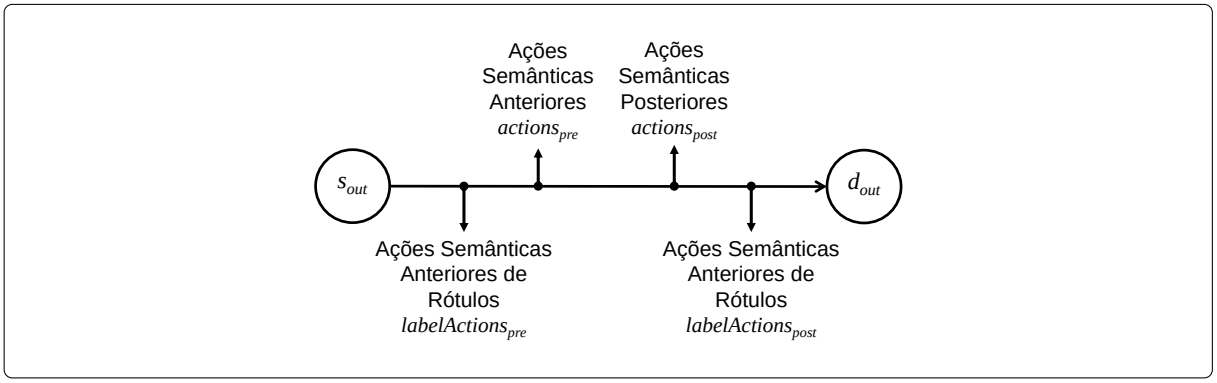
Figura 5.9: Estrutura de dados associadas a uma transição $trans_{out}$.



Fonte: autor.

As ações semânticas deverão ser executadas obedecendo a sequência apresentadas na Figura da esquerda para a direita. São considerados dois momentos de execução. O primeiro momento é logo no início da transição, sendo por isso as ações chamadas de

Figura 5.10: Ações semânticas associadas a uma transição $trans_{out}$.



Fonte: autor.

anteriores (sufixo *pre*), e o segundo é antes da efetiva mudança do estado corrente do autômato para o estado destino, chamadas de posteriores (sufixo *post*).

As ações semânticas de rótulos ($labelActions_{pre}$ e $labelActions_{post}$) executam o algoritmo de José Neto (1993) para a geração da árvore sintática. São executadas cada qual no seu momento com a sua lista específica de rótulos. Esta diferenciação é feita para garantir que as informações sintáticas provenientes das regras gramaticais possam ser geradas na sequência correta em tempo de execução, mesmo que tenha havido alterações na topologia do autômato decorrentes de simplificações como a eliminação de transições em vazio e de não determinismos conforme apresentado mais adiante neste item.

As demais ações semânticas ($actions_{pre}$ e $actions_{post}$) poderão ser utilizadas para a implementação de quaisquer outras rotinas. Nos experimentos realizados, uma ação posterior associada a transições que consomem um *token* contendo um símbolo terminal (que representa uma categoria gramatical) imprime na cadeia de saída a palavra da sentença de entrada correspondente ao terminal. Estas ações semânticas podem ser também utilizadas para realizar ações adaptativas (JOSÉ NETO, 1993) para alterações no próprio autômato.

A Tabela 5.3 apresenta ações semânticas relacionadas às transições do autômato desta etapa de geração das estruturas de saída. Elas compartilham as variáveis globais cujas descrições estão na Tabela 5.5 e os tipos em Java na Figura 5.11.

O tipo `sketch` usado na definição da variável `transitions` é uma estrutura de dados contendo informações necessárias para a construção de transições. Na sua inicialização são especificados os seguintes dados: `name` – identificador; `source` – estado origem; `token` – *token* que provoca a transição; `target` – estado destino.

As Figuras A.7 a A.15 apresentam as ações semânticas executadas nas transições escritas na linguagem Java. Estas rotinas recebem como parâmetro de entrada a estrutura de dados `token` que provocou a transição e contém as informações quanto ao seu tipo (`token.type`), ao seu valor (`token.value`) correspondente à palavra da gramática de entrada, e uma lista de rótulos associados (`token.nextLabels`). Observe que este *token*

Tabela 5.5: Etapa 2 - Variáveis globais.

Variável	Descrição
`helperPairStack`	Pilha de pares de contadores de estados com métodos `getFirst` para obter o primeiro elemento e `getSecond` para o segundo. Eles definem estados inicial e final de um escopo.
`transitions`	Lista de estrutura de dados de transições.
`current`	Estado corrente.
`counter`	Contador de estados.
`main`	Submáquina principal.
`machine`	Submáquina corrente.

Fonte: autor.

Figura 5.11: Etapa 2 – Tipos das variáveis globais.

```
Stack<Pair<Integer, Integer>> helperPairStack;
List<Sketch> transitions;
int current;
int counter;
String main;
String machine;
```

Fonte: autor.

corresponde a um elemento da lista de expressões da estrutura de dados de uma regra de produção construído na Etapa 1 anterior, e a lista de rótulos apontados por `nextLabels` são originários do elemento subsequente ao *token* na lista mista de elementos.

A ação da Figura A.7 realiza a inicialização de estruturas de dados para a construção de uma nova submáquina. Nesta implementação está convencionado que a primeira submáquina referente à primeira regra da gramática é a submáquina principal do autômato de pilha estruturado.

Nas ações semânticas de finalização de uma regra de produção (Figura A.9), de fechamento de escopo (Figura A.12) criação e fechamento de uma transição reversa (Figuras A.14 e A.15) o *token* de entrada é reaproveitado atribuindo-se o símbolo ϵ a `token.type` e `token.value` para indicar uma transição em vazio.

Ao final do processamento do transdutor baseado no reconhecedor de gramática em Wirth modificada é obtida uma estrutura de dados que descreve um novo autômato de pilha estruturado. Este autômato não é determinístico, nem simplificado.

Nos experimentos foram implementadas técnicas de eliminações parciais de transições em vazio e de não determinismos em autômatos de pilha com um critério adicional de preservar a sequência de rótulos das produções originais. Ambos os tratamentos são descritos a seguir.

A eliminação da transição em vazio é realizada somente nas situações descritas na Tabela 5.6 que também indica qual dos estados origem (*source*) ou destino (*target*) é

eliminado na operação. A eliminação é decidida com base na quantidade de transições que são originadas (*sourceOutQty*) e terminadas (*sourceInQty*) nos estados origem e destino (*targetOutQty* e *targetInQty* respectivamente) que a compõe, e considerando a manutenção da característica de se ter somente um estado de saída em cada submáquina.

Tabela 5.6: Condições para eliminação de transição em vazio.

Item	$source(q_s)$		$target(q_t)$		Estado a eliminar
	In (*sourceInQty*)	Out (*sourceOutQty*)	In (*targetInQty*)	Out (*targetOutQty*)	
1	0	> 0	1	> 0	q_t
2	> 0	> 1	1	> 0	q_t
3	1	1	> 0	–	q_s
4	> 1	1	1	–	q_s
5	> 1	1	> 1	> 0	q_s

Fonte: autor.

As Figuras 5.12 e 5.13 apresentam exemplos de eliminação de transições em vazio. Em ambas as figuras, a situação inicial está representada na parte superior onde a transição em vazio p_{st} entre o estado origem q_s e destino q_t está grafada com três informações: o token ϵ, as informações de rótulos anteriores $l_{\epsilon_{pre}}$ e posteriores $l_{\epsilon_{post}}$.

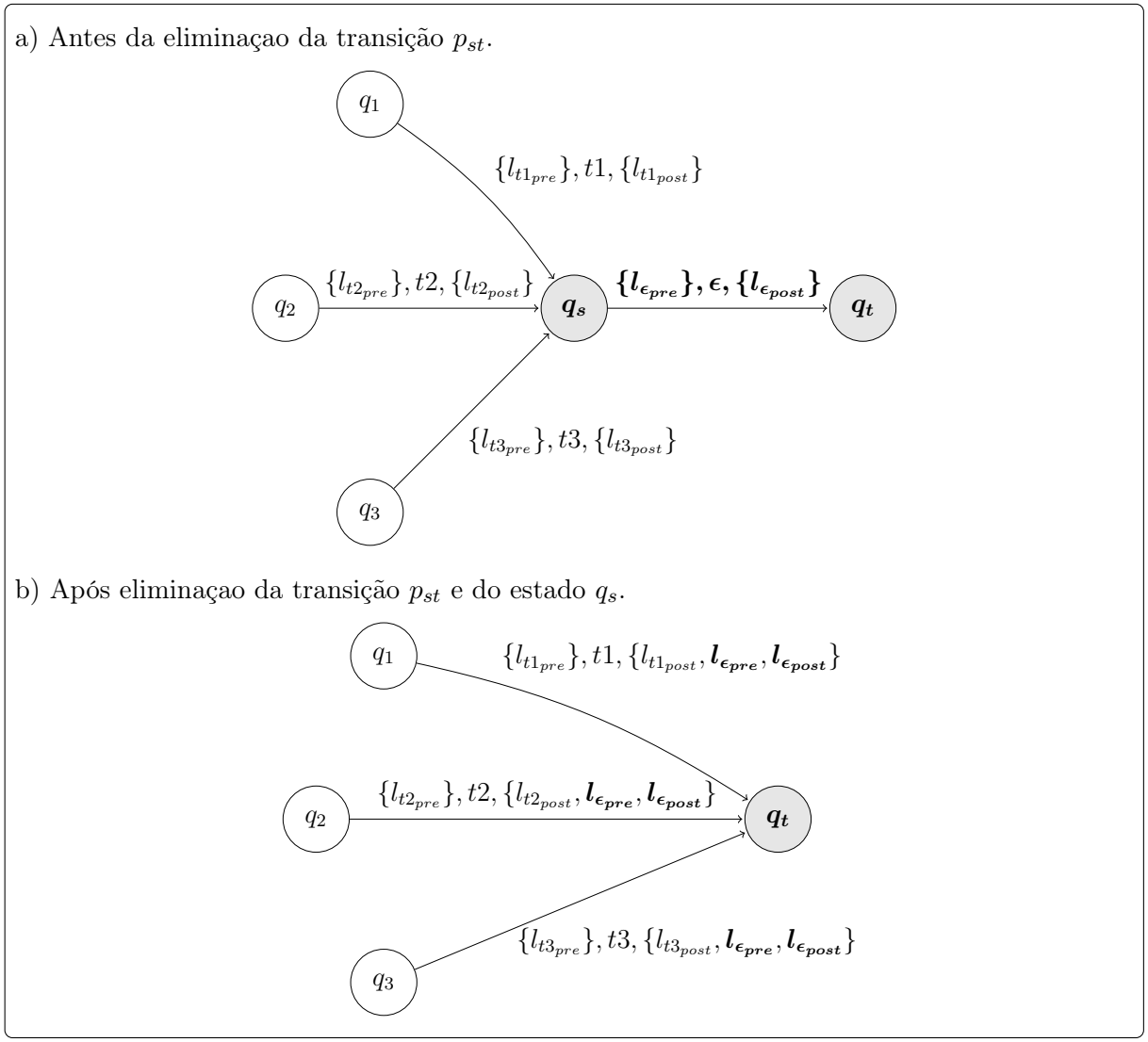
No primeiro caso estão representadas as transições que tem como destino o estado origem (q_s) que é eliminado no processo de eliminação da transição em vazio. No segundo caso estão representadas as transições que se iniciam no estado destino (q_t) que é eliminado.

Ao eliminar uma transição em vazio os rótulos devem ser manipulados de forma a manter a sequência em que seriam processados durante a execução do autômato caso não ocorresse a eliminação. Por exemplo, na Figura 5.12 os rótulos $l_{\epsilon_{pre}}$ e $l_{\epsilon_{post}}$ da transição em vazio são copiados após a estruturas de dados dos rótulos posteriores das 3 transições anteriores que tinham como destino o estado q_s antes da eliminação. Na Figura 5.13 é apresentada a outra situação em que os rótulos da transição em vazio são copiados de forma que sejam processados antes dos rótulos anteriores das transições que finalizariam no estado q_t a ser eliminado.

Após a eliminação de alguma transição em vazio, as análises da existência de mais transições deste tipo e da viabilidade de sua deleção devem ser refeitas até que não haja mais possibilidade de se efetuar outra eliminação.

Nos experimentos foi implementada a eliminação parcial de não-determinismos decorrentes de transições múltiplas que consomem um mesmo símbolo terminal ou uma mesma submáquina a partir de um mesmo estado de origem. Se estas transições tiverem estruturas de dados de rótulos também coincidentes é criado um estado auxiliar que incorpora os todos os respectivos estados destinos. A partir deste estado auxiliar será emanado o conjunto de todas as transições subsequentes que se originavam dos estados destinos

Figura 5.12: Eliminação de transições em vazio - Caso 1.



Fonte: autor.

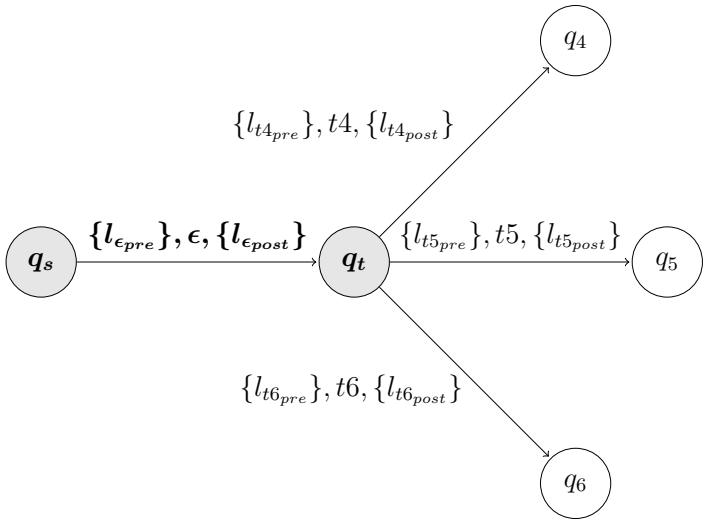
existentes antes da alteração. A Figura 5.14 apresenta um exemplo no qual originalmente existem 3 transições com o consumo de um mesmo símbolo a partindo de um mesmo estado e a simplificação é realizada somente com duas delas que possuem o mesmo conjunto de dados de rótulos.

A Figura 5.15 apresenta um exemplo de aplicação da eliminação de transições em vazio e de não-determinismos na submáquina de reconhecimento de sintagmas nominais. A parte superior da figura apresenta a submáquina com a topologia original gerada pelo transdutor da Etapa 2, e a parte superior a mesma submáquina simplificada. O desenho desta figura foi gerada com o pacote *GraphViz* [2] (GANSNER; NORTH, 2000) a partir da saída no formato "dot" (GANSNER; KOUTSOFIOS; NORTH, 2015).

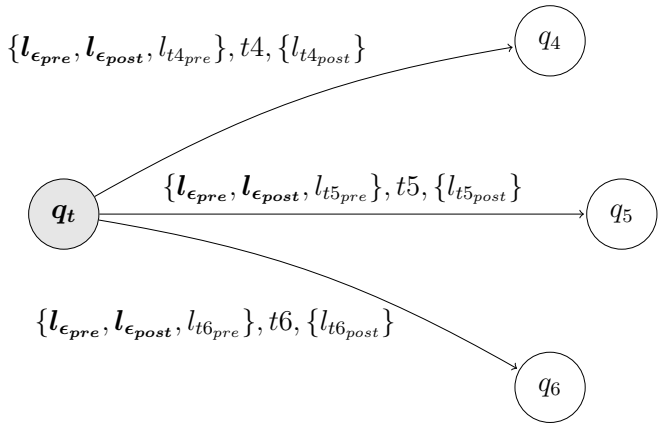
[2]Disponível em <https://www.graphviz.org> Acesso em: 2 de janeiro de 2019.

Figura 5.13: Eliminação de transições em vazio - Caso 2.

a) Antes da eliminaçao da transição p_{st}.

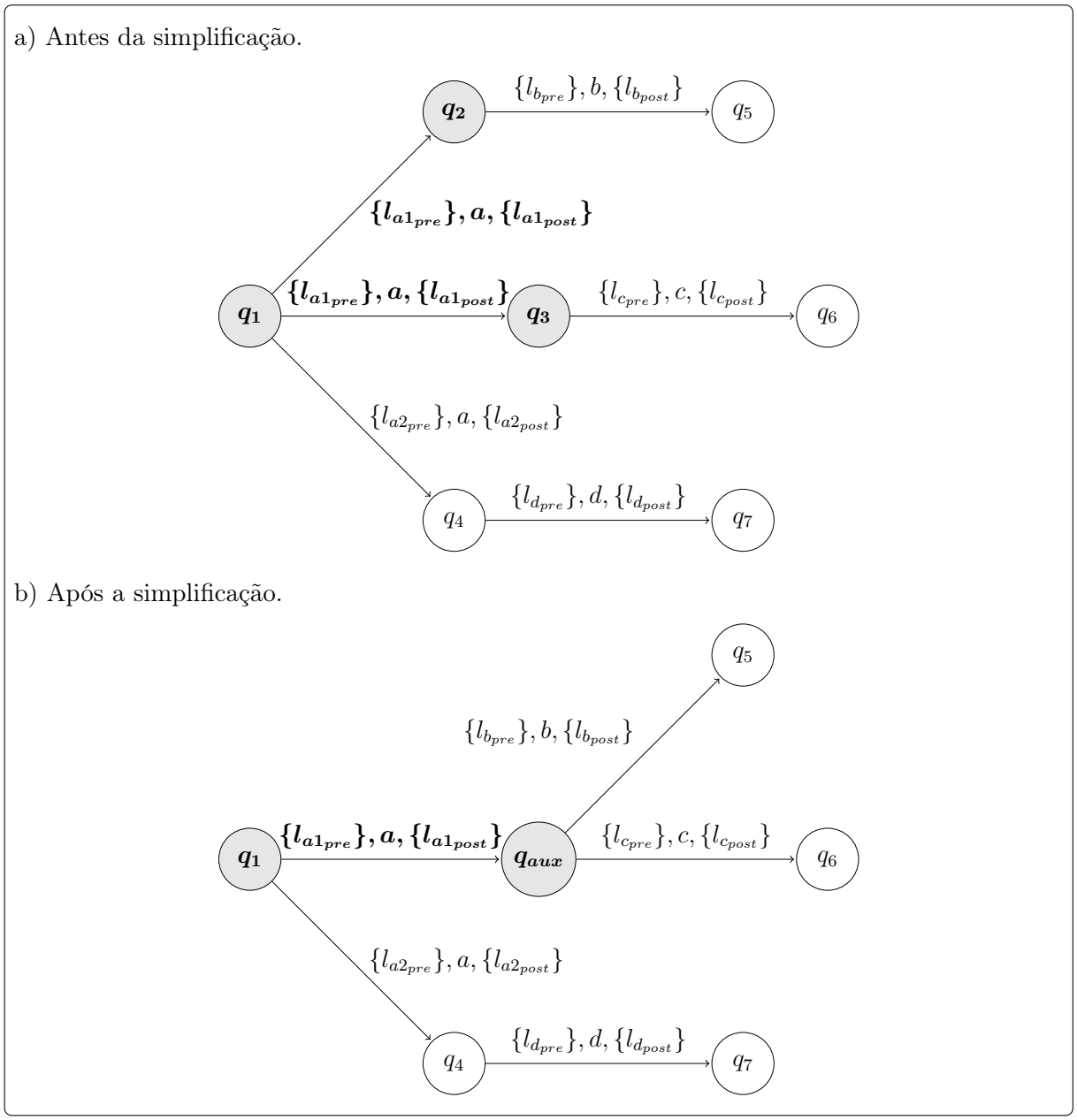


b) Após eliminaçao da transição p_{st} e do estado q_t.



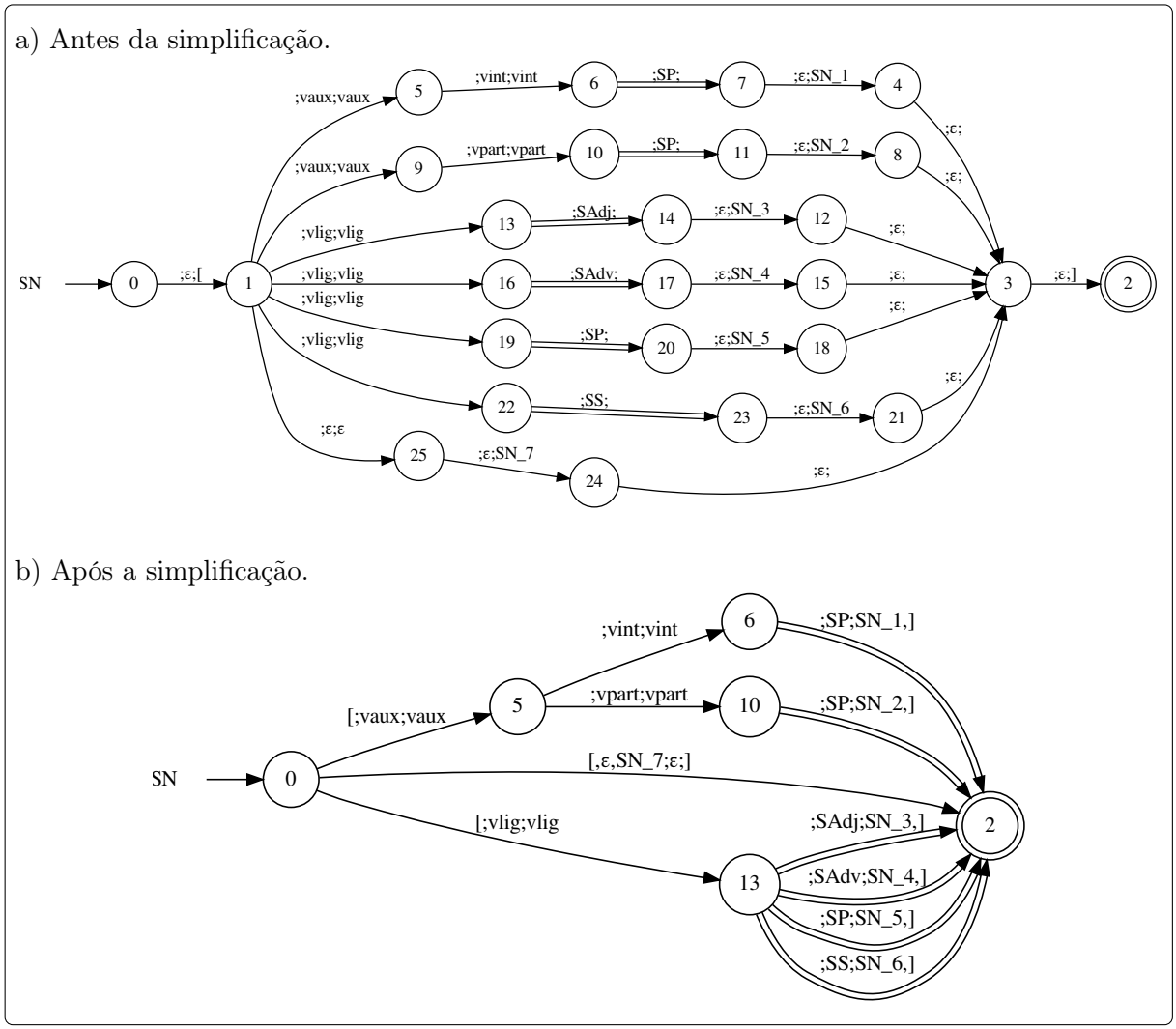
Fonte: autor.

Figura 5.14: Exemplo de eliminação parcial de não-determinismo.



Fonte: autor.

Figura 5.15: Simplificação de submáquina de reconhecimento de sintagma nominal.



Fonte: autor.

5.3 CONSTRUÇÃO DO ANALISADOR SINTÁTICO LIVRE DE CONTEXTO PARA PORTUGUÊS BRASILEIRO

5.3.1 ANALISADOR LÉXICO PARA PORTUGUÊS BRASILEIRO

O analisador léxico-morfológico é acionado pelo analisador sintático fornecendo *tokens* gerados a partir da cadeia de entrada constituída por uma sentença na língua portuguesa brasileira. Ele permite operações de retorno de *token* comportando-se como uma pilha. Durante a sua execução ele utiliza informações coletadas previamente e armazenadas de forma estruturada, ou seja, a sua implementação é dividida em duas partes. A primeira é referente à construção da base de informações 5.3.1, e a segunda é a responsável pela análise propriamente dita 5.3.1.

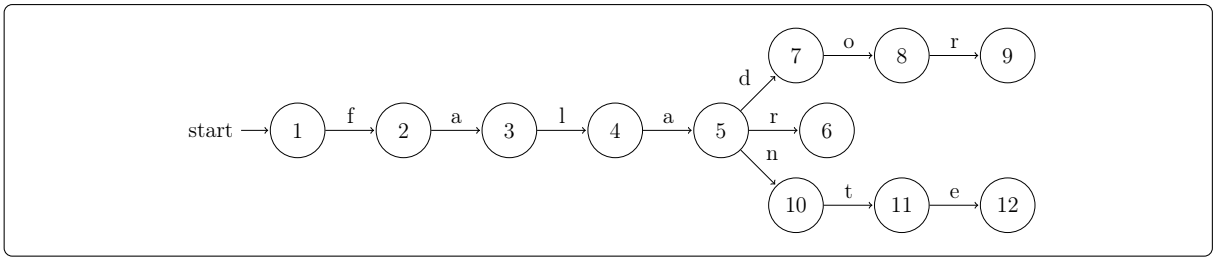
Construção da estrutura de dados

Para a realização dos experimentos foi implementado um dicionário e um flexionador para verbos, substantivos e adjetivos que se flexionam de forma regular. A construção é composta de módulos de processamentos.

O primeiro consiste na construção da estrutura de dados de dicionário a partir de corpus anotado e outros dicionários armazenados em arquivos. A este módulo, de acordo com a classe gramatical das palavras coletadas são adicionados módulos flexionadores.

A coleta de palavras de dicionários é baseado no autômato adaptativo coletor de nomes apresentado por José Neto (1993). O objetivo deste algoritmo era o de coletar identificadores em linguagens de programação para o seu posterior uso num analisador léxico. Na variante aqui utilizada o objetivo é a construção de um autômato reconhecedor de palavras do português brasileiro que adicionado a rotinas semânticas implemente um transdutor que na sua saída apresente qual é a categoria de cada palavra e outros atributos gramaticais. Na sua configuração final este autômato apresenta uma topologia em que palavras com prefixos comuns compartilham o trecho inicial da trajetória de reconhecimento. A Figura 5.16 apresenta parte da configuração deste autômato que aceita as palavras “falador”, “falar” e “falante”.

Figura 5.16: Configuração parcial do autômato adaptativo coletor de nomes.



Fonte: autor.

Ao final da palavra aceita é gerado um estado adicional e uma transição com um símbolo marcador de fim de palavra que não faça parte dos caracteres que compõe a língua analisada. A este estado final é associada uma estrutura de dados contendo as categorias gramaticais que a palavra pode ser classificada e os atributos adicionais. Além disso, se as categorias corresponderem àquelas que são flexionadas de forma regular, haverá a indicação de se alterar a configuração do autômato adicionando uma submáquina para reconhecer as possíveis alternativas de sufixos. Por exemplo, no estado 9 da Figura 5.16 as seguintes informações seriam associadas à palavra "falador":

- Adjetivo, masculino, singular, flexionador para adjetivo
- Substantivo, masculino, singular, flexionador para substantivo

No caso dos verbos regulares, após o reconhecimento de seu infinitivo, o autômato alterará a sua configuração para acrescentar transições e estados a partir do estado no qual é reconhecido o seu radical para que todos os casos de flexão sejam reconhecidos. Por exemplo, na Figura 5.16 após o reconhecimento da cadeia "falar", o autômato seria alterado para incluir todas os sufixos correspondentes à sua conjugação verbal a partir do estado 4, no qual é reconhecido a radical "fal-".

O arquivo com o dicionário ou corpus deve estar codificado em UTF-8 (YERGEAU, 2003). As palavras, as respectivas categorias gramaticais e atributos adicionais devem estar no formato do *Penn Treebank* (TAYLOR; MARCUS; SANTORINI, 2003), que é uma linguagem de anotação sintática. Para a leitura e a extração de informações do arquivo foi utilizada a biblioteca `BracketParseCorpusReader` do *Natural Language Toolkit* (NLTK)[3] (BIRD; KLEIN; LOPER, 2009).

Executor da análise léxico-morfológica

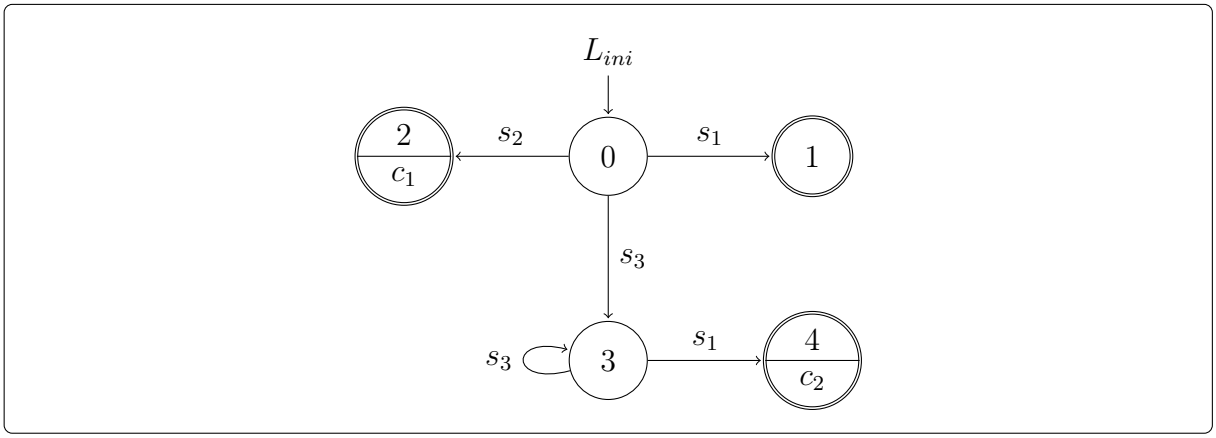
O processamento da cadeia de entrada é realizada caractere por caractere para identificar as subcadeias que formam os *tokens*. Na língua portuguesa o alfabeto é latino e as palavras são separadas por caracteres de espaço, símbolos de pontuação, ou caracteres de controle para manipulação por computadores como tabulação, mudança de linha, entre outros.

De forma hierárquica, o processo se inicia com um autômato finito da Figura 5.17 que tem por objetivo segmentar a cadeia de entrada, gerando subcadeias que são candidatos a constituir os *tokens* a serem analisadas pelo autômato do Item 5.3.1 anterior. A Tabela 5.7 descreve o símbolo $\sigma \in \Sigma$ a ser consumido nas transições, onde Σ é o conjunto de todos os caracteres que podem ser utilizados na escrita de uma sentença em codificação UTF-8. Ao reconhecer uma subcadeia nos estados de aceitação 2 e 4, este autômato identifica o *token* obtido em sinal de pontuação (c_1) ou se é uma palavra (c_2).

Se a subcadeia é um sinal de pontuação, o processamento do *token* é concluído e o analisador léxico morfológico repassa o resultado para o analisador sintático.

[3]Disponível em <http://www.nltk.org/howto/corpus.html> Acesso em: 2 de janeiro de 2019.

Figura 5.17: Autômato finito determinístico inicial L_{ini} do analisador léxico-morfológico.



Fonte: autor.

Tabela 5.7: Símbolos a serem consumidos nas transições do autômato finito determinístico L_{ini}.

Rótulo	Descrição
s_1	$\sigma \in \{\text{␣}, \texttt{\textbackslash t}, \texttt{\textbackslash n}, \text{' , " }\}$
s_2	$\sigma \in \{\text{símbolos de pontuação}\}$
s_3	$\sigma \notin \{\text{símbolos de pontuação}, \text{␣}, \texttt{\textbackslash t}, \texttt{\textbackslash n}, \text{' , " }\}$

Fonte: autor.

Se a subcadeia for uma palavra, ela passará por etapas de análise em outros autômatos seguindo passos similares ao analisador morfológico proposto por Contier, Padovani e José Neto (2017) que emprega técnicas adaptativas.

Primeiramente é verificada se a palavra é um termo invariável, ou seja, aquele que os linguistas consideram que a classificação gramatical estável, como preposições, conjunções e pronomes. Nesta etapa palavras resultantes da aglutinação de outras são separadas. Por exemplo, a palavra "da", que é a junção da preposição "de" com o artigo definido "a", é desmembrada nestas duas palavras que a constituem.

No passo seguinte, é utilizado o autômato adaptativo descrito no item 5.3.1 para se obter as possíveis classificações gramaticais.

5.3.2 ANALISADOR SINTÁTICO PARA PORTUGUÊS BRASILEIRO

Geração do ambiente de execução do transdutor livre de contexto

A estrutura de dados construída na Etapa 2 (Item 5.2.2) que descreve um autômato de pilha estruturado é utilizada para a geração da máquina de estado que implementa o transdutor livre de contexto.

Conforme exposto no item 5.2.2 em cada transição são executadas em sequência as ações semânticas para processar os dados de rótulos anteriores, a ação semântica com o *token* e as ações relacionados aos rótulos posteriores.

Existem três tipos de transições de acordo com o tipo do *token* com o qual é feita a mudança de estado, a saber "não terminal", "terminal" ou em vazio ("ϵ"). Na construção do autômato, na transição provocada por um *token* do tipo terminal é associada a ação semântica `ActionTermTransition` (Algoritmo 5.1 do item 5.4). Esta ação insere na cadeia de saída o conteúdo do *token*, que corresponde ao texto da palavra da cadeia de entrada a partir do qual foi obtido o *token*.

Em todas as transições, caso estejam presentes, os rótulos são processados por uma mesma ação semântica. A rotina processa cada elemento individual da lista de rótulos, que é passada como parâmetro. Se o rótulo faz referência a alguma regra de produção, a rotina acessa também a informação sobre o tipo de recursão existente (à direita ou à esquerda) para definir a cadeia de saída. Os Algoritmos 5.2 a 5.8 responsáveis pelo tratamento de cada tipo de rótulo estão indicados na Tabela 5.8. Como estes algoritmos também tratam da análise de padrões de dependências, as suas descrições estão no item 5.4. Nestas descrições, os trechos destacados na cor preta referem-se ao pseudocódigo do transdutor livre de contexto e o texto em azul são específicos à análise de dependências. O resultado do processamento é apresentado na cadeia de saída pelos comandos de ***print***.

As rotinas utilizam uma variável global `transducerStack` com estrutura de pilha para implementar as regras de mapeamento de rótulos proposto por José Neto (1993) e que também estão descritas na Tabela 5.8. A primeira coluna da tabela contém os elementos de rótulo, cujos significados estão na Tabela 5.4. Para cada tipo de rótulo, as colunas seguintes apresentam o que deve ser apresentado na cadeia de saída, a ação de movimentação da pilha `transducerStack`, e a indicação da descrição do algoritmo de tratamento já mencionado anteriormente. O símbolo π na tabela representa o conteúdo da pilha `transducerStack` entre o topo e a primeira ocorrência do delimitador "`]`" não incluindo este último. Os símbolos $\uparrow$ e $\downarrow$ são usados para indicar as operações de empilhamento e desempilhamento de um elemento da pilha.

Tabela 5.8: Regras de mapeamento de rótulos e saídas.

Rótulo	Saída	Pilha	Algoritmo da ação semântica
ϵ	`()`		5.2
σ	`(`σ`)`		5.3
`X`$_i\nabla$	`X`$_i$`)`		5.6
`X`$_i$	`X`$_i$`)`π	$\uparrow \pi$	5.8
∇`X`$_3$	`(`	$\downarrow$`)X`$_i$	5.7
`[`	`[(`	$\downarrow$`]`	5.4
`]`	π`]`	$\uparrow \pi$`]`	5.5

Fonte: adaptado de José Neto (1993).

Execução não determinística

O transdutor trata os não determinismos inerentes à linguagem natural relacionados à ambiguidade léxica e sintática (Item 2.2.1), e também daquele resultante do processo de conversão da gramática livre de contexto para o reconhecedor equivalente (Item 5.2.2), realizando a análise das diversas alternativas possíveis com o uso do ambiente de execução (*multi-thread*) proporcionado pelo Java. Ou seja, o modelo de execução atribui uma *thread* para cada trajetória de execução possível do autômato de pilha estruturado no decorrer da análise da cadeia de entrada.

A análise da cadeia de entrada é iniciada somente com uma única *thread* de execução, porém quando ocorre uma situação em que há mais de uma possibilidade de transição a partir de um estado do autômato, uma nova *thread* é criada.

Em cada passo de execução, para cada elemento da lista de *tokens* fornecido pelo analisador léxico-morfológico é verificado se há alguma transição possível, de forma similar à operação de *look ahead* de 1 símbolo em analisadores sintáticos usados em compiladores (AHO et al., 2007). Desta forma, é montada uma lista com o conjunto de possíveis transições, ou seja, considerando que existam n possíveis categorias gramaticais para o símbolo corrente da cadeia de entrada e m transições originadas no estado corrente, são analisadas $n \times m$ casos para se obter a lista de transições a serem percorridas. Em suma, a criação de novas *threads* ocorre nas seguintes situações:

1. Um único símbolo da cadeia de entrada foi classificada em mais de uma classe gramatical pelo analisador léxico-morfológico, gerando uma lista de *tokens*, e existem transições para mais de um *token*.

2. Para um único *token* há mais uma transição possível.

Este modelo de execução requer estruturas de dados que podem ser agrupadas em três tipos de acordo com necessidade de compartilhamento entre *threads*:

1. **Compartilhado apenas para leitura** – São as estruturas de dados imutáveis: a descrição do próprio transdutor (estados, transições e rotinas semânticas) e a cadeia de entrada.

2. **Compartilhado para escrita** – É a estrutura de lista indexada por um identificador da *thread* que contém os resultados de análise de cada instância de execução. O acesso a esta lista é serializado para evitar inconsistência.

3. **Não compartilhado** – São estruturas de dados que contêm a situação corrente e a memória de execução de cada *thread*. São as estruturas do analisador léxico, o estado corrente de execução do transdutor (submáquina e estado), a pilha de controle do autômato, a pilha do transdutor, e a cadeia de saída.

A criação de *thread* pode ocorrer em qualquer estado. Tanto aquela que vinha sendo executada quanto a nova devem prosseguir com o processamento acessando todas as informações de estado armazenadas até o momento (item 3 acima) que poderão ser alteradas distintamente por cada uma delas. Portanto, os dados não compartilhados devem ser copiados na criação da nova *thread*, antes de iniciar a sua execução. Os dados do estado corrente do analisador léxico-morfológico também precisam ser replicados pois ele suporta a operação de retorno de símbolo pelo sintático. Na implementação realizada, a *thread* original prossegue com a execução da primeira alternativa da lista de transições a serem efetuadas, e antes disso, inicia a criação de uma nova *thread* para cada uma das alternativas restantes. Para tornar determinístico o primeiro passo de execução na nova *thread*, na sua criação é passada uma estrutura de dados contendo a classe gramatical específica do *token* a ser considerada e o estado destino para tornar única a transição seguinte.

5.4 ANÁLISE DE DEPENDÊNCIA DE CONTEXTO

O objetivo da análise de dependência é obter para cada palavra da sentença de entrada uma estrutura de dados associada com informações sobre as suas relações de dependência. Na implementação realizada, a representação destes dados foi baseada no formato utilizado nas anotações da versão 2.0 do projeto *Universal Dependencies* para o esforço compartilhado de elaboração de *treebanks* (NIVRE et al., 2017; MUNIZ; CHALUB; RADEMAKER, 2017) de diversas línguas naturais no evento *Conference on Computational Natural Language Learning* (CoNLL 2017).

Além das informações adicionadas pelo analisador morfológico na formação de *tokens* a partir da cadeia de entrada, são acrescentados dados descritos na Tabela 5.9 à cada palavra da sentença em análise. Desta forma, as folhas da árvore sintática estrutural serão compostas por esta estrutura de dados.

Tabela 5.9: Dados de relações de dependência associados às palavras da sentença de entrada.

Identificador	Descrição
`id`	Número sequencial da palavra dentro da sentença, iniciando em 1. No caso de expressão multipalavra, pode ser uma faixa de números.
`POS`	Classe gramatical (Tabela 6.2).
`feats`	Atributos secundários da palavra.
`head`	Número do elemento do qual é dependente, ou zero se for elemento raiz.
`depRel`	Tipo de relação de dependência (Tabela 6.5) que possui com o elemento do qual depende.

Fonte: autor.

A análise das relações de dependência de contexto, conforme apresentado no item 4.5 é realizada durante o processo de reconhecimento da estrutura livre de contexto, usando

padrões de dependência previamente levantados. Estes padrões são agrupados em listas associadas a cada regra de produção da gramática original, que são identificadas durante o *parsing* da sentença de entrada.

Cada padrão é armazenado numa estrutura de dados `DepPattern` com as informações da Tabela 5.10.

Tabela 5.10: Estrutura de dados de um padrão de dependência `DepPattern`.

Identificador	Descrição
`value`	Identificador: nome do não terminal ou do terminal.
`main`	Número do elemento que é o núcleo do constituinte.
`headDirection`	Sentido (nulo, esquerda ou direita) em que se encontra o elemento do qual o elemento é dependente.
`constituentList`	Lista dos elementos constituintes (Figura 5.11)

Fonte: autor.

As informações de dependência são armazenadas na lista `constituentList` composta por estruturas de dados que descrevem cada elemento constituinte, cujo conteúdo está descrita na Tabela 5.11.

Tabela 5.11: Estrutura de dados de um elemento da lista de constituintes.

Identificador	Descrição
`id`	Número sequencial iniciando em 1 do elemento corrente na lista `constituentList`.
`value`	Identificador: nome do não terminal ou do terminal.
`type`	Tipo: terminal ou não terminal.
`head`	Número do elemento do qual é dependente, no caso de uma relação com um elemento interno da lista, ou zero se for elemento externo.
`depRel`	Tipo de relação de dependência (Tabela 6.5) que possui com o elemento do qual depende.
`leftDepList`	Lista de tipo de relações de dependência externas à esquerda. (Tabela 6.5).
`rightDepList`	Lista de tipo de relações de dependência externas à direita. (Tabela 6.5).

Fonte: autor.

Por exemplo, o padrão (c) da Figura 4.14 é do tipo "não terminal", com o valor `PadVerb` e teria os elementos descritos na Tabela 5.12 na lista `constituentList`.

Considerando um caso em que todos os constituintes de um não terminal são terminais, como nos padrões exemplos (b) e (c) da Figura 4.15 referentes a um sintagma substantivo `SS`, o padrão encontrado é usado para preencher a estrutura de dados de cada elemento (Tabela 5.9).

Se o não terminal em análise não estiver no topo da árvore sintática, algumas informações estarão disponíveis somente após a análise do nó de hierarquia superior ao

Tabela 5.12: Exemplos de estruturas de dados de elementos da lista de constituintes.

Identificador	Elemento 1	Elemento 2	Elemento 2
`id`	1	2	3
`type`	terminal	não terminal	não terminal
`value`	`vtdi`	`SS`	`SP`
`head`	0	1	1
`depRel`	-	`obj`	`iobj`
`leftDepList`	[`nsubj`]	[ ]	[ ]
`rightDepList`	[`punct`]	[ ]	[ ]

Fonte: autor.

qual ele está conectado, e ficarão sem preenchimento. Estas informações parciais são então armazenados no mesmo tipo de estrutura de dados usados para a descrição de um padrão (Tabela 5.10, com a ressalva de que os elementos da lista de constituintes `constituentList` possui dois campos adicionais além daqueles presentes na estrutura da Tabela 5.11: o primeiro é o `id_sentence` referente ao número do elemento dentro da cadeia de entrada; o segundo é o `word` que corresponde à palavra da sentença.

A identificação da regra de produção, assim como de seus elementos constituintes são obtidas na execução da ação semântica de processamento de rótulos (Item 5.3.2). Este conjunto de informações são justamente as saídas das regras de mapeamento de rótulos descritas na Tabela 5.8 e da ação semântica associada à transição em que um símbolo terminal, a palavra em linguagem natural, é consumida. Estes dados consequentemente constituem a cadeia de entrada do processo de análise das relações de dependências.

O algoritmo de levantamento dos padrões de dependência presentes na sentença utiliza uma pilha auxiliar cujo elemento é uma estrutura de dados contendo informações obtidas da saída do transdutor livre de contexto. A Tabela 5.13 apresenta a relação entre os símbolos de saída do transdutor (primeira coluna) e as movimentações a serem realizadas na pilha auxiliar (terceira coluna). A segunda coluna informa a partir de qual rotina semântica é obtida a saída do transdutor, e a última contém uma breve descrição sobre a operação na pilha. Os símbolos $\uparrow$ e $\downarrow$ são usados para indicar as operações de empilhamento e desempilhamento de um elemento da pilha. O símbolo Π representa o conteúdo da pilha entre o topo e a primeira ocorrência do delimitador "`]`" não incluindo este último.

O algoritmo é executado nas ações semânticas do reconhecedor livre de contexto gerado (item 5.3.2), que são responsáveis pela obtenção da árvores estrutural da sentença de entrada. A descrição de sua implementação foi dividida em algoritmos constituintes menores para fins de clareza e estão apresentadas a seguir. Nestas descrições, os trechos grafados na cor azul são específicos do tratamento de padrões de dependências.

Além da pilha `transducerStack` utilizada na obtenção de árvore estrutural livre de contexto (item 5.3.2), a pilha `depStack`, cujo comportamento está descrito na Tabela

Tabela 5.13: Mapeamento da saída do transdutor livre de contexto para identificação de padrão de relações de dependências.

Saída (LC)	Origem da Saída (LC)	Pilha	Observação
`()`	rótulos	$\downarrow$ `data`$_{\epsilon}$	`data`$_{\epsilon}$: dados de uma cadeia vazia presente na gramática.
‘‘`word`’’	transição	$\downarrow$ `data`$_{\text{“word”}}$	`data`$_{\text{“word”}}$: dados de uma palavra da sentença analisada.
`(`σ`)`	rótulos	$\downarrow$ `data`$_{\sigma}$	`data`$_{\sigma}$: dados de um terminal.
`X`$_i$`)`	rótulos	$\uparrow \Pi$, $\downarrow$`data`$_{\texttt{X}_i}$	`data`$_{\texttt{X}_i}$: dados do padrão de dependência da regra `X`$_i$ obtidas a partir dos elementos (Π) retirados da pilha.
`[(`	rótulos	$\downarrow$`]`	`]`: Delimitador de elementos de uma regra de produção.
`]`	rótulos	$\uparrow$`]`	`]`: Delimitador de elementos de uma regra de produção.

Fonte: autor.

5.13, é usada como variável global. Os elementos desta pilha são estruturas de dados que contém o tipo de elemento (`type`) e informações adicionais (`data`) conforme este tipo, que incluem uma outra estrutura de dados correspondente a um nó `node` de um padrão de dependência.

O Algoritmo 5.1 é executado na ação de tratamento das transições em que há consumo de símbolos terminais e que obtém as informações associadas (a palavra em análise e seus atributos) a estes terminais.

Algoritmo 5.1 Ação semântica de tratamento de transição com símbolo terminal

```
procedure ActionTermTransition(token)
    print token.value
    depStackElement.type ← "word"
    depStackElement.data ← (token.value, token.attributes)
    depStack.Push(depStackElement)
end procedure
```

Os Algoritmos 5.2 a 5.9 fazem parte da ação semântica que processa as listas de rótulos (item 5.2.2) que estejam associadas a qualquer tipo de transição.

Algoritmo 5.2 Ação semântica de tratamento do rótulo ϵ

```
procedure ActionLabelEpsilon(label)
    print "(ϵ)"
    depStackElement.type ← "empty"
    depStack.Push(depStackElement)
end procedure
```

Algoritmo 5.3 Ação semântica de tratamento de rótulo do tipo terminal

```
procedure ActionLabelTerm(label)
    print "(label)"
    word ← depStack.Pop()
    node.type ← "term"
    node.data ← informações de word (posição na sentença, atributos)
    depStackElement.type ← "term"
    depStackElement.data ← (node)
    depStack.Push(depStackElement)
end procedure
```

Algoritmo 5.4 Ação semântica de tratamento do rótulo "[”

```
procedure ActionLabelLeftBracket(label)
    print "[(";
    transducerStack.Push("]")
    depStackElement.type ← "["
    depStack.Push(depStackElement)
end procedure
```

Algoritmo 5.5 Ação semântica de tratamento do rótulo "]”

```
procedure ActionLabelRightBracket(label)
    while trasducerStack is not empty do
        if transducerStack.Top() ≠ "]" then
            stackElement ← transducerStack.Pop()
            outputList.Append(stackElement)
            poppedTransducerStackElements.Push(stackElement)
        else
            transducerStack.Pop()
            break
        end if
    end while
    outputList.Reverse()
    outputList.Append("]")
    print outputList
    for each element in poppedTransducerStackElements do
        if element = não terminal then
            depParseNtermStackElement(depStack,element)
        end if
    end for
end procedure
```

Algoritmo 5.6 Ação semântica de tratamento de rótulo do tipo não terminal com recursão à esquerda na regra de produção

```
procedure ActionLabelNTermLeftRecursion(label)
    print "[("
    transducerStack.Push(label)
    transducerStack.Push("]")
    depStackElement.type ←"]"
    depStack.Push()
end procedure
```

Algoritmo 5.7 Ação semântica de tratamento de rótulo do tipo não terminal com recursão à direita na regra de produção

```
procedure ActionLabelNTermRightRecursion(label)
    print "[("
    depParseNtermStackElement(depStack, label)
end procedure
```

Algoritmo 5.8 Ação semântica de tratamento de rótulo do tipo não terminal sem recursão na regra de produção

```
procedure ActionLabelNTermWithoutRecursion(label)
    while trasducerStack is not empty do
        if transducerStack.Top() ≠ "]" then
            stackElement ← transducerStack.Pop()
            outputList.Append(stackElement)
            poppedTransducerStackElements.Push(stackElement)
        else
            break
        end if
    end while
    outputList.Reverse()
    outputList.Append(label)
    outputList.Append(")")
    print outputList
    for each element in poppedTransducerStackElements do
        if element = não terminal then
            depParseNtermStackElement(depStack, label)
        end if
    end for
    depParseNtermStackElement(depStack, label)
end procedure
```

Os Algoritmos 5.5, 5.7 e 5.8 compartilham o tratamento de não terminal descrito no Algoritmo 5.9.

Algoritmo 5.9 Tratamento de padrão de dependência associado a não terminal

```
procedure DEPPARSENTERMSTACKELEMENT(depStack, label)
    while depStack is not empty do
        if depStack.Top() ≠ "]" then
            depStackElement ← depStack.Pop()
            poppedDepStackElements.Push(depStackElement)
        else
            depStack.Pop()
            break
        end if
    end while
    depPatternsList ← resultado da busca de padrões de dependências que correspondam ao não terminal label e aos elementos constituintes em poppedDepStackElements
    node.type ← "nterm"
    node.data ← depPatternsList
    for each element in poppedDepStackElements do
        if element = não terminal then
            node.addChild(element)
        end if
    end for
    newDepStackElement.type ← "nterm"
    newDepStackElement.data ← node
    depStack.Push(newDepStackElement)
end procedure
```

Ao final do reconhecimento da sentença é obtida a sua árvore sintática livre de contexto no qual cada nó correspondente a um não terminal terá na sua estrutura de dados uma lista de possíveis padrões de dependência. A pilha `depStack` conterá o elemento que corresponde ao não terminal raiz da gramática.

A árvore finalizada será percorrida a partir deste nó raiz de forma recursiva na sequência *preorder* (AHO; HOPCROFT; ULLMAN, 1983). Em cada nó, ao finalizar a análise dos nós descendentes sequencialmente da esquerda para direita, serão preenchidas as relações de dependência da estrutura de dados associado.

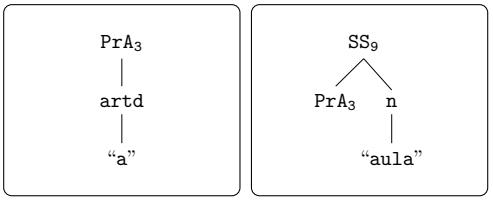
A Figura 5.18 apresenta um exemplo de identificação dos padrões de relações de dependências de duas regras de produção a partir de uma saída do transdutor livre de contexto.

A partir da identificação do padrão existente no nó da árvore sintática procede-se à busca do conjunto de possíveis padrões armazenados correspondentes. Com as informações do padrão, durante o *parsing* livre de contexto são preenchidas as eventuais relações de dependência que possam existir entre os nós filhos. As relações dos elementos descendentes que podem ser preenchidos com as informações disponíveis são também pro-

Figura 5.18: Exemplo de identificação de padrões de relações de dependência.

Saída (LC): `[( [( ‘‘a’’ (artd) PrA3) ] ‘‘aula’’ (n) SS9) ]`

Padrões a identificar:



Análise da saída:

Saída (LC)	Pilha	Observação
`[(`	`]`	
`[(`	`],]`	
`‘‘a’’`	`],],data`$_{\text{“a”}}$	
`(artd)`	`],],data`$_{\text{“a”}}$`,data`$_{artd}$	
`PrA3)`	`],data`$_{\text{PrA3}}$`,]`	Identificação de `PrA3`
`]`	`],data`$_{\text{PrA3}}$	
`‘‘aula’’`	`],data`$_{\text{PrA3}}$`,data`$_{\text{“aula”}}$	
`(n)`	`],data`$_{\text{PrA3}}$`,data`$_{\text{“aula”}}$`,data`$_{artd}$	
`SS9)`	`data`$_{\text{SS9}}$`,]`	Identificação de `SS9`
`]`	`data`$_{\text{SS9}}$	

Fonte: autor.

cessados, considerando o nó em análise como sendo o menor ancestral comum. Ao final do processamento cada terminal deverá ter na estrutura de dados que o descreve qual é o elemento da qual ele é dependente e o tipo de relação existente, exceto a palavra que é a raiz.

5.5 SUMÁRIO

Esta seção apresentou aspectos de implementação dos módulos que compõe o sistema, incluindo detalhes relativos àqueles utilizados em etapas de montagem de estruturas e autômatos e outros relacionados à efetiva operação de análise de uma sentença.

CAPÍTULO 6

ENSAIOS

6.1 INTRODUÇÃO

Neste capítulo são apresentados os experimentos realizados para validar os dispositivos projetados para atender às propostas apresentadas nesta tese.

Serão caracterizados os dados de entrada e os resultados obtidos após o processamento.

6.2 CARACTERIZAÇÃO DOS ENSAIOS

A seguir são caracterizados os dados que foram usados nas análises léxico-morfológica, livre de contexto e de relações de dependência.

6.2.1 ANÁLISE LÉXICO-MORFOLÓGICA

Construção

O analisador léxico foi construído a partir de dados disponíveis publicamente em corpus listadas a seguir.

O primeiro é o subconjunto em português brasileiro de um corpus com sentenças[1] etiquetadas por máquina e corrigidas manualmente (FREITAS; ROCHA; BICK, 2008). O motivo da escolha é que as mesmas sentenças também foram anotadas com informações de dependência no projeto *Universal Dependencies*[2] usadas em etapa posterior.

A quantidade de palavras distintas no corpus é de 14.751 de um total de 80.394 presentes em 4216 sentenças.

A Tabela 6.1 apresenta as classes gramaticais[3] presentes no corpus Bosque e as quantidades de cada uma delas.

A partir de um segundo corpus (AIRES, 2000)[4] etiquetadas por máquina e corrigida manualmente foram obtidas 18.427 palavras distintas de um total de 104.966 palavras, classificadas nas classes gramaticais listadas na Tabela 6.2. Observa-se que a classificação é mais detalhada que no Bosque. Considerando esta característica, nos experimentos as

[1]Disponível em <http://corpora.di.uminho.pt/linguateca/FS/Bosque_CF_8.0.PennTreebank.ptb.gz> Acesso em: 2 de janeiro de 2019.

[2]Disponível em <https://github.com/UniversalDependencies/UD_Portuguese-Bosque> Acesso em: 2 de janeiro de 2019.

[3]Disponível em <https://www.linguateca.pt/Floresta/BibliaFlorestal/completa.html> Acesso em: 2 de janeiro de 2019.

[4]Disponível em <http://www.nilc.icmc.usp.br/nilc/download/corpus100.txt>. Acesso em: 2 de janeiro de 2019.

Tabela 6.1: Classes gramaticais do corpus Bosque.

Símbolo	Classe gramatical	Qtd.
n	nome, substantivo	40728
prop	nomes próprios	11977
n-adj	flutuação entre substantivos adjetivos	633
adj	adjetivos	10424
adv	advérbios	9251
v-.*	verbos	26537
v-fin	finitos	15877
v-ger	gerúndios	863
v-pcp	particípios	4730
v-inf	infinitivos	5067
art	artigos	29793
pron-.*	pronomes	11147
pron-det	determinativos	5054
pron-indp	independentes	3309
pron-pers	pessoais	2784
prp	preposições	33048
intj	interjeições	39
conj-.*	conjunções	7506
conj-s	subordinativa	2328
conj-c	coordenativa	5178
prefixos	elemento composto	179
total		190513

Fonte: adaptado de Freitas, Rocha e Bick (2008).

classes foram normalizadas seguindo as abreviações usadas neste corpus. Nos atributos secundários foram usadas as abreviações do corpus Bosque.

O terceiro corpus utilizado para o flexionador de verbos, substantivos e adjetivos é o TeP 2.0 (MAZIERO et al., 2008)[5]. Esta base conta com 3.636 verbos, 3.381 substantivos e 3.535 adjetivos.

6.2.2 ANÁLISE LIVRE DE CONTEXTO

Obtenção do reconhecedor a partir de gramática em Wirth modificada

Nos experimentos foram utilizadas regras livres de contexto do português brasileiro adaptados do trabalho de Luft (2002), e que consideram o subconjunto das sentenças cuja sintaxe estão colocadas na ordem direta. Ou seja, são casos em que o termo regente precede o regido, como nos exemplos: "festa alegre", "o vaso teria quebrado, se não tivesse protegido", "ele queria que viajassem". Na ordem indireta de colocação, o regido (determinante) precede o regente (determinado) resultando nos exemplos: "alegre festa", "se não

[5]Disponível em <http://www.nilc.icmc.usp.br/tep2/> Acesso em: 2 de janeiro de 2018.

Tabela 6.2: Classes gramaticais no corpus de Aires (2000).

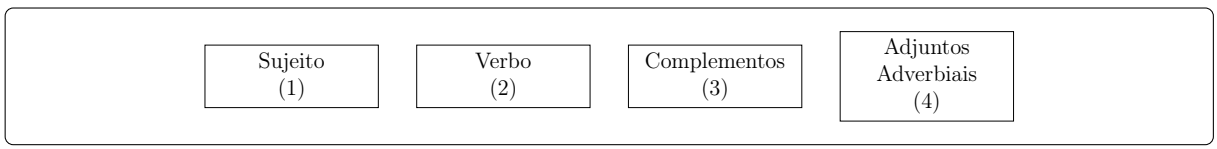
Categoria	Abreviação	Categoria	Abreviação
adjetivo	`adj`	pronome apassivador	`papass`
advérbio	`adv`	pronome de realce	`preal`
artigo	`art`	pronome tratamento	`ptra`
número cardinal	`nc`	preposição	`prep`
número ordinal	`ord`	verbo auxiliar	`vaux`
outros números	`no`	verbo de ligação	`vlig`
substantivo comum	`n`	verbo intransitivo	`vint`
nome próprio	`np`	verbo transitivo direto	`vtd`
conj. coordenativa	`conjcoord`	verbo transitivo indireto	`vti`
conj. subordinativa	`conjsub`	verbo trans. direto e indireto	`vtdi`
pronome demonstrativo	`pd`	interjeição	`i`
pronome indefinido	`pind`	locução adverbial	`ladv`
pronome oblíquo átono	`ppoa`	locução conjuncional	`lconj`
pronome pessoal caso reto	`ppr`	locução prepositiva	`lprep`
pronome possessivo	`pps`	locução pronominal	`lp`
pronome relativo	`pr`	palavra denotativa	`pden`
pronome oblíquo tônico	`ppot`	locução denotativa	`lden`
pronome interrogativo	`pint`	palavras ou símbolos residuais	`res`

Fonte: adaptado de Aires (2000).

tivesse protegido, o vaso teria quebrado", "que viajassem, queria ele".

A ordem direta implica na distribuição dos constituintes imediatos da oração conforme descrito na Figura 6.1, assim como a ordem dos elementos internos destes. Exceto o verbo, os demais itens não são obrigatórios. Se houver mais de um complemento (item 3), a sequência na ordem direta será primeiro o objeto direto, em seguida o objeto indireto ou predicativo do objeto.

Figura 6.1: Posições básicas da oração.



Fonte: adaptado de Luft (2002).

Dentre os constituintes, a Tabela 6.3 descreve os sintagmas e as suas composições.

Os ensaios consideraram inicialmente a análise de períodos simples na ordem direta dos padrões oracionais propostos por Luft (2002), a saber:

1. Padrão impessoal (`PadImp`) – São orações sem sujeito, que tem o verbo como núcleo do predicado. Exemplos: "Está quente", "Chove".

2. Padrão pessoal nominal (`PadNom`) – São orações com sujeito, com predicado nominal, que tem um nome ou pronome (substantivo, adjetivo, advérbio) como núcleo. O

Tabela 6.3: Constituintes de sintagmas.

Sintagma	Símbolo	Ordem dos constituintes
Sintagma Substantivo (núcleo: substantivo)	`SS`	Quantitativos + Pronomes Adjetivos + Sintagma Adjetivo$_1$ (Adjetivos explicativos, subjetivos) + Substantivo + Sintagma Adjetivo$_2$ + Sintagma Preposicional + Oração Adjetiva
Sintagma Verbal (núcleo: verbo)	`V`	Pré-verbais + Verbo Auxiliar + Verbo Principal
Sintagma Adjetivo (núcleo: adjetivo)	`SAdj`	Advérbio de intensidade + Adjetivo + Sintagma Preposicional (complemento nominal)
Sintagma Adverbial (núcleo: adverbio)	`SAdv`	Advérbio de intensidade + Advérbio + Sintagma Preposicional (complemento nominal)
Sintagma Preposicional (núcleo: preposição + SS)	`SP`	Preposição + Sintagma Substantivo

Fonte: adaptado de Luft (2002) e Contier, Padovani e José Neto (2017).

verbo é chamado de verbo de ligação (`vlig`). Exemplos: "O menino é esforçado". "O cachorro está na rua".

3. Padrão pessoal verbal (`PadVerb`) – São orações com sujeito, com predicado verbal, que tem um verbo como núcleo. O verbo pode ser intransitivo (`vint`), sem complemento; transitivo direto (`vtd`), transitivo indireto (`vti`), ou transitivo direto e indireto (`vtdi`) respectivamente com os complementos objeto direto, objeto indireto e ambos. Exemplos: "A mala pesa vinte quilos", "Eu entreguei o livro à bibliotecária".

4. Padrão pessoal verbo-nominal (`PadVerbNom`) – São orações com sujeito, cujo núcleo do predicado é um verbo transitivo predicativo e que tem como complementos um objeto direto e um predicativo do objeto. Exemplos: "O pai considera seu filho um gênio", "Ela acordou gripada".

A Figura 6.2 apresenta a fórmula geral dos padrões oracionais.

Figura 6.2: Fórmula geral para oração `S` na ordem direta.

```
S = (SS|ϵ) V (SS|ϵ) (SS|SAdj|SAdv|SP|ϵ) (SP|ϵ) (SP|ϵ) .
```

Fonte: adaptado de Luft (2002).

A fórmula geral é obtida a partir de regras parciais da gramática adotada para a análise livre de contexto de períodos simples na notação de Wirth modificada descrita na Figura 6.3.

As regras de produção dos sintagmas, que refletem as descrições da Tabela 6.3, estão apresentadas na Figura 6.4 em expressões consolidadas a partir de expressões parciais.

Figura 6.3: Gramática livre de contexto para períodos simples.

```
Sentence = PadImp ‘‘punct’’ .          PadVerb = ‘‘vint’’ .
Sentence = SS PadNom ‘‘punct’’ .       PadVerb = ‘‘vtd’’ SS .
Sentence = SS PadVerb ‘‘punct’’ .      PadVerb = ‘‘vti’’ SP .
Sentence = SS PadVerbNom ‘‘punct’’ .   PadVerb = ‘‘vti’’ SP SP.
PadImp = ‘‘vlig’’ SS .                 PadVerb = ‘‘vti’’ SAdv .
PadImp = ‘‘vlig’’ SAdj .               PadVerb = ‘‘vtdi’’ SS SP .
PadImp = ‘‘vlig’’ SAdv .               PadVerb = ‘‘vtdi’’ SS SP SP .
PadImp = ‘‘vlig’’ SP .                 PadVerb = ‘‘vtdi’’ SS SAdv .
PadImp = ‘‘vint’’ .                    PadVerbNom = ‘‘vtpred’’ SS SS .
PadImp = ‘‘vtd’’ SS .                  PadVerbNom = ‘‘vtpred’’ SS SAdj .
PadImp = ‘‘vti’’ SP .                  PadVerbNom = ‘‘vtpred’’ SS SP .
PadNom = ‘‘vlig’’ SS .                 PadVerbNom = ‘‘vtpred’’ SS SAdv .
PadNom = ‘‘vlig’’ SAdj .               PadVerbNom = ‘‘vtpred’’ SS .
PadNom = ‘‘vlig’’ SAdv .               PadVerbNom = ‘‘vtpred’’ SAdj .
PadNom = ‘‘vlig’’ SP .                 PadVerbNom = ‘‘vtpred’’ SP .
```

Fonte: adaptado de Luft (2002) e José Neto (2015).

Figura 6.4: Gramática livre de contexto de sintagmas básicos.

Regras de produção de sintagmas:

```
SS = ( "art" | ϵ ) ( "nc" | "ord" | ϵ) ( "pps" | "lp" | "pd" | ϵ )
     ( SAdj | ϵ ) ( "n" | "np" | "ppr" | ϵ ) ( SAdj | ϵ ) ( SP |  ϵ ) .
SAdj = "adj" "adj" SP
SAdv = "adv" "adv" SP
SP = "prep" SS .
```

Fonte: adaptado de Luft (2002).

A Figura 6.5 apresenta uma gramática que considera períodos compostos e que inclui a descrição dos sintagmas constituintes. Os não-terminais estão descritos na Tabela 6.4.

6.2.3 ANÁLISE DE DEPENDÊNCIAS

A Tabela 6.5 apresenta as relações de dependência universais para o português levantadas por Rademaker et al. (2017) nas quais são baseadas as anotações de dependência utilizadas nos ensaios. Em cada linha consta um tipo de dependência, com a primeira coluna apresentando o símbolo usado nas anotações, o tipo e a indicação da figura no Apêndice A.2 que contém a sua descrição com exemplo. A lista segue a ordem alfabética dos símbolos da primeira coluna.

Estes tipos de relações foram aplicadas na construção dos padrões utilizados na implementação do analisador de dependências armazenados em estruturas de dados descritos nas Tabelas 5.10 e 5.11. A Figura 6.6 apresenta como exemplo a regra de produção da gramática do constituinte não terminal `PadNom_2` e padrão de dependência correspondente num formato de árvore para facilitar a visualização.

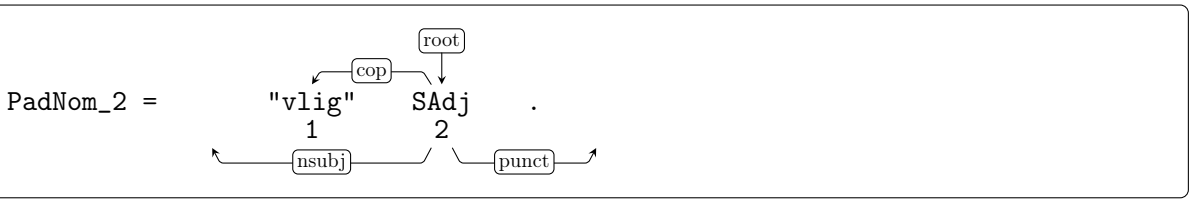
No Apêndice A.3 as Figuras A.26 a A.27 apresentam regras de produção levantadas e os padrões de dependência correspondentes.

Figura 6.5: Gramática completa de Luft (2002) na notação de Wirth modificada.

```
F = PCC .                       PCC = OC ‘‘conjcoord’’ OC ‘‘punct’’ .    SS = ‘‘nc’’ ‘‘n’’ .
F = PCS .                       PCC = OC ‘‘conjcoord’’ PCC ‘‘punct’’ .   SS = ‘‘nc’’ ‘‘n’’ SAdj .
F = PS .                        PCS = OP .                               SS = ‘‘ppr’’ .
GV = ‘‘vtd’’ SS .               PS = OA .                                SS = ‘‘ppoa’’ .
GV = ‘‘vti’’ SP .               PrA = ‘‘artd’’ .                         SS = ‘‘pr’’ .
Nuc = SS SV .                   PrA = ‘‘arti’’ .                         SS = ‘‘n’’ .
O = SN .                        PrA = ‘‘pd’’ .                           SS = ‘‘np’’ .
O = SS SN .                     PrA = ‘‘pind’’ ‘‘artd’’ .                 SS = OS .
O = SS SV .                     PrA = ‘‘pind’’ .                         SS = PrA ‘‘n’’ .
O = SV .                        PrA = ‘‘pps’’ .                          SS = PrA ‘‘n’’ OSAR .
OA = Nuc SAdv ‘‘punct’’ .       PrA = ‘‘pr’’ .                           SS = PrA ‘‘np’’ .
OA = SN ‘‘punct’’ .             SAdj = ‘‘adj’’ .                         SV = ‘‘vaux’’ ‘‘vint’’ .
OA = SS SN ‘‘punct’’ .          SAdj = ‘‘advint’’ ‘‘adj’’ OS .           SV = ‘‘neg’’ ‘‘vti’’ SP .
OA = SS SV ‘‘punct’’ .          SAdv = ‘‘adv’’ .                         SV = ‘‘vint’’ .
OA = SS SV SAdv ‘‘punct’’ .     SAdv = ‘‘advt’’ .                        SV = ‘‘vint’’ SAdv .
OA = SS SVN ‘‘punct’’ .         SAdv = ‘‘advint’’ OS .                   SV = ‘‘vtd’’ SS .
OA = SV ‘‘punct’’ .             SAdv = ‘‘pr’’ .                          SV = ‘‘vtdi’’ SS SAdv .
OC = SS SV .                    SAdv = OS .                              SV = ‘‘vtdi’’ SS SP .
OP = SAdv Nuc ‘‘punct".         SAdv = SP .                              SV = ‘‘vtdi’’ SS SP SP .
OP = SN ‘‘punct".               SN = ‘‘vaux’’ ‘‘Inf’’ SP .               SV = ‘‘vti’’ SAdv .
OP = SS SN ‘‘punct".            SN = ‘‘vaux’’ ‘‘part’’ SP .              SV = ‘‘vti’’ SP .
OP = SS SV ‘‘punct".            SN = ‘‘vlig’’ SAdj .                     SV = ‘‘vti’’ SP SP .
OP = SV ‘‘punct".               SN = ‘‘vlig’’ SAdv .                     SV = GV SAdv .
OS = ‘‘conjsub’’ O.             SN = ‘‘vlig’’ SP .                       SV = SS ‘‘vtd’’ .
OS = SAdv GV .                  SN = ‘‘vlig’’ SS .                       SVN = ‘‘vtpred’’ SAdj .
OSAR = ‘‘vint’’ .               SN = ϵ .                                 SVN = ‘‘vtpred’’ SP .
OSAR = SAdv ‘‘vint’’ .          SP = ‘‘ppr’’ .                           SVN = ‘‘vtpred’’ SS .
OSAR = SAdv SS ‘‘vint’’ .       SP = ‘‘ppoa’’ .                          SVN = ‘‘vtpred’’ SS SAdj .
OSAR = SP SS SP ‘‘vti’’ .       SP = ‘‘prep’’ SS .                       SVN = ‘‘vtpred’’ SS SAdv .
OSAR = SS SS ‘‘vtd’’ .                                                   SVN = ‘‘vtpred’’ SS SP .
OSAR = SS SV .                                                           SVN = ‘‘vtpred’’ SS SS .
```

Fonte: adaptado de Luft (2002).

Figura 6.6: Regra de produção e padrão de relações de dependência de `PadNom_2`



Fonte: adaptado de Luft (2002) e José Neto (2015).

Tabela 6.4: Não terminais da gramática.

Não terminal	Descrição
F	Frase
GV	Grupo verbal - verbo e complemento(s)
Nuc	Núcleo
O	Oração
OA	Oração absoluta
OC	Oração coordenada
OP	Oração principal
OS	Oração subordinada
OSAR	Oração subordinada adjetiva restritiva
PCC	Período composto por coordenação
PCS	Período composto por subordinação
PS	Período simples
PrA	Pronome adjetivo – determinantes ou determinativos
SAdj	Sintagma adjetivo
SAdv	Sintagma adverbial
SN	Sintagma nominal – predicado nominal: verbo de ligação + predicativo
SP	Sintagma preposicional
SS	Sintagma substantivo
SV	Sintagma verbal
SVN	Sintagma verbo nominal

Fonte: adaptado de Luft (2002).

6.3 RESULTADOS

Execução do transdutor sintático

A gramática para períodos simples na forma direta foi validado submetendo-se à sua entrada sentenças exemplo indicadas por Luft (2002).

A Figura 6.7 apresenta um exemplo de análise de uma sentença de período simples com a saída no formato textual com parênteses e colchetes (JOSÉ NETO, 1993) para a sentença "Leo recebeu um presente de seu pai.".

A Figura 6.8 apresenta um exemplo de período composto por subordinação do tipo substantiva no formato de árvore sintática. Esta representação visual foi gerada pelo programa LaTeX(LAMPORT, 1994) a partir da descrição textual utilizada pelo pacote *forest*[6], que foi obtida por meio da conversão do formato de parênteses e colchetes.

Execução da análise de dependências

A Figura 6.9 apresenta a árvore de dependências da sentença exemplo do item 6.3 com tipos de relação de acordo com aqueles definidos na Tabela 6.5. Na parte inferior são listados os conteúdos das variáveis relevantes para a construção da árvore de dependência.

[6]Disponível em <https://ctan.org/pkg/forest> Acesso em: 2 de janeiro de 2019.

Tabela 6.5: Relações de dependências do português brasileiro.

Símbolo	Descrição	Figura no Apêndice
acl:inf	Oração modificadora de nome - oração adjetiva	A.22
acl:relcl	Oração modificadora de nome - oração adjetiva relativa	A.22
advcl	Oração modificadora adverbial	A.22
advmod	Modificador adverbial	A.19
amod	Modificador adjetivo	A.19
appos	Modificador apositivo	A.18
aux	Auxiliar	A.21
case	Marcação de caso	A.18
cc	Conjunção coordenativa	A.20
ccomp	Complemento oracional	A.22
conj	Associado	A.20
cop	Copula	A.21
csubj	Sujeito oracional	A.22
csubj:pass	Sujeito oracional passivo	A.22
dep	Dependência não classificada	A.16
det	Determinante	A.18
det:poss	Determinante possessivo	A.18
expl:pv	Expletivo - verbo pronomial	A.19
fixed	Fixo	A.23
flat	Homogêneo	A.23
iobj	Objeto indireto	A.17
mark	Marcador	A.21
nmod	Modificador nominal	A.18
nsubj	Sujeito	A.17
nummod	Modificador numérico	A.18
obj	Objeto direto	A.17
obl	Oblíquo nominal	A.19
orphan	Órfão	A.24
parataxis	Parataxe	A.25
punct	Pontuação	A.16
root	Raiz	A.16
vocative	Vocativo	A.19
xcomp	Complemento	A.22

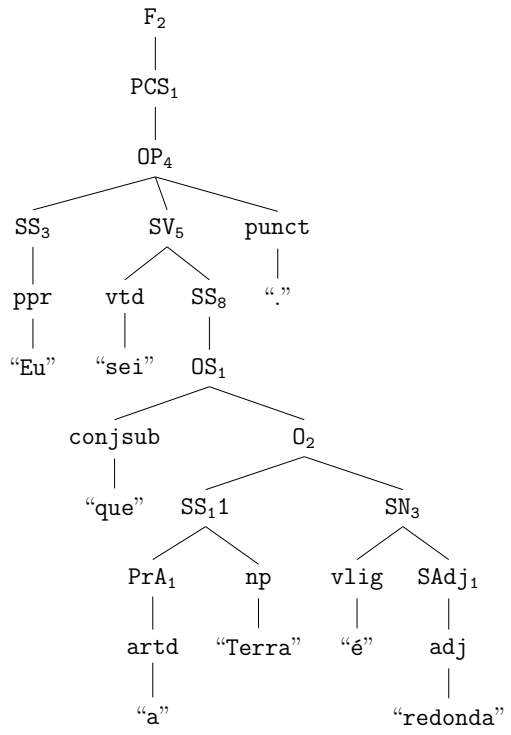
Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura 6.7: Exemplo de árvore sintática de saída de período simples.

```
[
 (
  [( "Leo" (np) SS_1) ]
  [( "recebeu" (vtdi)
   [( "um" (art) "presente" (n) SS_2) ]
   [( "de" (prp)
    [( "seu" (pps) "pai" (n) SS_3) ]
    SP_1)
   ]
   PadVerb_1)
  ]
  "." (punct)
  Sentence_1)
]
```

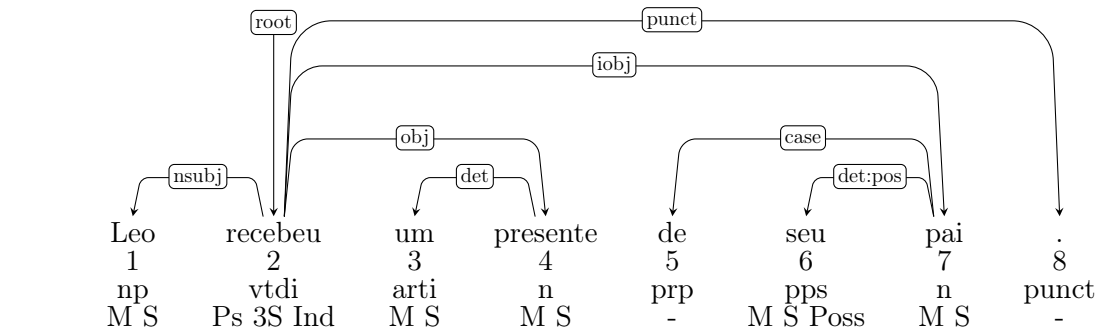
Fonte: autor.

Figura 6.8: Exemplo de árvore sintática de saída de período composto.



Fonte: autor.

Figura 6.9: Exemplo de saída de relações de dependência de uma sentença de período simples.



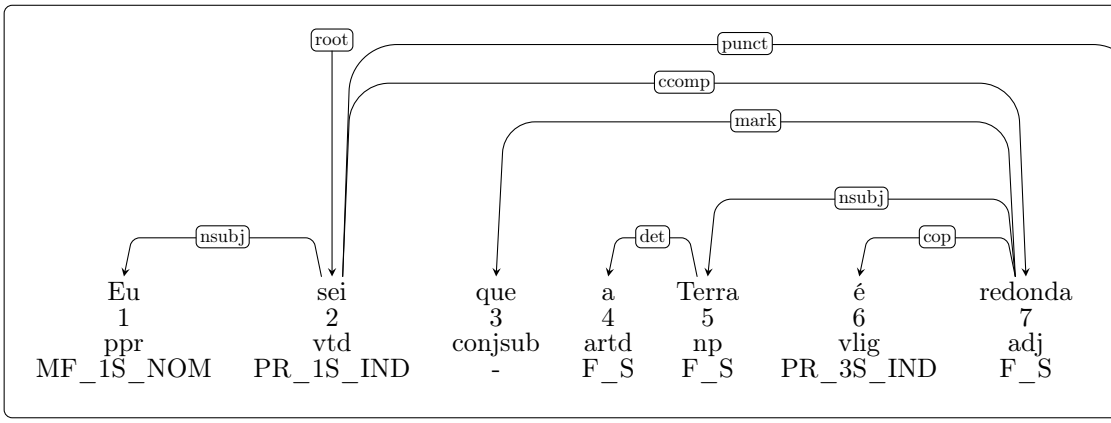

```
ID  value   word         head depRel
1   np      "Leo"        2    nsubj
2   vtdi    "recebeu"    0    root
3   arti    "um"         4    det
4   n       "presente"   2    obj
5   prp     "de"         7    case
6   pps     "seu"        7    det:pos
7   n       "pai"        2    iobj
8   punct   "."          2    punct
```

Fonte: autor.

A Figura 6.10 apresenta a árvore de dependências da sentença composta por subordinação cuja estrutura livre de contexto está descrita na Figura 6.8.

Nas duas Figuras (6.9 e 6.10) os diagramas foram gerados pelo pacote *tikz-dependency*[7] do programa LaTeXa partir da saída textual do *parser*.

Figura 6.10: Exemplo de saída de relações de dependência de uma sentença de período composto.



Fonte: autor.

[7]Disponível em <https://ctan.org/pkg/tikz-dependency> Acesso em: 2 de janeiro de 2019.

No Apêndice A.4 estão listados outros exemplos de execução nas Figuras A.33 a A.35.

6.4 SUMÁRIO

O conjunto de sistemas implementados necessitam ser alimentados com dados relacionados à língua analisada, que neste trabalho é o português brasileiro, incluindo dicionários, corpus e regras gramaticais e de dependência. Nesta seção foram detalhadas os dados utilizados para realizar ensaios dos sistemas implementados e exemplos de resultados de análises de sentenças.

CAPÍTULO 7

CONCLUSÃO

Este capítulo apresenta discussões sobre a técnica de análise de linguagem natural utilizada nesta tese, contribuições obtidas pela pesquisa desenvolvida e indicações de trabalhos futuros para dar continuidade ao assunto.

7.1 DISCUSSÕES

Os ensaios realizados possibilitaram a validação de método proposto para a análise de texto de um subconjunto de língua portuguesa brasileira. As técnicas empregadas são orientadas a regras, especificamente de constituição estrutural das sentenças e das relações de dependências entre as suas palavras. Os ensaios utilizaram regras gramaticais livre de contexto para sentenças na ordem direta obtidos de uma referência específica (LUFT, 2002) e padrões de dependência manualmente elaborados com base em especificações criadas por um grupo de pesquisadores (NIVRE et al., 2017). Por meio da manipulação da especificação destas regras, com operações de inclusão ou exclusão é possível alterar o subconjunto da língua tratado pelo sistema resultante. Essa característica do método permitir a geração de analisadores modificados de forma incremental por meio da alteração das regras gramaticais abre a possibilidade de se obter gradualmente um sistema de processamento mais aderente à língua analisada. Mais ainda, isso pode ser explorado para aproximar o sistema ao comportamento dinâmico da língua natural.

Com relação à especificação de regras de dependência, usando-se o mesmo autômato de pilha gerado automaticamente, com a implementação de algumas rotinas semânticas e a adição de novas estruturas de dados é viável a construção de um sistema para levantar regras de dependência a partir de um corpus anotado. Cada sentença anotada seria submetido ao *parser* livre de contexto, e ao final do processamento de uma subcadeia referente a um não terminal da gramática, as relações de dependência dos elementos constituintes correspondentes poderiam ser extraídos e se o padrão ainda não estiver presente na lista do analisador, ele pode ser armazenado para uso futuro.

Um outro ponto a ressaltar nesta tese é o fato dela ter permitido reunir assuntos desenvolvidos em diversas áreas para a elaboração de uma técnica para análise de sentenças. Foram consolidados aspectos tratados em áreas como compiladores (JOSÉ NETO, 2016, 1993), aplicações de técnicas adaptativas em linguagem natural (PADOVANI; JOSÉ NETO, 2017) análise estrutural da língua portuguesa (MIOTO; SILVA; VASCONCELLOS, 2013; LUFT, 2002), análise de dependências universais (NIVRE et al., 2017), recursos publicamente disponíveis, principalmente corpora, para estudos da língua portuguesa

brasileira (RADEMAKER et al., 2017; MAZIERO et al., 2008; AIRES, 2000), bibliotecas de desenvolvimento e software de código aberto (CEREDA; JOSÉ NETO, 2016; BIRD; KLEIN; LOPER, 2009).

7.2 CONTRIBUIÇÕES

São listados a seguir os resultados significativos obtidos no desenvolvimento desta tese.

1. Apresentação de um método de levantamento de um grafo a partir de uma sentença em linguagem natural na ordem direta, que descreva simultaneamente a estrutura sintática dada por uma gramática livre de contexto e as relações de dependência entre seus elementos constituintes.

2. Técnicas de projeto e aspectos de implementação de geração automatizada de um transdutor que obtém a árvore sintática livre de contexto de um subconjunto de sentenças em português brasileiro a partir da descrição gramatical da linguagem.

3. Um método de divisão hierarquizada dos processos de análise de uma sentença em problemas menores baseado nos constituintes de uma sentença.

4. Um método de análise de sentenças usando métodos determinísticos para obter velocidade no reconhecimento por meio de técnica adaptativa no tratamento de inversões.

5. Técnicas de projeto e aspectos de implementação para o tratamento de não determinismos na análise de sentenças.

6. Identificação de padrões de relações de dependência associados à regras livre de contexto em sentenças na ordem direta.

A pesquisa desenvolvida obteve os resultados significativos secundários enumerados abaixo.

1. Disponibilização de uma ferramenta desenvolvida na linguagem Java para a geração automática de analisadores sintáticos livre de contexto a partir de especificação da gramática da linguagem tratada na notação de Wirth modificada. A ferramenta inclui módulos de reconhecimento da especificação gramatical, geração do autômato de pilha estruturado e geração do transdutor. Ela é resultado da consolidação de diversas propostas esparsas (em linguagem formal, natural, linguística, regras de dependência de contexto aplicáveis a linguagem natural) com o objetivo de obter uma formalização que permita a automatização do processamento de linguagem natural.

2. Disponibilização de um construtor de analisador morfológico para o português brasileiro com os seguintes módulos: construtor de dicionário a partir de corpus anotado no formato CoNLL-U (NIVRE et al., 2017) desenvolvido na linguagem Python; flexionador de um conjunto de classes gramaticais (verbos, substantivos e adjetivos) com regras de flexão regulares baseado em técnicas adaptativas implementado em Java.

3. Versão expandida do analisador acima incorporando variantes irregulares.

4. Disponibilização de um transdutor desenvolvido na linguagem Java que obtém a árvore sintática e relações de dependências entre os elementos constituintes de um subconjunto de sentenças do português brasileiro, que permite a alteração de regras livre de contexto e de dependências.

As ferramentas desenvolvidas nesta tese estão publicamente disponíveis na plataforma GitHub [1] de hospedagem de código fonte.

7.3 TRABALHOS FUTUROS

A pesquisa desenvolvida nesta tese pode ter como desdobramento trabalhos nos assuntos abaixo enumerados.

1. Aprimoramento da descrição da gramática livre de contexto do português brasileiro juntamente com as regras de dependência e aquelas associadas a atributos secundários dos elementos constituintes, para permitir a aceitação de um conjunto maior de sentenças,

2. Utilização de métodos desenvolvidos para o processamento de outras línguas naturais.

3. Utilização de métodos desenvolvidos para o processamento de linguagens de programação.

4. Inferência de regras de dependência existentes em constituintes da sentença a partir de corpus anotado.

5. Otimização do tratamento de não-determinismo visando à diminuição das alternativas a serem analisadas, por exemplo, executando a validação de regras decorrentes de resultados parciais de análise de relações de dependência.

6. Estudo de ambiguidades nos padrões de relações de dependências e desenvolvimento de métodos de desambiguação.

[1] Disponível em <http://github.com/nkmiura> Acesso em: 2 de janeiro de 2019.

7. Avaliações sobre a relevância e abrangência do conjunto de regras.

8. Estudo de regras para subárvores compostas por múltiplos padrões de relações de dependências.

9. Desenvolvimento de módulos de verificação da aderência dos atributos secundários das palavras às regras de dependência e a sua incorporação no *parser*.

REFERÊNCIAS BIBLIOGRÁFICAS

AGERRI, R. et al. Big data for natural language processing: a streaming approach. *Knowledge-Based Systems*, v. 79, p. 36 – 42, 2015. ISSN 0950-7051. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950705114003992>. Acesso em: 2 de janeiro de 2019.

AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. *Data Structures and Algorithms*. 1st. ed. Massachusetts, EUA: Addison-Wesley, 1983. ISBN 0201000237.

AHO, A. V. et al. *Compiler - Principles, Techniques & Tools*. 2nd. ed. [S.l.]: Pearson Education, Inc., 2007. ISBN 978-0321486813.

AIRES, R. V. X. *Implementação, adaptação, combinação e avaliação de etiquetadores para o português do Brasil*. Dissertação (Mestrado) — Universidade de São Paulo, 2000. Disponível em: <http://www.nilc.icmc.usp.br/nilc/download/dissertacao2000rachel.zip>. Acesso em: 2 de janeiro de 2019.

ALMEIDA, N. M. de. *Gramática Metódica da Língua Portuguesa*. São Paulo: Saraiva, 2005.

BECHARA, E. *Moderna gramática portuguesa*. 37. ed. Rio de Janeiro: Nova Fronteira, 2009. ISBN 978-85-209-3049-6.

BICK, E. *The parsing system "Palavras": automatic grammatical analysis of Portuguese in a constraint grammar framework*. Tese (Doutorado) — Aarhus Universitetsforlag, Aarhus, Dinamarca, 2000.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Processing with Python: analyzing Text with the Natural Language Toolkit*. Sebastopol, EUA: O'Reilly Media, 2009. ISBN 978-0-596-51649-9. Disponível em: <http://www.nltk.org/book>. Acesso em: 2 de janeiro de 2019.

CAMBRIA, E.; WHITE, B. Jumping NLP curves: a review of natural language processing research [review article]. *IEEE Computational Intelligence Magazine*, v. 9, n. 2, p. 48–57, May 2014. ISSN 1556-603X.

CEREDA, P. R. M. *Macros como mecanismos de abstração em transformações textuais*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 2018.

CEREDA, P. R. M.; JOSÉ NETO, J. AA4J: uma biblioteca para implementação de autômatos adaptativos. In: WORKSHOP DE TECNOLOGIA ADAPTATIVA, 10, 2016, **Anais...**. São Paulo: EPUSP, 2016. p. 16–26. ISBN 978-85-86686-86-3.

CONTIER, A.; PADOVANI, D.; JOSÉ NETO, J. O reconhecedor gramatical adaptativo linguístico, experimentos e resultados comparativos. In: WORKSHOP DE TECNOLOGIA ADAPTATIVA, 11, 2017, **Anais...**. São Paulo: EPUSP, 2017. p. 51–62. ISBN 978-85-86686-90-0.

COOPER, K.; TORCZON, L. *Construindo Compiladores*. 2. ed. Rio de Janeiro: Elsevier, 2014.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introduction to Algorithms*. 3rd. ed. Massachusetts, EUA: The MIT Press, 1990. ISBN 0262033844, 9780262033848.

FRASER, N. Prolegomena to a formal theory of dependency grammar. *UCL Working Papers in Linguistics*, p. 298–319, 1990.

FREITAS, C.; ROCHA, P.; BICK, E. Floresta sintá(c)tica: Bigger, thicker and easier. In: TEIXEIRA, A. et al. (Ed.). *Computational Processing of the Portuguese Language*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 216–219. ISBN 978-3-540-85980-2.

FURUI, S. et al. Fundamental technologies in modern speech recognition [from the guest editors]. *Signal Processing Magazine, IEEE*, v. 29, n. 6, p. 16–17, 2012.

GAIFMAN, H. Dependency systems and phrase-structure systems. *Information and Control*, v. 8, n. 3, p. 304 – 337, 1965. ISSN 0019-9958. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0019995865902329>. Acesso em: 2 de janeiro de 2019.

GANSNER, E.; KOUTSOFIOS, E.; NORTH, S. *Drawing graphs with dot*. 2015. Disponível em: <http://graphviz.org/pdf/dotguide.pdf>. Acesso em: 2 de janeiro de 2019.

GANSNER, E. R.; NORTH, S. C. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, v. 30, n. 11, p. 1203–1233, 2000.

HEINZ, J.; HIGUERA, C. de la; ZAANEN, M. van. *Grammatical Inference for Computational Linguistics*. [S.l.]: Morgan & Claypool Publishers, 2015. (Synthesis Lectures on Human Language Technologies). ISBN 978-1-60845-977-3, 978-1-60845-978-0.

HERINGER, H. J. Dependency Syntax: Formalized models. In: _____. *Handbücher zur Sprach- und Kommunikationswissenschaft (Handbooks of linguistics and communication science)*. Berlin, Alemanha: Walter de Gruyter & Co., 1993. v. 1, p. 316–328. ISBN 3-11-009586-6.

HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, v. 29, n. 6, p. 82–97, 2012.

HOPCROFT, J. E. An $n \log n$ algorithm for minimizing states in a finite automaton. In: INTERNATIONAL SYMPOSIUM ON THE THEORY OF MACHINES AND COMPUTATIONS, 1971, **Proceedings. . . .** Haifa, Israel: Academic Press, 1971. p. 189–196.

JÄGER, G.; ROGERS, J. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, The Royal Society, v. 367, n. 1598, p. 1956–1970, 2012.

JOSÉ NETO, J. *Contribuições à metodologia de construção de compiladores, Tese (Livre-Docência)*. São Paulo: Escola Politécnica da Universidade de São Paulo, 1993.

JOSÉ NETO, J. Um levantamento da evolução da adaptatividade e da tecnologia adaptativa. *Revista IEEE América Latina*, v. 5, n. 7, p. 496–505, Novembro 2007.

JOSÉ NETO, J. *PCS5004 - Slides Aula 5 - Tratamento adaptativo de alguns fenômenos linguísticos*. 2015. Disponível em: <https://sites.google.com/site/2015pcs5004/material-para-download/PCS5004\%202015\%20-\%20AULA\%205\%20-\%20Adaptatividade\%20em\%20Lingu\%C3\%ADstica.pdf?attredirects=0&d=1>. Acesso em: 2 de janeiro de 2019.

JOSÉ NETO, J. *Introdução à compilação*. 2. ed. Rio de Janeiro: Elsevier, 2016. ISBN 978-85-352-7810-1.

JOSÉ NETO, J.; MAGALHÃES, M. E. S. Reconhecedores sintáticos-uma alternativa didática para uso em cursos de engenharia. In: CONGRESSO NACIONAL DE INFORMÁTICA, 14, 1981, **Anais...**. São Paulo: [s.n.], 1981. p. 171–181.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 2. ed. Upper Saddle River, NJ, USA: Prentice Hall, Inc., 2008. ISBN 0131873210.

KÜBLER, S.; MCDONALD, R.; NIVRE, J. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, Morgan & Claypool Publishers, v. 2, n. 1, p. 1–127, 2009.

LAMPORT, L. *LATEX: a document preparation system*. Reading, EUA: Addison-Wesley Pub. Co., 1994. ISBN 978-02-0152-983-8.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. ISSN 0028-0836.

LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. 2nd. ed. Upper Saddle River, EUA: Prentice Hall PTR, 1997. ISBN 0132624788.

LIEBERMAN, E. et al. Quantifying the evolutionary dynamics of language. *Nature*, v. 449, n. 7163, p. 713–716, 2007. ISSN 0028-0836.

LUFT, C. P. *Moderna gramática brasileira: edição revista e atualizada.* São Paulo: Globo Livros, 2002. ISBN 978-85-2503-621-6.

MANNING, C. D. Computational linguistics and deep learning. *Comput. Linguist.*, MIT Press, Cambridge, MA, USA, v. 41, n. 4, p. 701–707, dez. 2015. ISSN 0891-2017. Disponível em: <http://dx.doi.org/10.1162/COLI_a_00239>. Acesso em: 2 de janeiro de 2019.

MANNING, C. D.; SCHÜTZE, H. *Foundations of Statistical Natural Language Processing*. Boston, EUA: MIT Press, 1999.

MARNEFFE, M.-C. D. et al. Universal Stanford dependencies: a cross-linguistic typology. In: CHAIR), N. C. C. et al. (Ed.). INTERNATIONAL CONFERENCE ON LANGUAGE RESOURCES AND EVALUATION, 9, 2014, **Proceedings...**. Reykjavik, Iceland: European Language Resources Association (ELRA), 2014. v. 14, p. 4585–92. ISBN 978-2-9517408-8-4.

MAZIERO, E. G. et al. A base de dados lexical e a interface web do TeP 2.0: thesaurus eletrônico para o português do Brasil. In: *Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2008. (WebMedia '08), p. 390–392. ISBN 978-85-7669-199-0. Disponível em: <http://doi.acm.org/10.1145/1809980.1810076>. Acesso em: 2 de janeiro de 2019.

MEL'ČUK, I. A. *Dependency syntax: theory and practice*. New York: SUNY press, 1988.

MIOTO, C.; SILVA, M. C. F.; VASCONCELLOS, R. E. *Novo Manual de Sintaxe*. 1. ed. São Paulo: Editora Contexto, 2013. ISBN 978-85-7244-800-0.

MUNIZ, H.; CHALUB, F.; RADEMAKER, A. CL-CONLLU: dependências universais em Common Lisp. In: WORKSHOP DE INICIAÇÃO CIENTÍFICA EM TECNOLOGIA DA INFORMAÇÃO E DA LINGUAGEM HUMANA, 5, 2017, **Anais...**. [S.l.: s.n.], 2017.

NIVRE, J. *Dependency grammar and dependency parsing. MSI report*. Växjö, Suécia, 2005. v. 5133, n. 1959, 1–32 p.

NIVRE, J. et al. *Universal Dependencies 2.0*. 2017. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Disponível em: <http://hdl.handle.net/11234/1-1983>. Acesso em: 2 de janeiro de 2019.

NOWAK, M. A.; KOMAROVA, N. L.; NIYOGI, P. Computational and evolutionary aspects of language. *Nature*, v. 417, n. 6889, p. 611–617, 2002.

OTHERO, G. de A. *A gramática da frase em português*. Porto Alegre: EDIPUCRS, 2009. ISBN 978-85-7430-845-8.

PADOVANI, D.; CONTIER, A.; JOSÉ NETO, J. Etiquetamento morfológico usando Tecnologia Adaptativa. In: WORKSHOP DE TECNOLOGIA ADAPTATIVA, 12, 2018, **Anais...**. São Paulo: EPUSP, 2018. p. 17–22. ISBN 978-85-86686-98-6.

PADOVANI, D.; JOSÉ NETO, J. Adaptive automata applied to natural language processing. *Procedia Computer Science*, v. 109, p. 1152 – 1157, 2017. ISSN 1877-0509. INTERNATIONAL CONFERENCE ON AMBIENT SYSTEMS, NETWORKS AND TECHNOLOGIES, 8, 2017, **Proceedings...**. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050917310554>. Acesso em: 2 de janeiro de 2019.

PEREIRA, F. Formal grammar and information theory: together again? *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, The Royal Society, v. 358, n. 1769, p. 1239–1253, 2000. ISSN 1364-503X. Disponível em: <http://rsta.royalsocietypublishing.org/content/358/1769/1239>. Acesso em: 2 de janeiro de 2019.

RADEMAKER, A. et al. Universal dependencies for portuguese. In: INTERNATIONAL CONFERENCE ON DEPENDENCY LINGUISTICS, 4, 2017, **Proceedings...**. Pisa, Itália: Linköping University Electronic Press, 2017. p. 197–206. ISBN 978-91-7685-467-9. Disponível em: <http://aclweb.org/anthology/W17-6523>. Acesso em: 2 de janeiro de 2019.

RAMOS, M.; JOSÉ NETO, J.; VEGA, I. *Linguagens Formais: teoria, modelagem e implementação*. Porto Alegre: Bookman, 2009.

RICCHETTI, P. M.; JOSÉ NETO, J. A practical method for the implementation of syntactic parsers. *WSEAS Transactions on Information Sciences and Applications*, v. 2, p. 429–434, 2005.

ROCHA, R. L. A. Tecnologia adaptativa aplicada ao processamento computacional de língua natural. *Revista IEEE Latin America*, v. 5, n. 7, p. 544–551, 2007.

SAON, G.; CHIEN, J.-T. Large-vocabulary continuous speech recognition systems: a look at some recent advances. *Signal Processing Magazine, IEEE*, v. 29, n. 6, p. 18–33, 2012.

SEBESTA, R. W. *Concepts of Programming Languages*. 10th. ed. [S.l.]: Pearson Education, Inc., 2012. ISBN 978-0-13-139531-2.

SEVERO, R. T. Língua e linguagem como organizadoras do pensamento em Saussure e Benveniste. *Entretextos*, Londrina, v. 13, n. 1, p. 80–96, 2014. Disponível em: <http://www.uel.br/revistas/uel/index.php/entretextos/article/view/14495>. Acesso em: 2 de janeiro de 2019.

SILVA FILHO, R. I.; ROCHA, R. L. A.; GUIRALDELLI, R. H. G. Learning in the limit: a mutational and adaptive approach. In: _____. *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence: Papers from the Ray Solomonoff 85th Memorial Conference, Melbourne, VIC, Australia, November 30 – December 2, 2011*. Berlin, Alemanha: Springer Berlin Heidelberg, 2013. p. 106–118. ISBN 978-3-642-44958-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-44958-1_8>. Acesso em: 2 de janeiro de 2019.

SIPSER, M. *Introduction to the theory of computation*. Boston, EUA: Thomson Course Technology, 2006. ISBN 0-534-95097-3.

TANIWAKI, C. Y. O. *Formalismos adaptativos na análise sintática de linguagem natural*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 2001.

TAYLOR, A.; MARCUS, M.; SANTORINI, B. The Penn Treebank: an overview. In: _____. *Treebanks: Building and Using Parsed Corpora*. Dordrecht: Springer Netherlands, 2003. p. 5–22. ISBN 978-94-010-0201-1. Disponível em: <https://doi.org/10.1007/978-94-010-0201-1_1>. Acesso em: 2 de janeiro de 2019.

TESNIÈRE, L. *Elements of structural syntax*. Amsterdam: John Benjamins Publishing Company, 2015. ISBN 978-90-272-1212-2.

WEIZENBAUM, J. Eliza – a computer program for the study of natural language communication between man and machine. *Commun. ACM*, ACM, New York, EUA, v. 9, n. 1, p. 36–45, jan. 1966. ISSN 0001-0782.

WINOGRAD, T. Understanding natural language. *Cognitive psychology*, Elsevier, v. 3, n. 1, p. 1–191, 1972.

WIRTH, N. What can we do about the unnecessary diversity of notation for syntactic definitions? *Communications of the ACM*, v. 10, n. 11, p. 822–823, nov 1977.

WISNIEWSKI, G.; LACROIX, O. A systematic comparison of syntactic representations of dependency parsing. In: WORKSHOP ON UNIVERSAL DEPENDENCIES, 2017, **Proceedings...**. Gothenburg, Suécia: Association for Computational Linguistics, 2017. p. 146–152.

YERGEAU, F. *UTF-8, a transformation format of ISO 10646*. [S.l.], 2003. 1-14 p. Disponível em: <http://www.rfc-editor.org/rfc/rfc3629.txt>. Acesso em: 2 de janeiro de 2019.

APÊNDICE A

APÊNDICE

A.1 AÇÕES SEMÂNTICAS ASSOCIADAS AO RECONHECEDOR DE GRAMÁTICA EM WIRTH MODIFICADA - ETAPAS 1 E 2

São apresentados abaixo trechos relevantes do código em Java que implementam as ações semânticas associadas às transições do reconhecedor de sentenças na notação de Wirth modificada.

A.1.1 AÇÕES SEMÂNTICAS DA ETAPA 1

As Figuras A.1 a A.6 apresentam as ações semânticas executadas nas transições do reconhecedor da Etapa 1.

Figura A.1: Etapa 1 - Cria nova submáquina.

```
// Cria nova estrutura de dados para nao terminal
currentNterm = labelGrammar.newNterm(token.getValue());

// Cria nova estrutura de dados para a regra de producao no nao terminal
currentProduction = currentNterm.addProduction(token.getValue());
```

Fonte: autor.

Figura A.2: Etapa 1 – Nova regra de produção.

```
// Cria nova estrutura de dados para dados de rotulos e insere o simbolo "["
registerLabelToken("[");
```

Fonte: autor.

Figura A.3: Etapa 1 – Finaliza regra de produção.

```
// Registra dado do token de fim de regra de producao com o metassimbolo "."
registerExpressionToken(token);
// Registra rotulo "]"
registerLabelToken("]");
// Associa dados da regra de producao ao rotulo do tipo nao terminal
currentProduction.labels.getLast().pushLabel(currentProduction.getIdentifier()
    ,currentProduction);
```

Fonte: autor.

Figura A.4: Etapa 1 – Nova transição com não terminal.

```
// Registra dado do token de nao terminal
registerExpressionToken(token);
// Registra dado de rotulo nulo
registerLabelToken(null);
```

Fonte: autor.

Figura A.5: Etapa 1 – Nova transição com terminal

```
// Registra dado do token terminal
registerExpressionToken(token);
// Registra dado de rotulo com o valor do terminal
registerLabelToken(token.getValue());
```

Fonte: autor.

Figura A.6: Etapa 1 – Nova transição com metassímbolo ϵ

```
// Registra dado do token com o metassimbolo ϵ
registerExpressionToken(token);
// Registra dado de rotulo com o metassimbolo ϵ
registerLabelToken("ϵ");
```

Fonte: autor.

A.1.2 AÇÕES SEMÂNTICAS DA ETAPA 2

As Figuras A.7 a A.15 apresentam as ações semânticas executadas nas transições do reconhecedor da Etapa 2.

Figura A.7: Etapa 2 – Cria nova submáquina.

```
helperStack.clear(); // Inicializa pilha auxiliar
current = 0; // Inicializa indicador de estado corrente
counter = 1; // Inicializa contador de estados
machine = token.getValue(); // Obtem nome da submaquina

// Determina o tipo da primeira transicao da submaquina em vazio
token.setType("ϵ");
token.setValue("ϵ");

// Inicializa rotulo de inicio de submaquina com "["
LinkedList<LabelElement> initialLabelElements;
LabelElement initialLabelElement;
initialLabelElements = new LinkedList<>();
initialLabelElement = new LabelElement();
initialLabelElement.setValue("[");
initialLabelElements.add(initialLabelElement);
token.getProductionToken().addPostLabels(initialLabelElements);

// Define estrutura de dados de transicao
transition = new Sketch(machine, current, token, counter);
transitions.add(transition);

// Atualiza indicador de estado corrente
current = counter;
// Incrementa contador de estado
counter++;

// Determina nome de submaquina
if (main == null) {
    main = machine;
}
```

Fonte: autor.

Figura A.8: Etapa 2 – Nova regra de produção.

```
// Armazena estados de inicio e final de uma submaquina
helperStack.push(new Pair<>(current, counter));

// Incrementa contador de estado
counter++;
```

Fonte: autor.

Figura A.9: Etapa 2 – Finaliza regra de produção.

```
// Determina tipo da ultima transicao da submaquina em vazio
token.setType("ϵ");
token.setValue("ϵ");

// Define estrutura de dados de transicao
Sketch transition = new Sketch(machine, current, token, helperStack.top().
    getSecond());
transitions.add(transition);

// Atualiza indicador de estado corrente
current = helperStack.top().getSecond();

helperStack.pop(); // Desempilha par de indicadores de estado
```

Fonte: autor.

Figura A.10: Etapa 2 – Nova transição com não terminal, terminal ou ϵ.

```
/* Define estrutura de dados da transicao cujo conteudo do token eh do
tipo nao terminal, terminal ou vazio ϵ */
Sketch transition = new Sketch(machine, current, token, counter);
transitions.add(transition);

// Atualiza indicador de estado corrente
current = counter;
counter++; // Incrementa contador de estado
```

Fonte: autor.

Figura A.11: Etapa 2 – Novo escopo.

```
// Empilha novo par de estados de inicio e fim de escopo
helperStack.push(new Pair<>(current, counter));

counter++; // Incrementa contador de estado
```

Fonte: autor.

Figura A.12: Etapa 2 – Fecha escopo.

```
// Determina o tipo de transicao em vazio
token.setType("ϵ");
token.setValue("ϵ");

// Define estrutura de dados de transicao
Sketch transition = new Sketch(machine, current, token, helperStack.top().
    getSecond());
transitions.add(transition);

// Atualiza indicador de estado corrente
current = helperStack.top().getSecond();

helperStack.pop(); // Desempilha par de indicadores de estado
```

Fonte: autor.

Figura A.13: Etapa 2 – Adiciona opção.

```
// Determina tipo da transicao em vazio para o inicio da opcao
token.setType("ϵ");
token.setValue("ϵ");

// Define estrutura de dados de transicao
Sketch transition = new Sketch(machine, current, token, helperStack.top().
    getSecond());
transitions.add(transition);

// Atualiza indicador de estado corrente para apontar o inicio do escopo atual
current = helperStack.top().getFirst();
```

Fonte: autor.

Figura A.14: Etapa 2 – Adiciona reverso.

```
// Determina o tipo de transicao em vazio
token.setType("ϵ");
token.setValue("ϵ");

// Define de transicao em vazio entre o estado atual e o inicio do escopo
    atual
Sketch transition = new Sketch(machine, current, token, helperStack.top().
    getSecond());
transitions.add(transition);

// Atualiza contador
current = helperStack.top().getSecond();

// Empilha novo par de indicadores de estados com escopo alternativo
helperStack.push(new Pair<>(helperStack.top().getSecond(), helperStack.top().
    getFirst()));
```

Fonte: autor.

Figura A.15: Etapa 2 – Fecha escopo reverso.

```
// Determina o tipo de transicao em vazio
token.setType("ϵ");
token.setValue("ϵ");

// Define transicao em vazio entre o estado atual e o final do escopo atual
Sketch transition = new Sketch(machine, current, token, helperStack.top().
    getSecond());
transitions.add(transition);

helperStack.pop(); // Desempilha par de indicadores de estado

// Atualiza indicador de estado corrente
current = helperStack.top().getSecond();

helperStack.pop();  // Desempilha par de indicadores de estado
```

Fonte: autor.

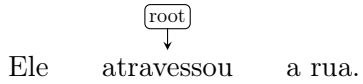
A.2 RELAÇÕES DE DEPENDÊNCIA UNIVERSAIS PARA O PORTUGUÊS

As Figuras A.16 a A.25 apresentam os tipos de relações de dependência encontradas no português conforme levantamento realizado por Rademaker et al. (2017) com os exemplos disponíveis publicamente[1].

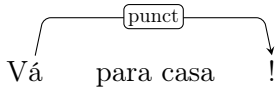
Em cada tipo, ao lado do nome em português foi colocado entre parênteses a nomenclatura adotada em inglês adotada nos trabalhos originais de *Universal Dependencies*.

Figura A.16: Relações para auxílio na anotação.

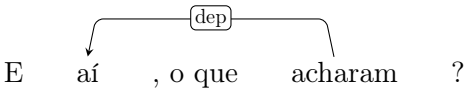
Raiz *(root - root)* – Indica a raiz da sentença.



Pontuação *(punct - punctuation)* – É a relação usada para os caracteres de pontuação numa oração.



Dependência não classificada *(dep - unspecified dependency)* – São dependências que não foram classificadas.



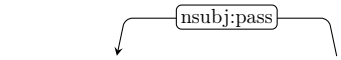
Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

[1]Disponível em <http://universaldependencies.org/treebanks/pt_bosque/index.html> Acesso em: 2 de janeiro de 2019.

Figura A.17: Relações de dependência principais.

Sujeito *(nsubj - nominal subject)* – É a relação de dependência entre o sujeito de uma oração e o verbo.

Um caso particular é a relação de dependência entre o sujeito e o verbo numa oração na voz passiva.



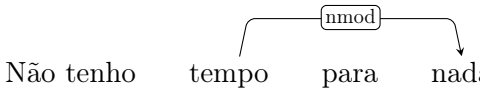
Objeto direto *(obj - object)* – É a relação de dependência existente entre o objeto direto e o verbo transitivo.

Objeto indireto *(iobj - indirect object)* – É a relação de dependência existente entre o objeto indireto e o verbo transitivo.

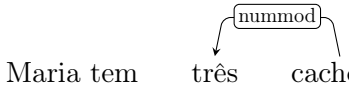
Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.18: Relações de dependência nominal.

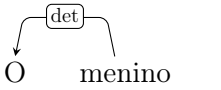
Modificador nominal *(nmod - nominal modifier)* – Esta relação é utilizada para indicar a dependência de uma expressão nominal de um substantivo ou oração substantiva.



Modificador numérico *(nummod - numeric modifier)* – O modificador numérico de um substantivo é qualquer frase numérica que modifica o sentido com uma quantidade.

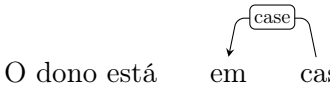


Determinante *(det - determiner)* – A relação de determinante ocorre entre um substantivo e o seu determinador. Os exemplos típicos de determinante são os artigos.

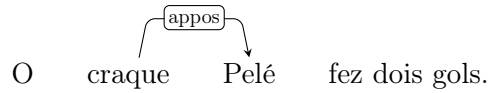


Determinante possessivo *(det:poss - determiner possessive)* – A relação de determinante ocorre entre um substantivo e um pronome possessivo associado.

Marcação de caso *(case - case marking)* – Refere-se à relação entre um elemento de marcação de caso, que seja tratado como uma palavra sintática separada, e a palavra ou oração que ele modifica.
Os elementos de marcação de caso são dependentes do substantivo ou oração que eles atribuem ou introduzem.



Modificador apositivo *(appos - appositional modifier)* – É a relação entre o aposto e outra palavra com valor substantivo ou pronominal que ele modifica, define, nomeia ou descreve. Palavras entre parênteses, assim como a definição de abreviações estão incluídas neste tipo de relação.

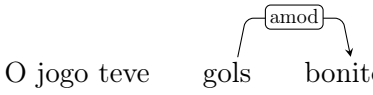


Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

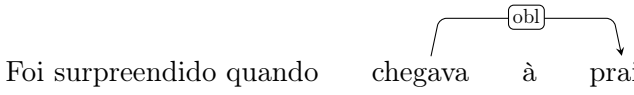
Figura A.19: Relações de dependência secundárias.

Modificador adverbial *(advmod - adverbial modifier)* – Corresponde à relação entre o advérbio e o verbo que ele modifica.

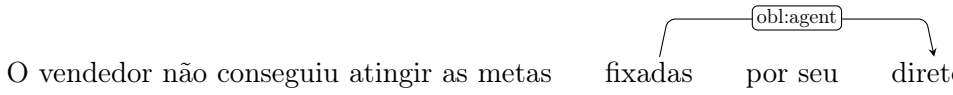
Modificador adjetivo *(amod - adjectival modifier)* – Corresponde à relação entre o adjetivo e o substantivo que ele modifica.



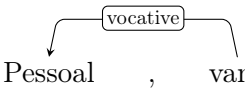
Oblíquo nominal *(obl - oblique nominal)* – É a relação entre um nominal (substantivo, pronome, ou oração nominal) usado como argumento não principal. Ele corresponde funcionalmente a um advérbio associado a um verbo, adjetivo ou outro advérbio.



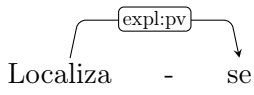
Esta relação também é utilizada para a dependência entre o agente da voz passiva e o verbo.



Vocativo *(vocative - vocative)* – É a relação existente entre o vocativo e a oração dirigida ao interlocutor.



Expletivo - verbo pronomial *(expl:pv - expletive pronomial verb)* – É a relação na qual a partícula *se* é dependente de um verbo pronomial.

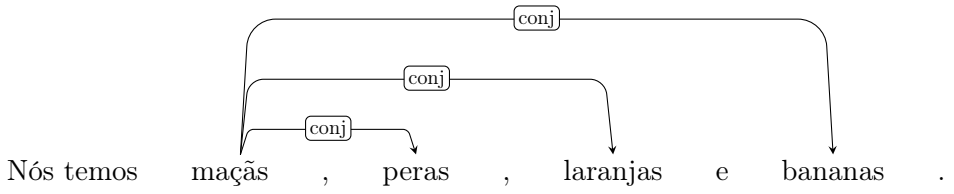


Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.20: Relações de dependência de coordenação.

Conjunção coordenativa *(cc - coordinating conjunction)* – Refere-se à relação de dependência entre o elemento de uma conjunção e a palavra precedente que faz o papel da conjunção.

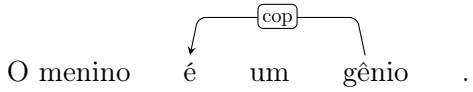
Associado *(conj - conjunct)* – É a relação entre dois elementos conectados por uma conjunção de coordenação. Na representação adotada, as conjunções são tratadas de forma assimétrica. O primeiro elemento da relação é considerado como item principal, e os demais elementos dependem dele.



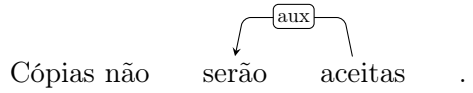
Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.21: Relações de dependência de palavras funcionais.

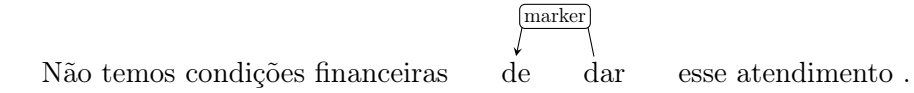
Copula *(cop - copula)* – A cópula é a relação na qual o verbo de ligação é dependente do predicativo do sujeito.



Auxiliar *(aux - auxiliary)* – Refere-se à relação na qual um verbo auxiliar é dependente do verbo principal numa oração. Os verbos *ser*, *fazer*, *ter* em expressões perifrásticas, assim como *começar*, *acabar*, *terminar* são exemplos deste tipo de relação.



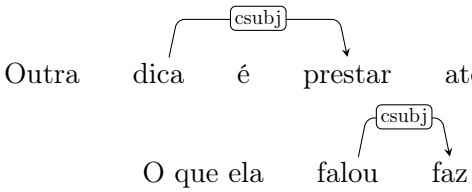
Marcador *(mark - marker)* – É a relação na qual a palavra que introduz uma oração subordinada é dependente do verbo dessa oração.



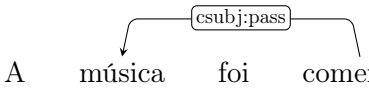
Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.22: Relações de dependência principais entre orações.

Sujeito oracional *(csubj - clausal subject)* – É a relação na qual a oração subordinada subjetiva é dependente do verbo.

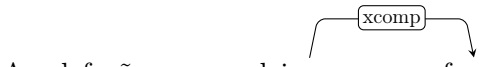


Sujeito oracional passivo *(csubj:pass - clausal subject)* – A oração subordinada subjetiva é dependente do verbo na voz passiva.

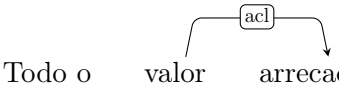


Complemento oracional *(ccomp - clausal complement)* – É a relação na qual uma oração subordinada é dependente de um verbo, como nas orações subordinadas objetiva direta, objetiva indireta e predicativa. Ocorre também quando a oração é dependente de um adjetivo.

Complemento *(xcomp - open clausal complement)* – É a relação de dependência entre um verbo ou adjetivo e uma oração subordinada sem o seu sujeito, ou um predicativo.

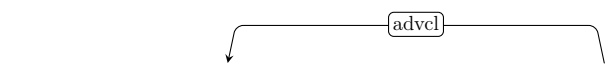


Oração modificadora de nome - orações adjetivas *(acl - clausal modifier of noun)* – Refere-se a orações subordinadas adjetivas que modificam um nome. O núcleo da oração modificadora é dependente do nome modificado pela mesma.



Em português, há o subtipo no qual a oração é relativa (*relcl - relative clause*).

Oração modificadora adverbial *(advcl - adverbial clause modifier)* – É uma oração subordinada adverbial que modifica um verbo ou outro predicado (adjetivo, etc.). O núcleo da oração modificadora é o dependente.



Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.23: Relações de dependência em expressões multipalavra.

Fixo *(fixed - fixed)* – Esta relação é uma das três que se pode ter em expressões com múltiplas palavras. Nesta relação as palavras tem uma estrutura fixa.

Homogêneo *(flat - flat)* – Esta relação é uma das três que se pode ter em expressões com múltiplas palavras. Nesta relação não há uma hierarquia ou posição fixa entre as palavras. São utilizadas para datas, nomes compostos.

Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

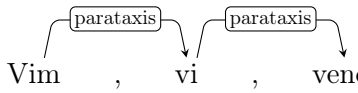
Figura A.24: Relação de dependência especial.

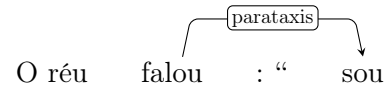
Órfão *(orphan - orphan)* – Esta relação é usada em elipses.

Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

Figura A.25: Relações de dependência fracas.

Parataxe *(parataxis - parataxis)* – Parataxe é uma sequência de frases justapostas, sem conjunção coordenativa.



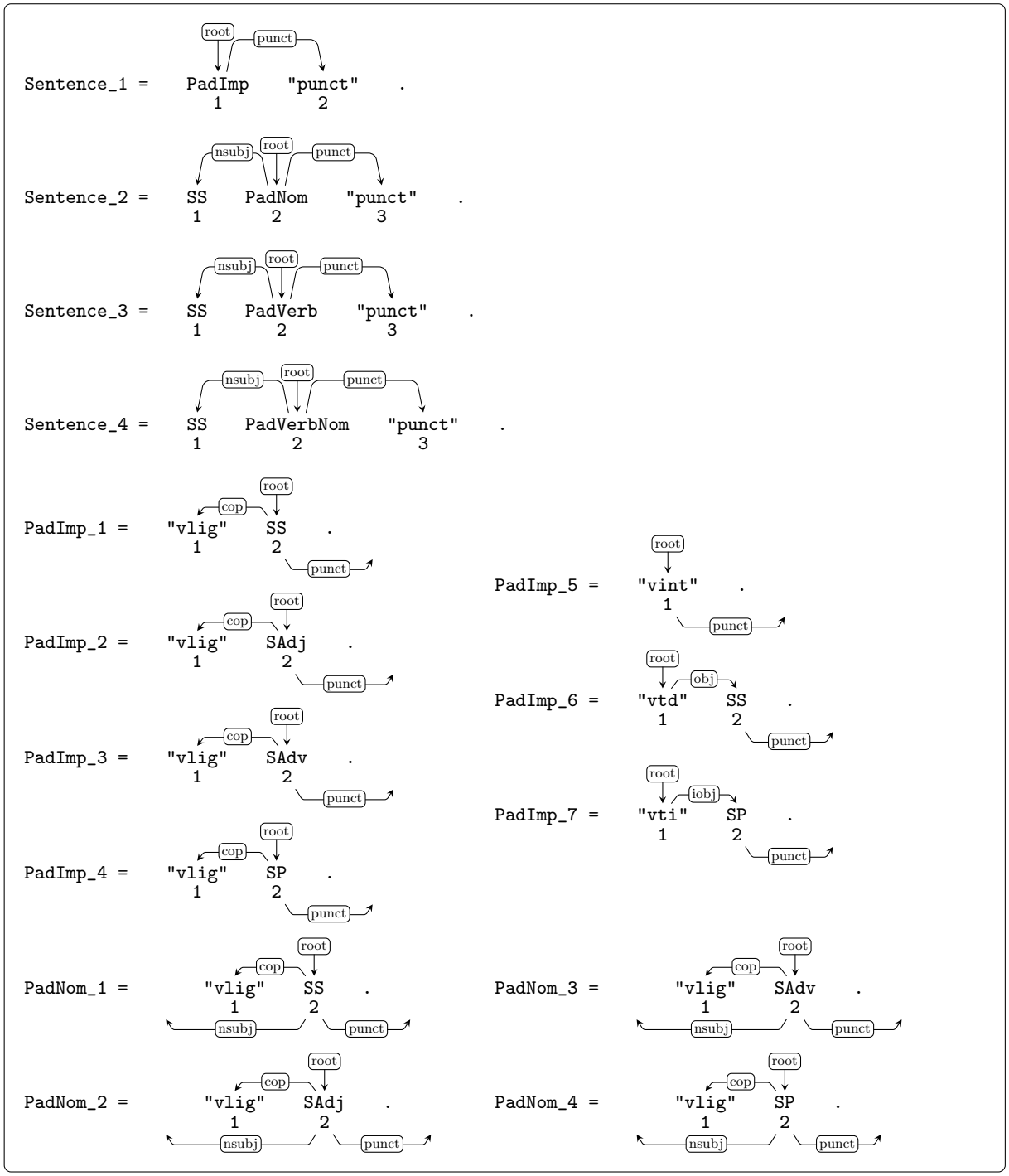


Fonte: adaptado de Rademaker et al. (2017) e Marneffe et al. (2014).

A.3 PADRÕES DE RELAÇÕES DE DEPENDÊNCIA

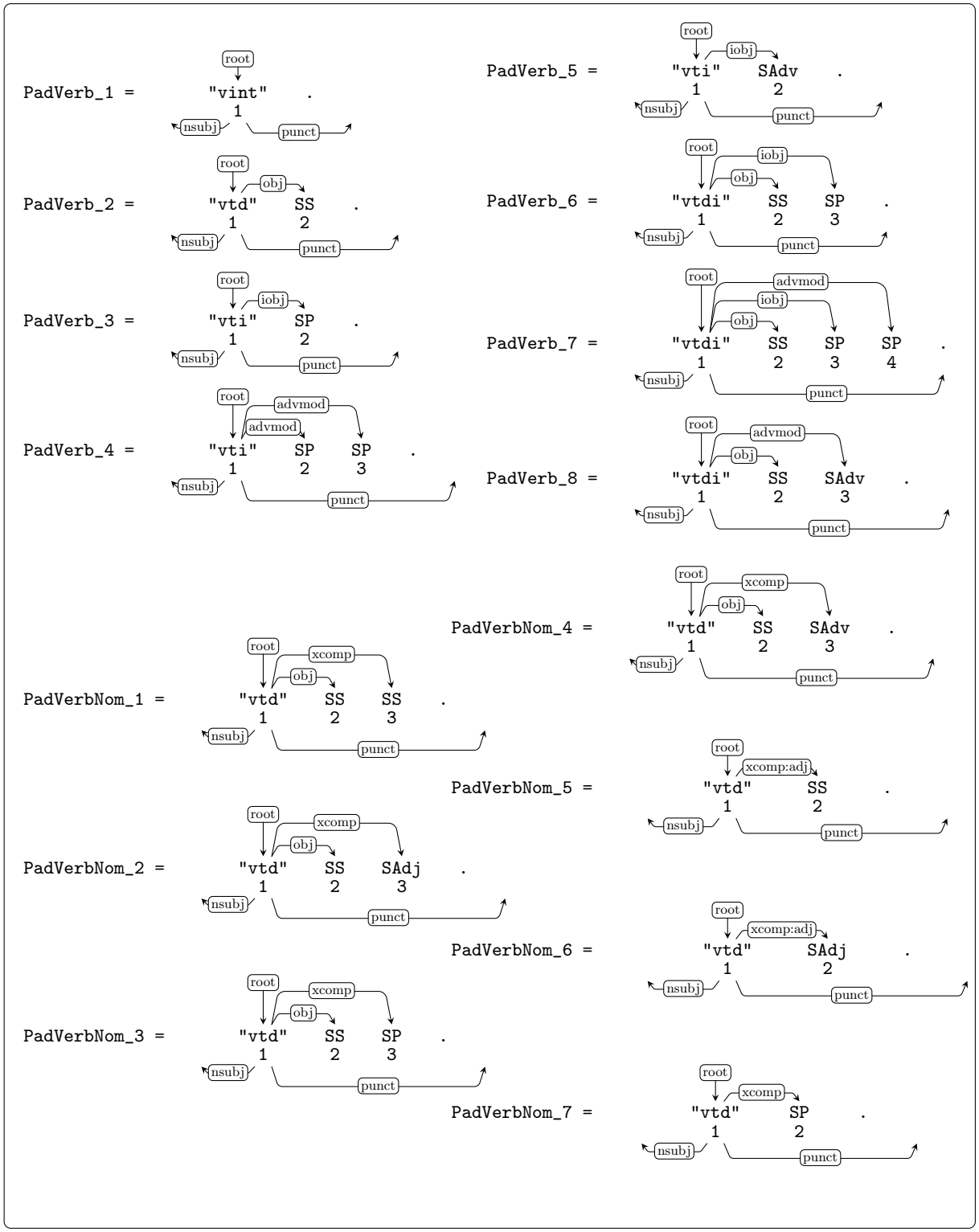
As Figuras A.26 a A.27 apresentam as regras da gramática adotada nos ensaios com a gramática livre de contexto de períodos simples descrita na Figura 6.3, juntamente com os respectivos padrões de relações de dependência correspondentes num formato gráfico.

Figura A.26: Regras de produção e padrões de relações de dependência - 1.



Fonte: adaptado de Luft (2002) e José Neto (2015).

Figura A.27: Regras de produção e padrões de relações de dependência - 2.



Fonte: adaptado de Luft (2002) e José Neto (2015).

As Figuras A.28 a A.32 apresentam as relações de dependências referentes às regras de produção da gramática para períodos simples e compostos descrita na Figura 6.5 e presentes nos exemplos de processamento listadas nas Figuras A.36 a A.42. As informações estão apresentadas conforme organização das estruturas de dados descritas nas Tabelas 5.10 e 5.11.

Figura A.28: Regras de produção e padrões de relações de dependência - 3.

value	OP_4		
main	2		
headDirection			
constituentList			
id	1	2	3
value	SS	SV	punct
type	nterm	nterm	term
head	2	0	2
depRel	nsubj	root	punct
leftDepList			
rightDepList			

value	OA_3		
main	2		
headDirection			
constituentList			
id	1	2	3
value	SS	SN	punct
type	nterm	nterm	term
head	2	0	2
depRel	nsubj	root	punct
leftDepList			
rightDepList			

value	OA_4		
main	2		
headDirection			
constituentList			
id	1	2	3
value	SS	SV	punct
type	nterm	nterm	term
head	2	0	2
depRel	nsubj	root	punct
leftDepList		[nsubj]	
rightDepList		[punct]	

value	O_1
main	1
headDirection	left
constituentList	
id	1
value	SN
type	nterm
head	0
depRel	advcl
leftDepList	
rightDepList	

value	O_3			
main	2			
headDirection	left			
constituentList				
id	1	2	1	2
value	SS	SV	SS	SV
type	nterm	nterm	nterm	nterm
head	2	0	2	0
depRel	nsubj	xcomp	nsubj	advcl
leftDepList				
rightDepList				

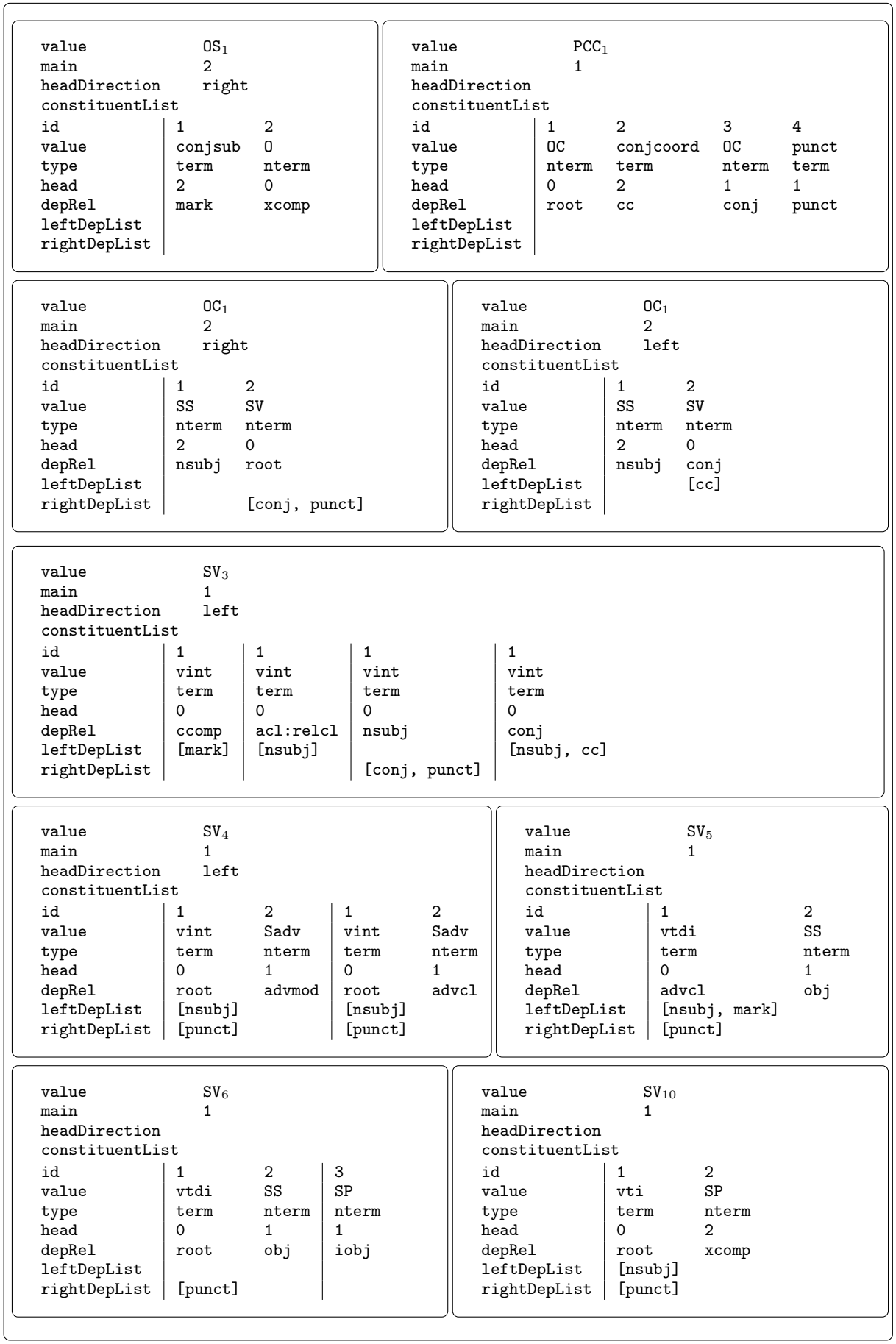
value	O_4
main	1
headDirection	left
constituentList	
id	1
value	SV
type	nterm
head	0
depRel	ccomp
leftDepList	
rightDepList	

value	$OSAR_2$	
main	2	
headDirection	left	
constituentList		
id	1	2
value	SS	SV
type	2	0
head	nsubj	acl:relcl
depRel		
leftDepList		
rightDepList		

value	OS_1			
main	2			
headDirection	left			
constituentList				
id	1	2	1	2
value	conjsub	O	conjsub	O
type	term	nterm	term	nterm
head	2	0	2	0
depRel	mark	advcl	mark	ccomp
leftDepList				
rightDepList				

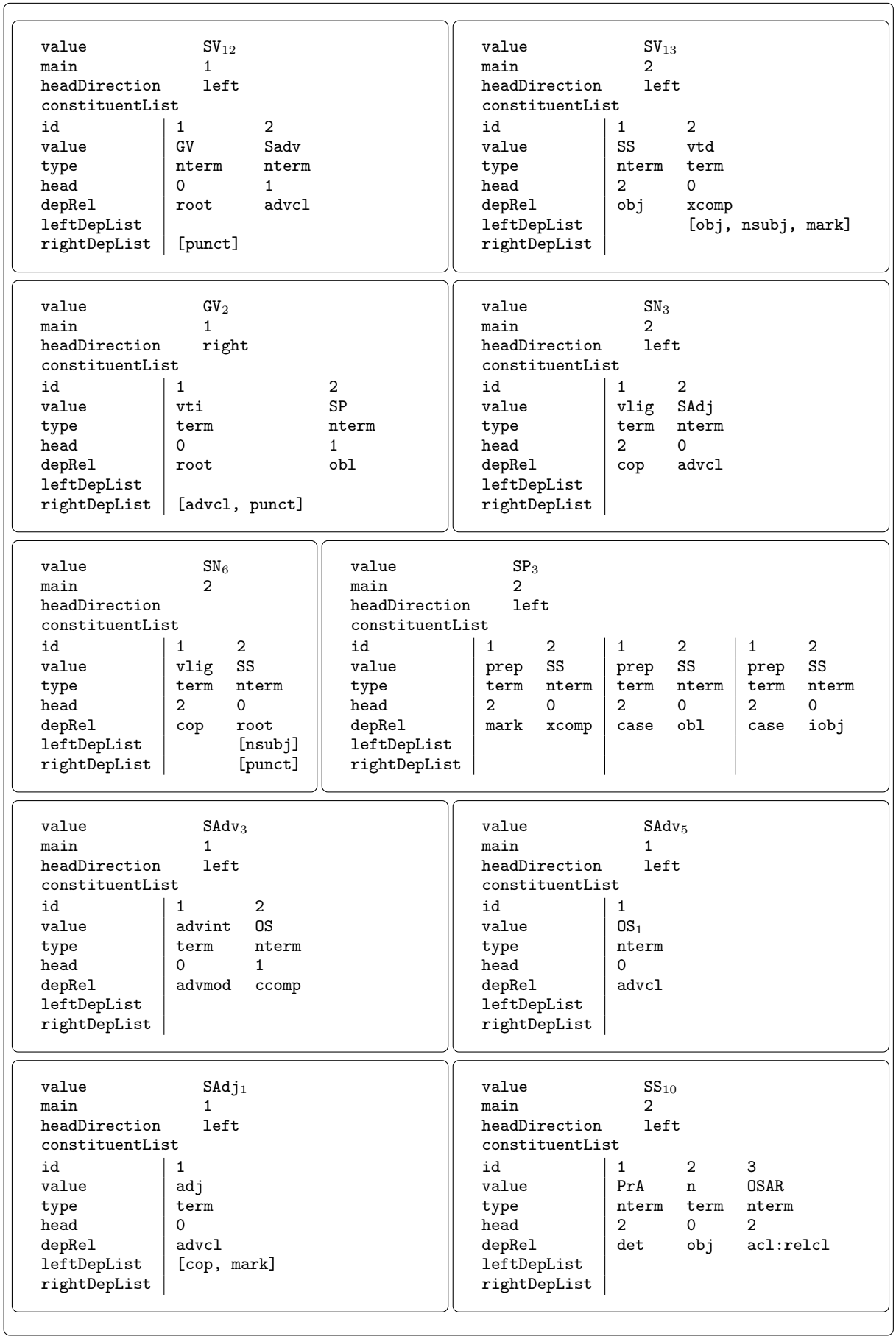
Fonte: autor.

Figura A.29: Regras de produção e padrões de relações de dependência - 4.



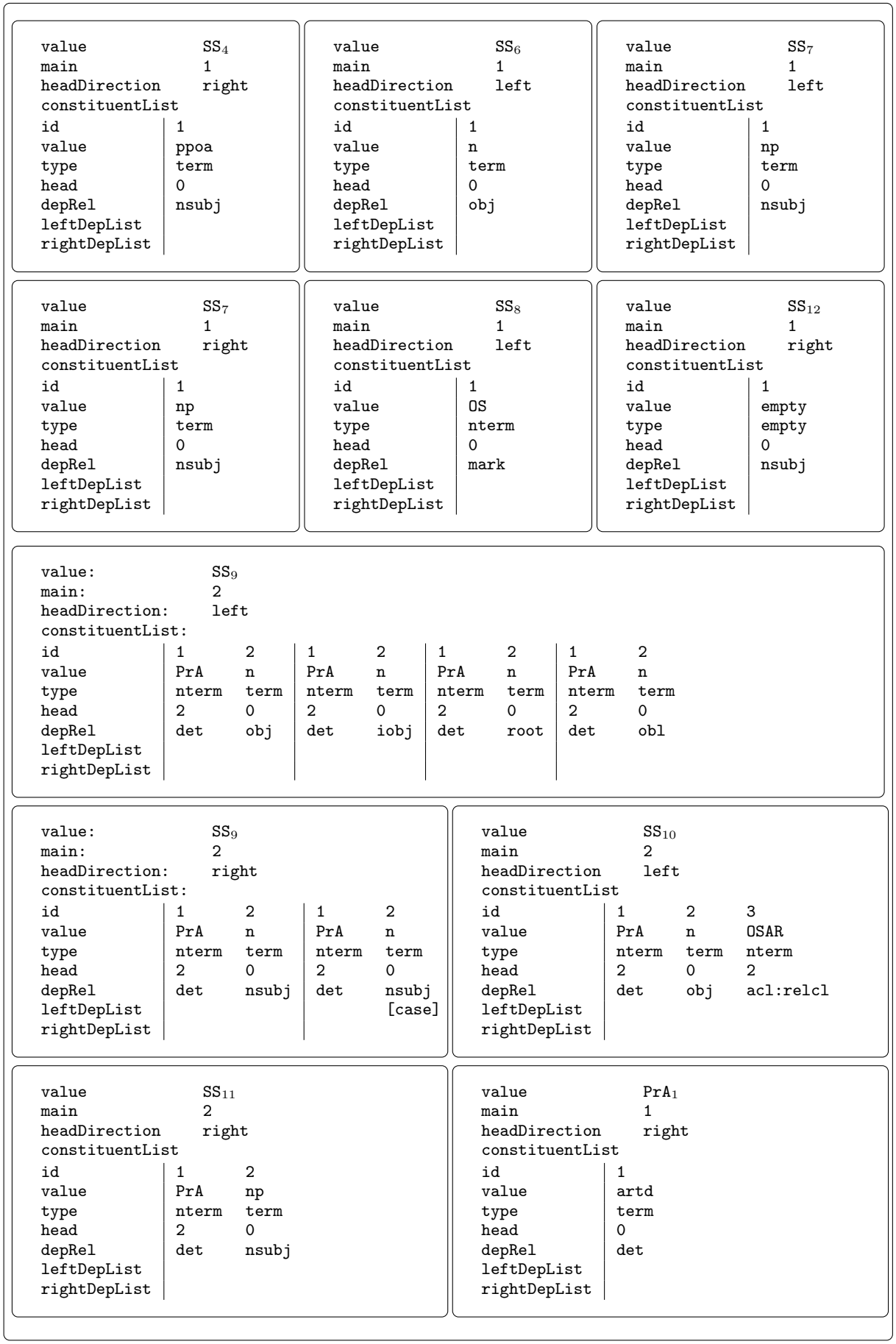
Fonte: autor.

Figura A.30: Regras de produção e padrões de relações de dependência - 5.



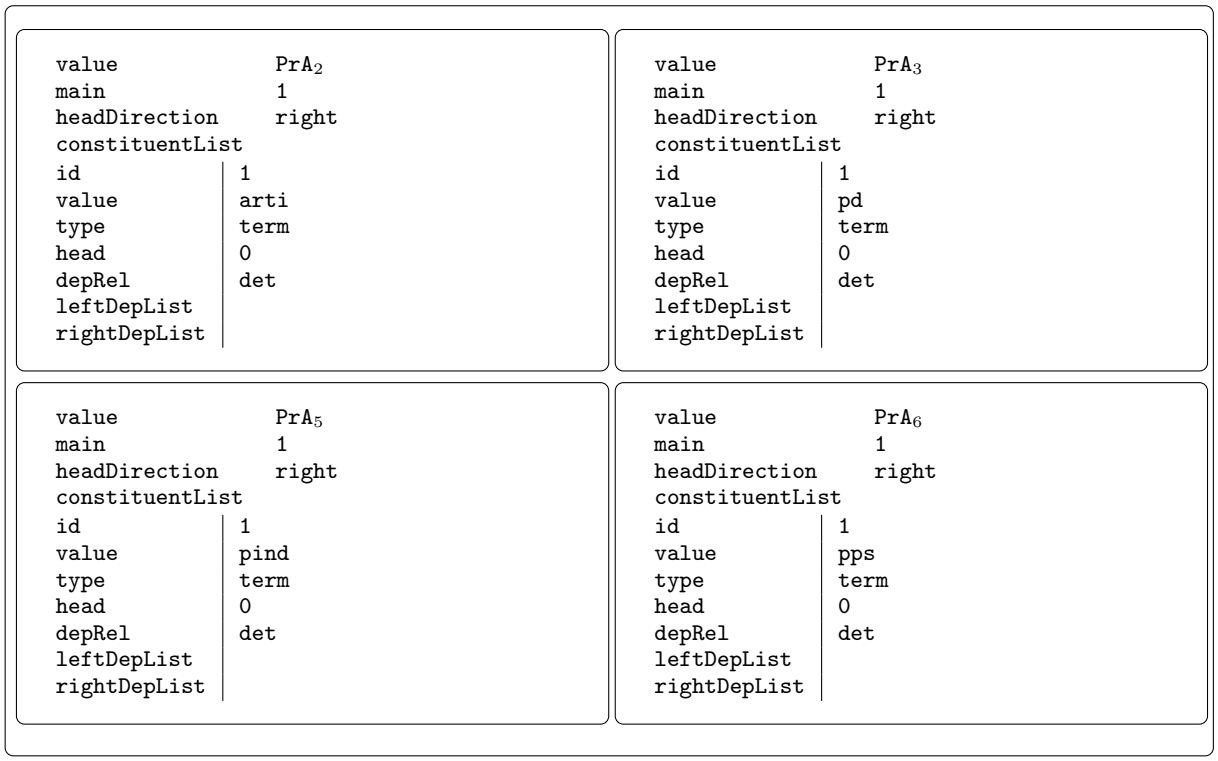
Fonte: autor.

Figura A.31: Regras de produção e padrões de relações de dependência - 6.



Fonte: autor.

Figura A.32: Regras de produção e padrões de relações de dependência - 7.

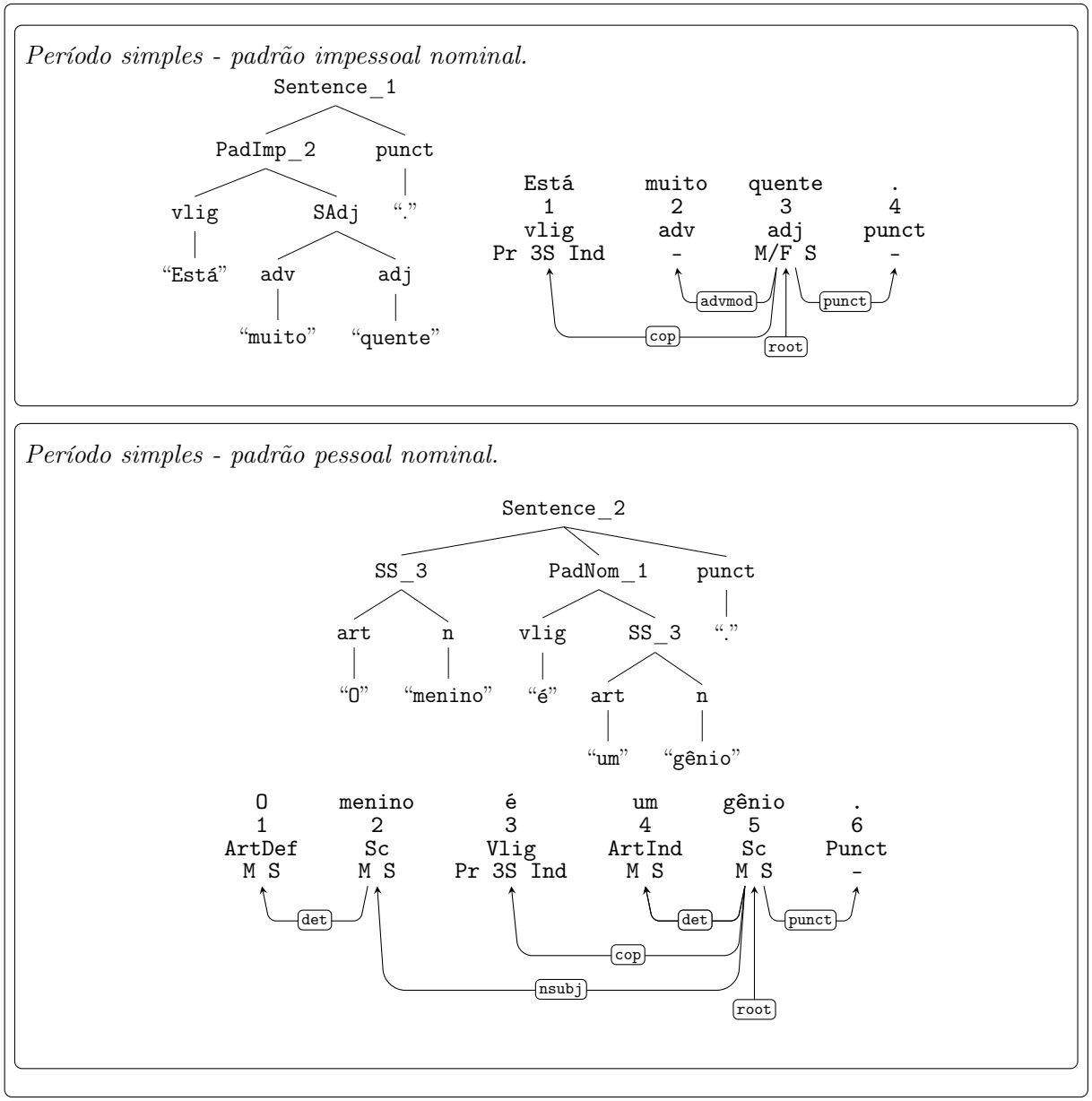


Fonte: autor.

A.4 EXEMPLOS DE PROCESSAMENTO

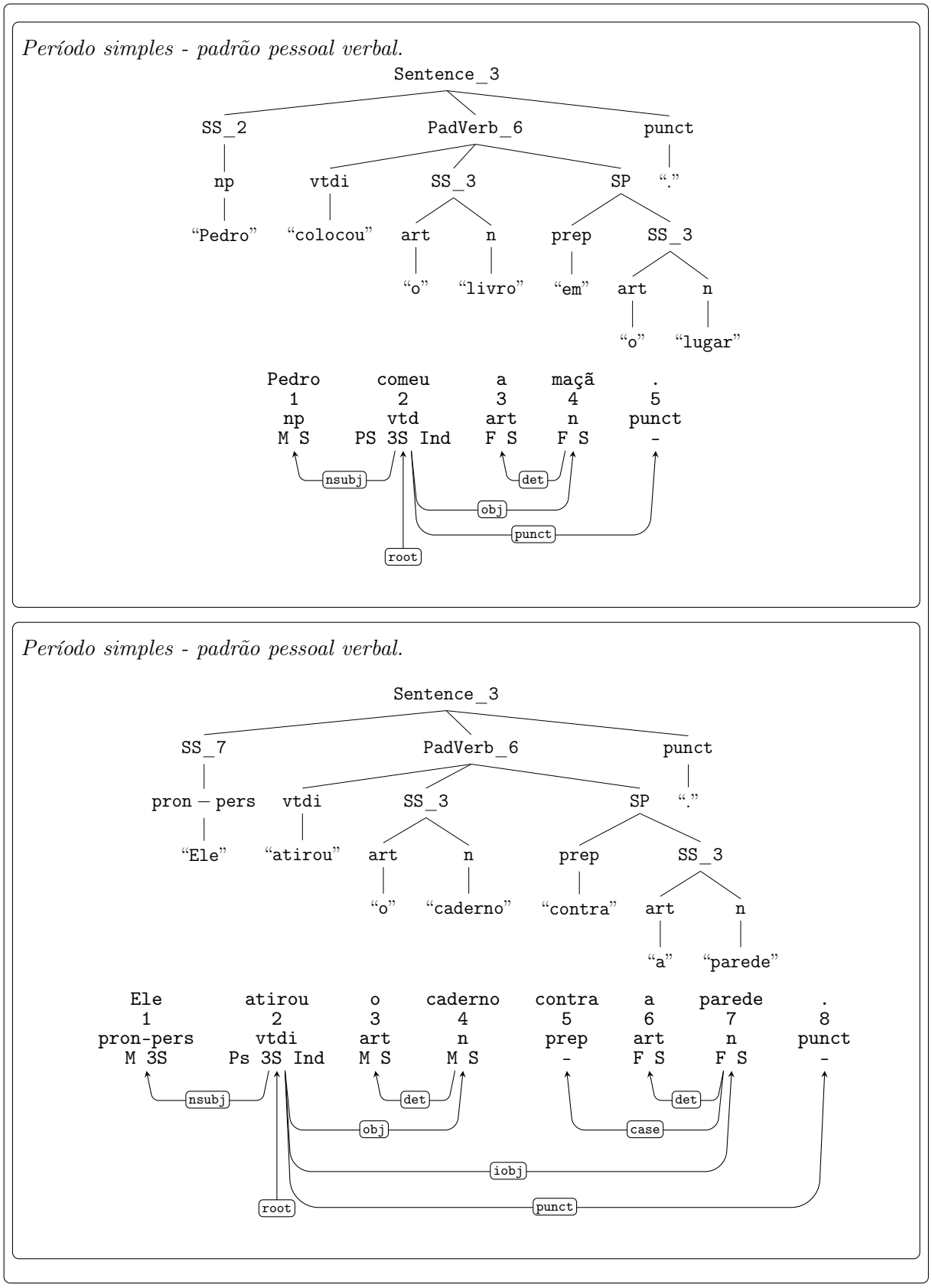
As Figuras A.33 a A.35 apresentam exemplos de saídas de processamento de análise de sentenças referentes à gramática para períodos simples descrita na Figura 6.3 e as Figuras A.36 a A.42 correspondem àquelas da gramática da Figura 6.5. Para facilitar a visualização, os resultados estão representados graficamente no formato de árvores. São elas a árvore estrutural e a de relações de dependência.

Figura A.33: Exemplo de processamento - 1.



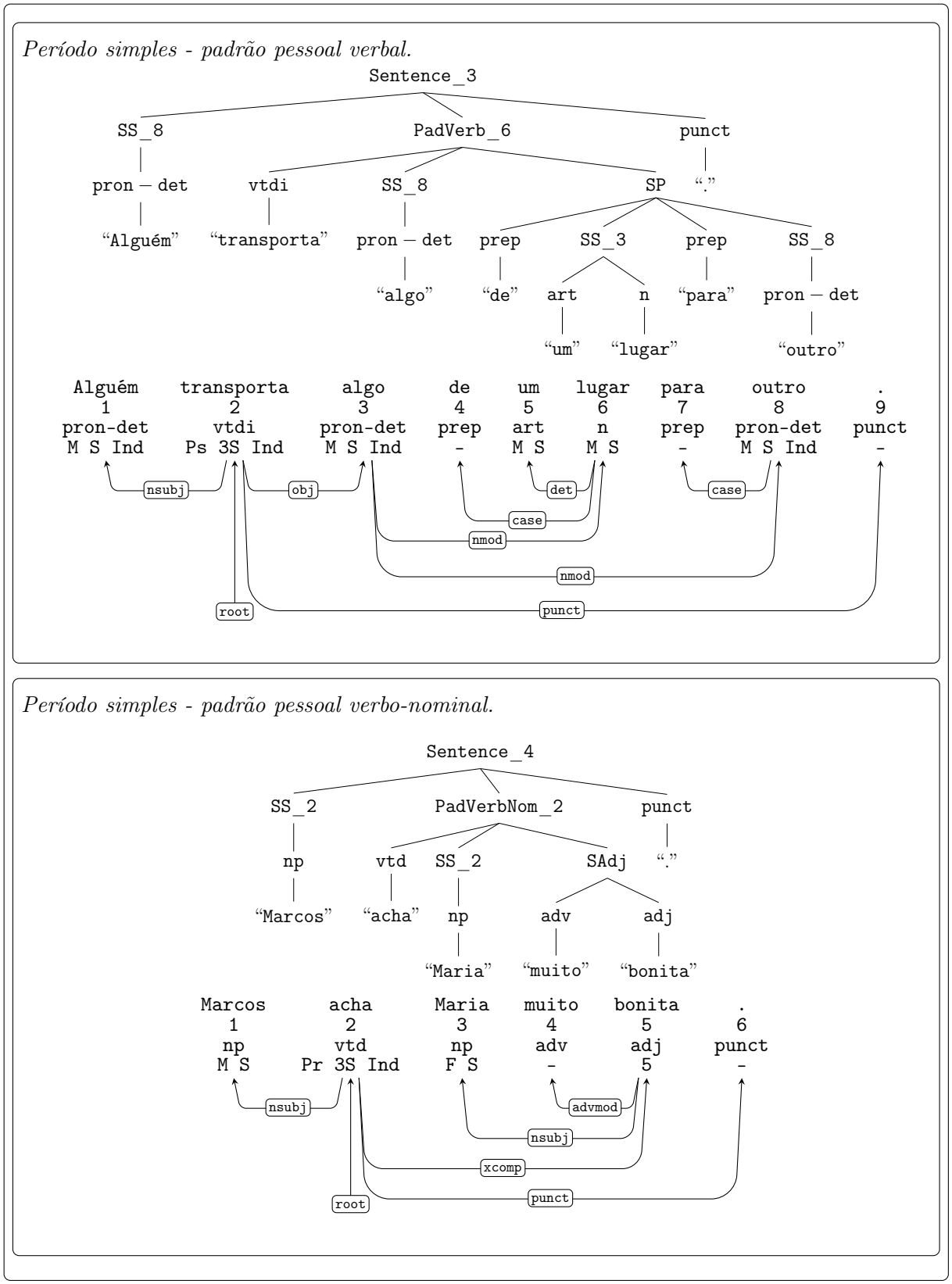
Fonte: autor.

Figura A.34: Exemplo de processamento - 2.



Fonte: autor.

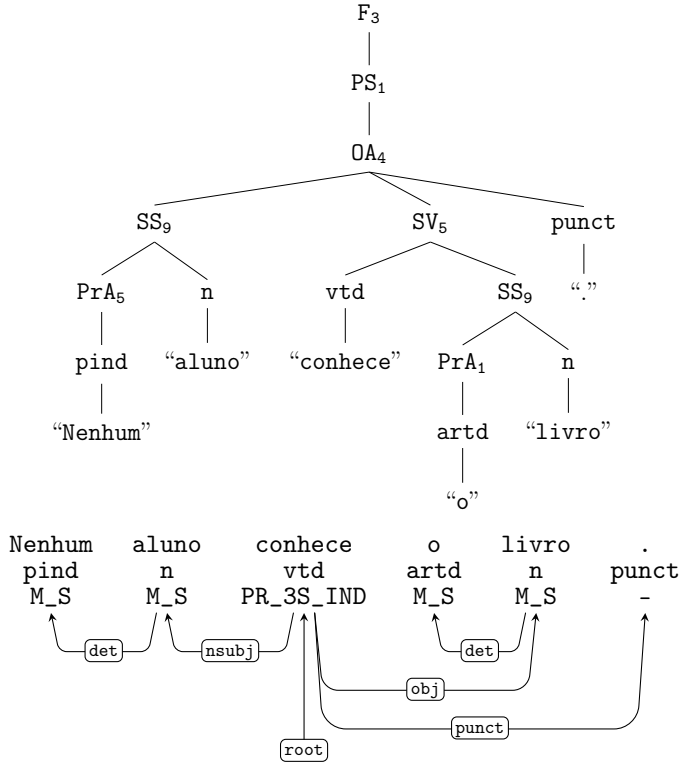
Figura A.35: Exemplo de processamento - 3.



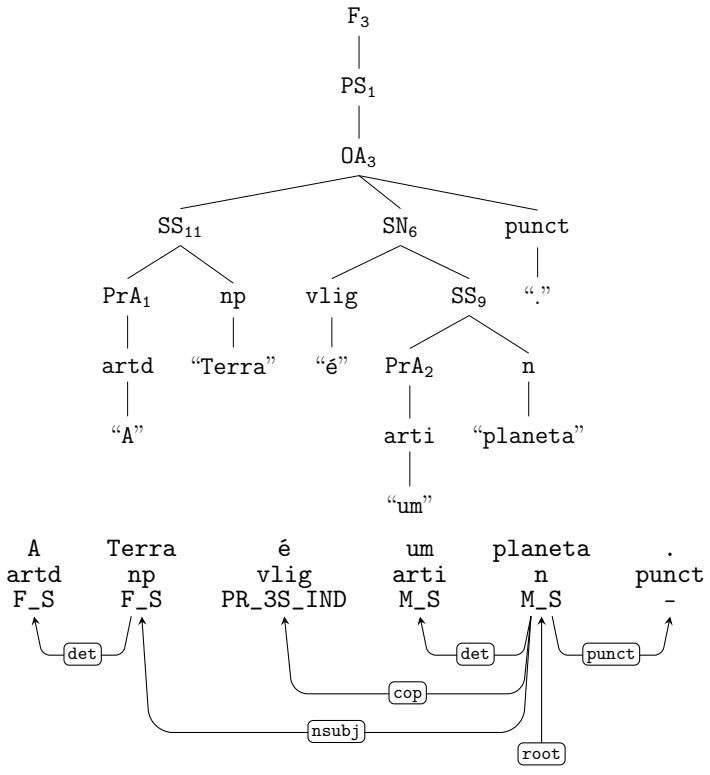
Fonte: autor.

Figura A.36: Exemplo de processamento - 4.

Período simples - oração transitiva.



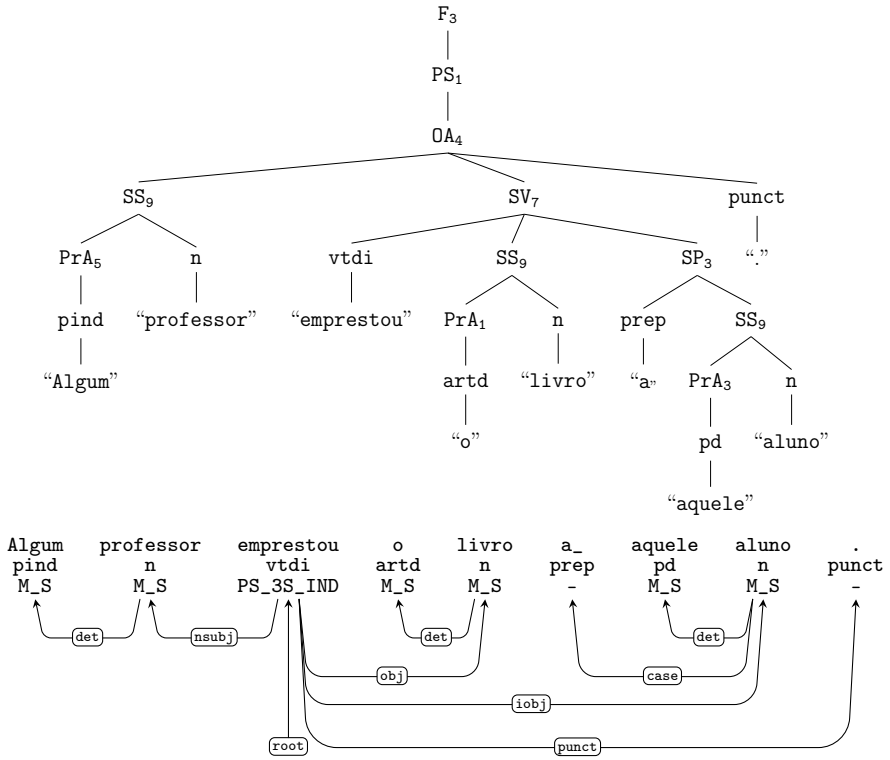
Período simples - oração de predicado nominal.



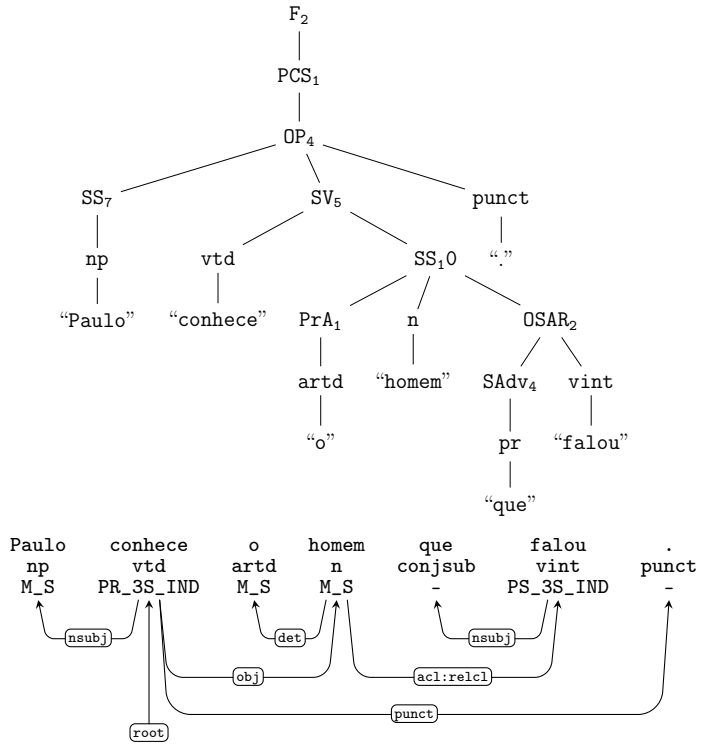
Fonte: autor.

Figura A.37: Exemplo de processamento - 5.

Período simples - oração de verbo transitivo direto e indireto.

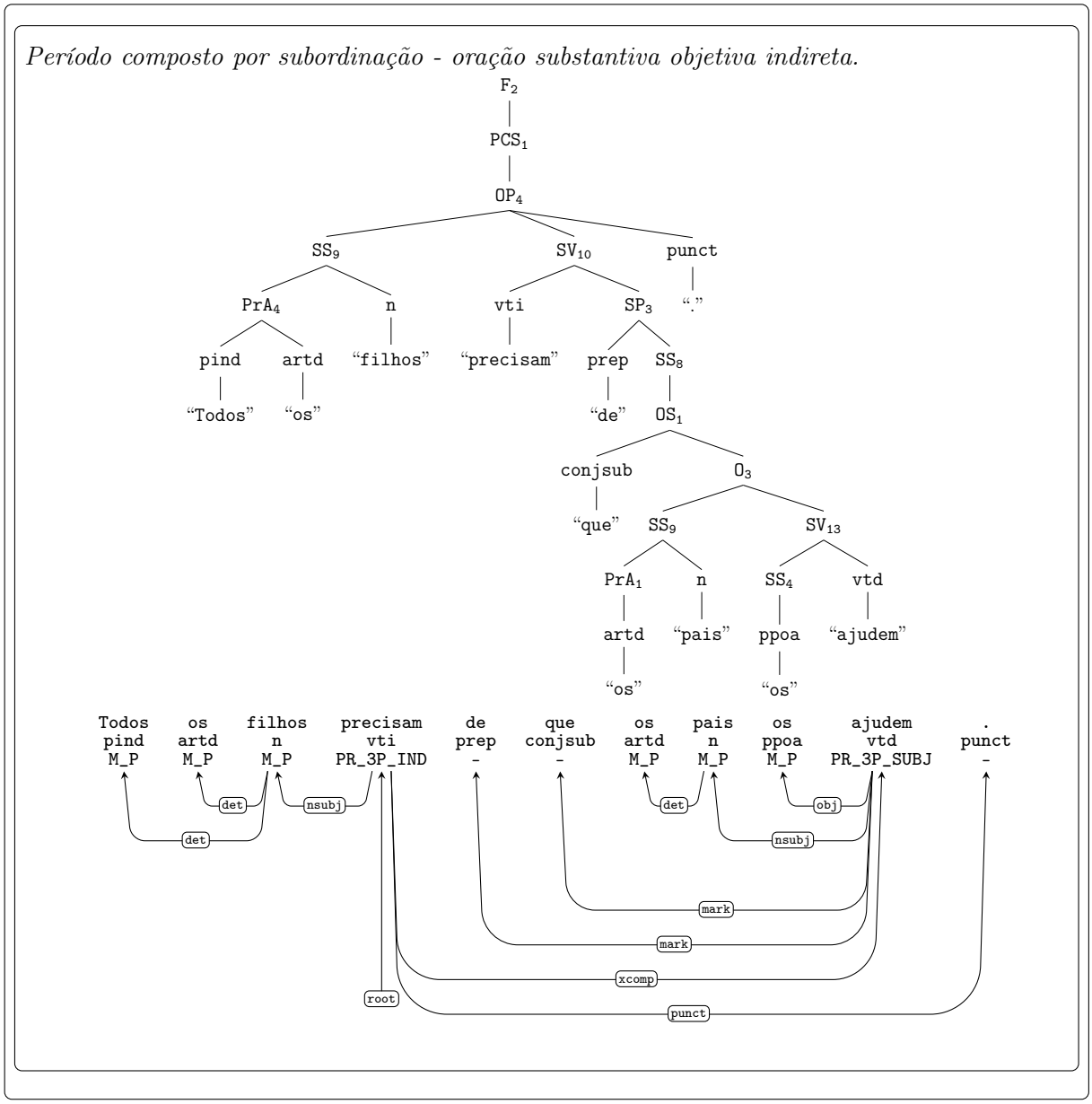


Período composto por subordinação - oração adjetiva restritiva.



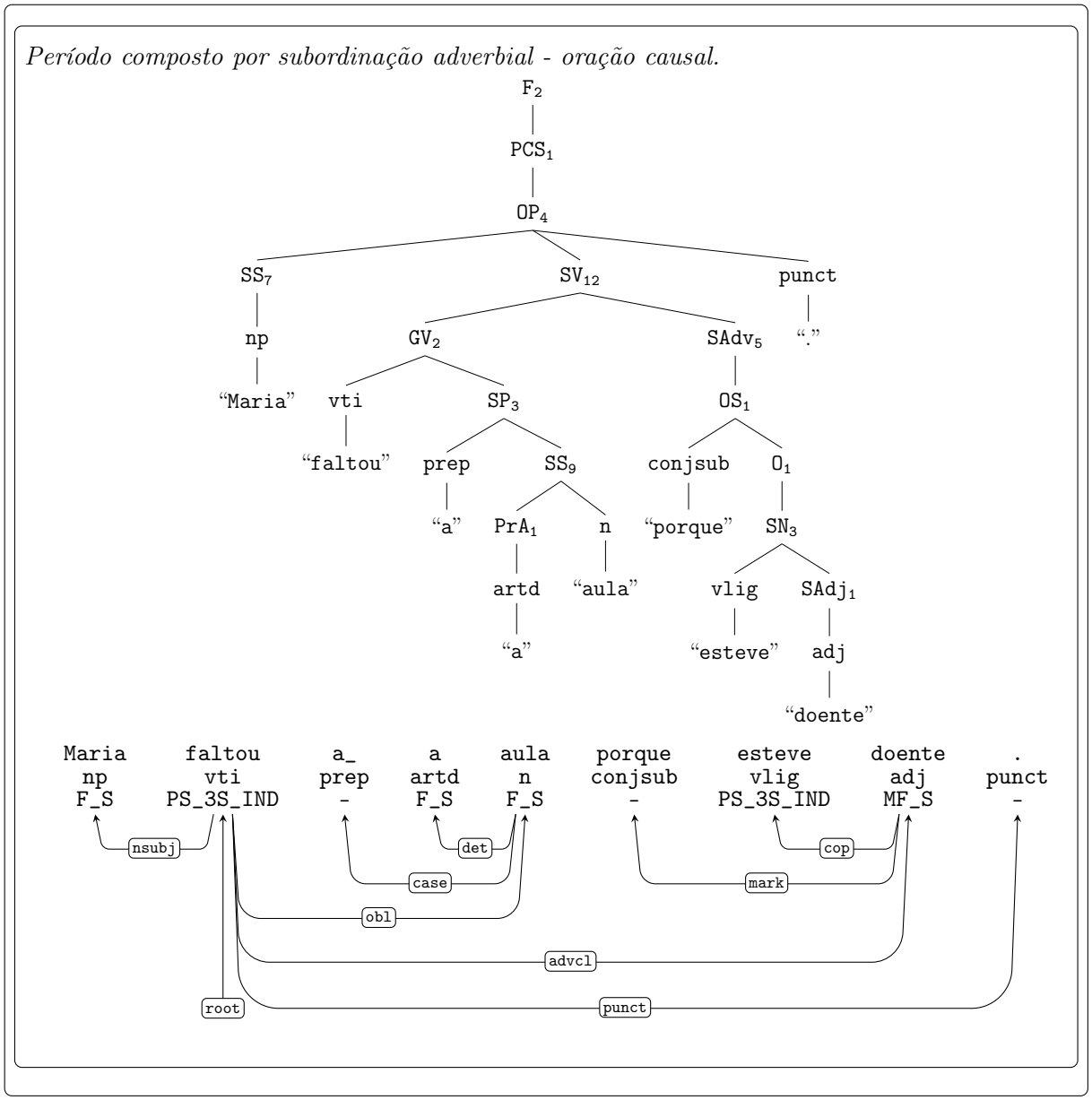
Fonte: autor.

Figura A.38: Exemplo de processamento - 6.



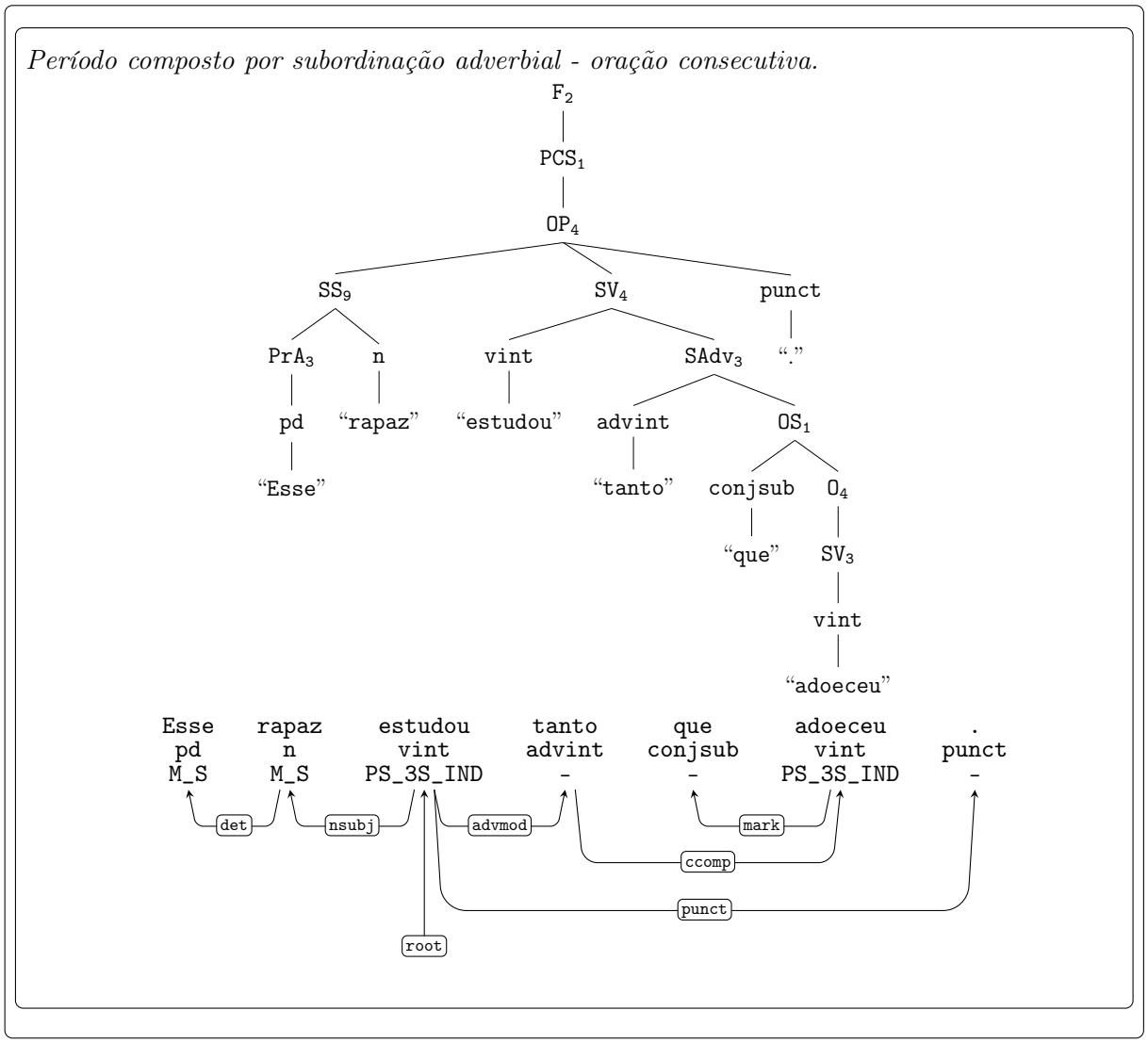
Fonte: autor.

Figura A.39: Exemplo de processamento - 7.



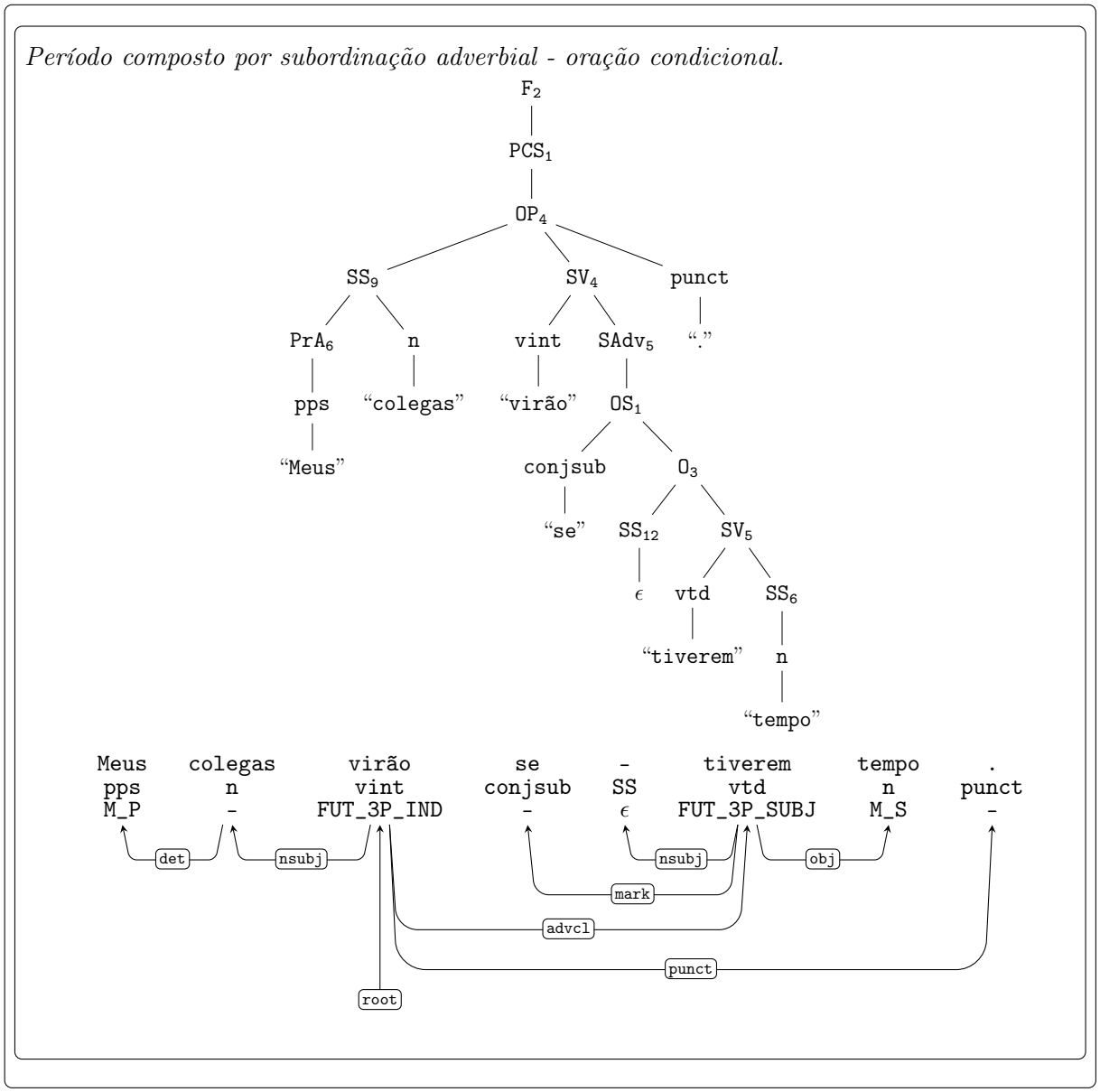
Fonte: autor.

Figura A.40: Exemplo de processamento - 8.



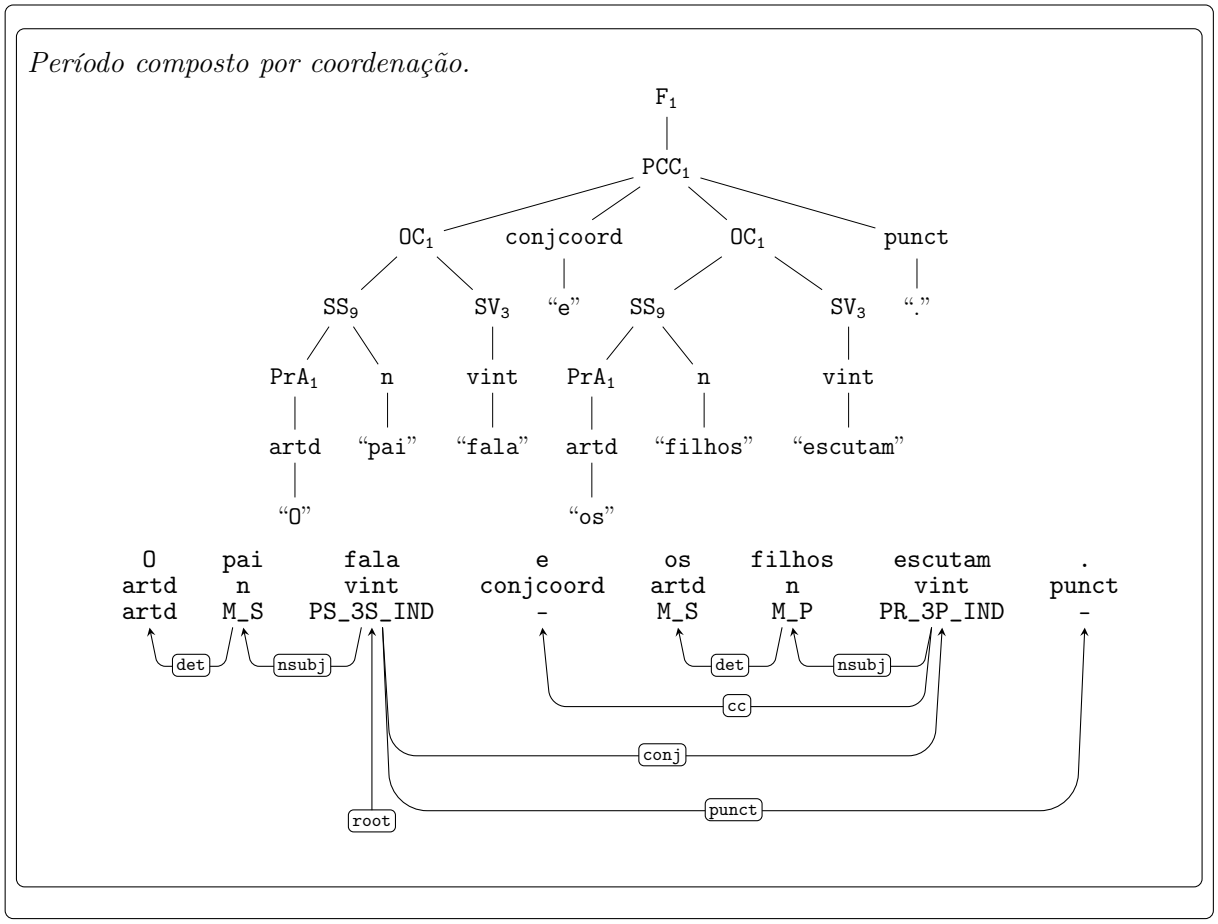
Fonte: autor.

Figura A.41: Exemplo de processamento - 9.



Fonte: autor.

Figura A.42: Exemplo de processamento - 10.



Fonte: autor.