

Este exemplar corresponde à redação final da
Tese/Dissertação devidamente corrigida e defendida
por: Ivan Santa Maria Filho
e aprovada pela Banca Examinadora.
Campinas, 30 de junho de 00
Adão de Paula
COORDENADOR DE POS-GRADUAÇÃO
CPG-IC

UNICAMP
BIBLIOTECA CENTRAL
SEÇÃO CIRCULANTE

PLIM — Processamento de Língua Natural
utilizando o Paradigma Multi-Agentes

Ivan Santa Maria Filho

Dissertação de Mestrado

PLIM — Processamento de Língua Natural utilizando o Paradigma Multi-Agentes

Ivan Santa Maria Filho¹

Abril de 2000

Banca Examinadora:

- Prof. Dra. Ariadne Maria Britto Rizzoni Carvalho (IC-UNICAMP) (Orientadora)
- Prof. Dr. Jacques Wainer (IC-UNICAMP)
- Prof. Dr. Jaime Simão Sichman (USP)
- Prof. Dr. Tomasz Kowaltowski (IC-UNICAMP)

¹Apoio financeiro do Conselho Nacional de Desenvolvimento Científico e Tecnológico — CNPq — entre março de 1998 e fevereiro de 1999 (processo 131962/98-1). Apoio financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo — FAPESP (processo 98/00605-8) — entre março de 1999 e agosto de 2000.

2006.16.242

IDADE	80
CHAMADA:	UNICAMP
	M337p
Ex.	
MB0 BC/	42930
OC.	461278100
C	☐
D	☒
ECO	R\$11,00
TA	25/10/00
CPD	

CM-00145918-B

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Maria Filho, Ivan Santa

M337p PLIM – Processamento de língua natural utilizando o paradigma
multi-agentes / Ivan Santa Maria Filho – Campinas, [S.P.:s.n.], 2000.

Orientador: Ariadne Maria Brito Rizzoni Carvalho

Dissertação (mestrado) – Universidade Estadual de Campinas,
Instituto de Computação.

1. Processamento de linguagem natural (Computação). 2.
Inteligência artificial distribuída. I. Carvalho, Ariadne Maria Brito
Rizzoni. II. Universidade Estadual de Campinas. Instituto de
Computação. III. Título.

PLIM — Processamento de Língua Natural utilizando o Paradigma Multi-Agentes

Este exemplar corresponde à redação final da
Dissertação devidamente corrigida e defendida
por Ivan Santa Maria Filho e aprovada pela
Banca Examinadora.

Campinas, 22 de abril de 2000.

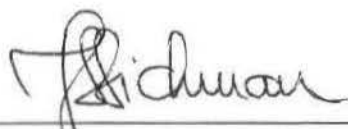


Prof. Dra. Ariadne Maria Britto Rizzoni
Carvalho (IC-UNICAMP) (Orientadora)

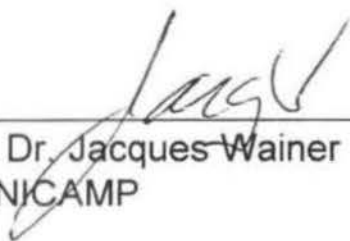
Dissertação apresentada ao Instituto de Com-
putação, UNICAMP, como requisito parcial para
a obtenção do título de Mestre em Ciência da
Computação.

TERMO DE APROVAÇÃO

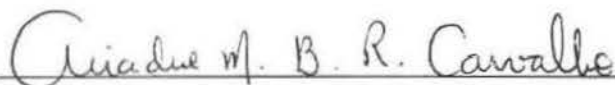
Tese defendida e aprovada em 30 de junho de 2000, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Jaime Simão Sichman
POLI -USP



Prof. Dr. Jacques Wainer
IC-UNICAMP



Profa. Dra. Ariadne Maria Brito Rizzoni Carvalho
IC-UNICAMP

Resumo

O processamento automático da língua natural é um problema multidisciplinar que envolve aspectos computacionais e lingüísticos, dentre outros. Neste trabalho analisamos algumas das teorias lingüísticas centradas na sintaxe que já foram utilizadas com sucesso em sistemas computacionais. A análise destas teorias e dos sistemas que as implementam considera a adequação lingüística e a complexidade computacional associada. Também são analisados os paradigmas de programação utilizados na implementação destes sistemas, identificando suas limitações. Apresentamos também um sistema distribuído baseado em multi-agentes que é capaz de se utilizar dos mecanismos lingüísticos tradicionais ao mesmo tempo em que supera parte das limitações dos sistemas que os implementam.

Abstract

Natural language processing is a multidisciplinary problem that covers, but is not limited to, Computer Science and Linguistics. This master thesis presents a survey on some of the syntactic-centered linguistic theories that have been successfully used in computer systems. An analysis of the linguistic adequacy and the computational complexity of each formalism is also presented. An analysis of the programming paradigms which were used to implement these systems and their limitations is also shown. Finally a distributed system based on the multi-agent paradigm is presented. This system is able to mimic many of the traditional system features while overcoming some of their limitations.

Agradecimentos

Ao CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico, processo 131962/98-1, e à FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo, processo 98/00605-8, pelo apoio recebido.

Gostaria de agradecer a minha orientadora por ter acreditado neste trabalho e por seu empenho, sem o qual o resultado final seria inevitavelmente pior. Aos meus amigos e colegas pelo apoio, companheirismo e compreensão durante os últimos dois anos. Em especial à turma de compiladores, com a qual passei muitas horas nervosas e divertidas no laboratório. Paulo “gaúcho” Ferreira Jr., Leonardo “tico migraña” Costa, Sandro Rigo, Bruno Kraychete e Marcelo “tuiuiu” Cintra.

Duas pessoas se destacam quando penso em como escolhi meu caminho. Élemer Surányi, meu grande amigo, que me apresentou aos computadores e sempre me incentivou com seu apoio e, especialmente, com seu exemplo. A outra, a quem tenho a honra de ter como amiga, é a Profa. Vera Lúcia Strube de Lima, minha mentora e principal responsável por eu ter escolhido a pesquisa acadêmica. Gostaria também de agradecer aos meus pais pelo conjunto da obra. Agora vocês já sabem a quem culpar :-).

Por fim eu gostaria de agradecer e dedicar este trabalho à pessoa mais importante na minha vida, Rachel Moreira Cunha, minha esposa, que sempre me apoiou e amou, mesmo nos momentos mais difíceis e nas piores incertezas. Pelo menos uma pessoa acreditou em mim o tempo todo, e não fui eu.

Trabalhos individuais não existem.

IVAN SANTA MARIA FILHO

Sumário

Resumo	v
Abstract	vi
Agradecimentos	vii
1 Introdução	1
1.1 Uma breve introdução ao PLN	3
1.2 Problemas intrínsecos ao PLN	4
1.3 Contornando dificuldades	6
2 Formalismos Lingüísticos	7
2.1 Sistemas de reescrita	8
2.1.1 Definição formal	8
2.1.2 Concatenação e fechamento	9
2.1.3 Língua	10
2.1.4 Mecânica	10
2.1.5 Hierarquia de gramáticas	14
2.2 A necessidade de formalismos específicos para o PLN	15
2.3 Formalismos para o PLN	17
2.3.1 Gramáticas de adjunção de árvores (TAGs)	17
2.3.2 Uma abordagem centrada na lingüística	39
2.3.3 Introdução à Government and Binding	39
2.3.4 Teoria Transformacional	41
2.3.5 Sintaxe X-Barra	42
2.3.6 A evolução do conceito de movimentação	43
2.3.7 Restrições de movimentação	45
2.3.8 Extraposition Grammars — XG	51
2.3.9 Bound-Movement Grammars — BMGs	63

3	Mecanismos de implementação	66
3.1	Sistemas tradicionais	67
3.2	Sistemas orientados a objetos	68
3.3	Arcabouços	71
3.4	Sistemas Multi-Agentes	73
3.4.1	Definição de agente	75
3.4.2	Tipos de agentes	76
3.4.3	Definindo um sistema multi-agentes	77
3.4.4	Tipos de decomposição	78
3.4.5	Uma possível formalização	79
3.5	Dois estudos de caso	81
3.5.1	Um Sistema Multi-Agentes para Processamento Distribuído de Lin- guagem Natural	81
3.5.2	Utilização do Paradigma Multi-Agentes no Processamento da Lin- guagem Natural: um Modelo Voltado à Resolução da Ambigüidade Léxica Categorical na Língua Portuguesa	87
4	Uma proposta de arquitetura e sua implementação	92
4.1	Motivação — Balanceando distribuição e simplicidade	93
4.2	Uma visão geral do modelo	95
4.3	Em busca de um formalismo gramatical simples e efetivo	98
4.3.1	Conseqüências das modificações para o novo formalismo	102
4.3.2	A linguagem de troca do analisador segmentado	104
4.3.3	O texto de entrada e os traços morfo-sintáticos	104
4.3.4	O processo de unificação	106
4.3.5	O operador “...” como um inversor de sentido	108
4.3.6	As agendas dos segmentos do analisador e alguns exemplos simples de derivação	109
4.3.7	Sumarizando o formalismo lingüístico	113
4.4	A definição da sociedade de agentes	115
4.5	Problemas conhecidos	121
5	Conclusões e a continuidade do trabalho	123
6	Apêndice A — Exemplos de derivação	128
6.1	Exemplos triviais	128
6.1.1	Uma derivação direta	129
6.1.2	Usando a adjunção	133
6.2	Reconhecendo referências cruzadas	136

6.3	Provendo recursão	138
6.4	Exemplos lingüísticos	139
6.5	Comentários sobre a paralelização	144

Bibliografia	146
---------------------	------------

Lista de Figuras

2.1	Representação em forma de árvore da derivação de <i>aabbba</i>	13
2.2	Duas possíveis descrições estruturais para “João viu Maria com o telescópio”.	16
2.3	Esboço esquemático das árvores iniciais e auxiliares.	18
2.4	Adjunção: representação esquemática e um exemplo.	19
2.5	Substituição esquemática e um exemplo.	20
2.6	Gramática TAG $G_{(a)}$ equivalente a G'	21
2.7	Seleção de α , β e γ	21
2.8	γ_1 : Adjunção de α_1 em β_3 sobre T	22
2.9	γ_2 : adjunção de β_4 em γ_1 sobre S	22
2.10	Uma TAG sem gramática livre de contexto fortemente equivalente.	23
2.11	Derivação em uma TAG sem gramática livre de contexto equivalente.	23
2.12	TAG $G_{(c)}$ sem gramática livre de contexto equivalente.	24
2.13	TAG G com restrições locais na adjunção.	25
2.14	TAG G com nodos B e Q ligados por um <i>link</i>	27
2.15	TAG G da figura 2.10 com <i>links</i>	27
2.16	Dependências cruzadas e encaixadas durante uma derivação.	28
2.17	Árvores iniciais que representam a sentença mínima de um verbo transitivo direto e de um verbo intransitivo.	29
2.18	Uma árvore inicial e uma árvore auxiliar representando uma das estruturas ‘qu’.	29
2.19	Instanciação de α_1	30
2.20	Derivação com itens instanciados.	30
2.21	Árvores elementares que complementam G	30
2.22	Controle de localidade estendida numa sentença ‘qu’.	32
2.23	Comportamento dos traços durante uma adjunção.	32
2.24	Comportamento dos traços durante uma substituição.	33
2.25	Parte de uma árvore inicial com <i>features</i> à mostra.	33
2.26	Árvore auxiliar com <i>features</i>	34
2.27	Derivação de “João tem comido banana” com unificação correta.	34

2.28	Possível família de árvores de verbos que pedem apenas um argumento. . . .	36
2.29	Organização do conhecimento necessário para instanciar uma língua natu- ral segundo GB.	40
2.30	Esquema das regras transformacionais.	42
2.31	Substituição com coindexação.	44
2.32	Movimentação de SN sobre estrutura original.	45
2.33	Dois tipos de adjunção.	46
2.34	Exemplo de movimentação 'qu'.	47
2.35	Restrição para sintagmas nominais complexos.	47
2.36	Restrição de termos coordenados.	48
2.37	Esquematização do "pied-piping".	48
2.38	Exemplo de Pied-Piping.	49
2.39	Restrição sobre projeção nominal mais à esquerda.	49
2.40	Estrutura que domina N^2 na restrição sujeito-frase	50
2.41	Exemplo de restrição sujeito-frase	50
2.42	Uma XG para cláusulas relativas simples.	52
2.43	Termos compostos vistos como árvores	53
2.44	Aplicação de regra XG para cláusulas relativas simples	54
2.45	"Ilhagem" de componentes por bloqueio	57
2.46	Aninhamento de derivações em uma XG	58
2.47	Possível árvore de derivação BMG para a sentença "o livro que a uma amiga João deu desapareceu".	64
4.1	Organização geral de um agente.	97
4.2	Árvore inicial do verbo comer.	99
4.3	A expansão de uma árvore inicial.	100
4.4	Nodos opcionais.	103
4.5	O seletor do nodo pé de uma árvore auxiliar é sempre descartado.	104
4.6	A linearização de uma árvore de derivação.	104
4.7	Uma árvore anotada e seu formato linearizado.	106
4.8	Um exemplo de codificação por traços.	107
4.9	A remoção da ocorrência mais recente do operador "...".	109
4.10	Conteúdo das agendas do primeiro e segundo segmentos do analisador. . . .	110
4.11	A expansão de um seletor de adjunção "_SA".	111
4.12	A expansão de uma solicitação complexa.	113
4.13	Uma ligação explícita e seu equivalente anotado.	114
4.14	Separando constituintes de uma derivação complexa.	115
4.15	Ilustrando o funcionamento da sociedade de agentes.	116

Capítulo 1

Introdução

“Nenhuma tecnologia suficientemente avançada
pode ser diferenciada de mágica.”

anônimo

“Computadores são inúteis.
Eles só podem te dar respostas.”

Pablo Picasso

Neste trabalho buscamos investigar a possibilidade de utilização dos sistemas multi-agentes no tratamento automático do português. Apesar das pesquisas recentes, notadamente através do projeto NALAMAS¹, a utilização de sistemas multi-agentes no processamento do português tem possibilidades ainda não investigadas. Este trabalho pretende demonstrar a adequação do paradigma de multi-agentes para o Processamento de Língua Natural (PLN), apresentando-o como uma evolução natural dos esforços aplicados à área.

A adequação do paradigma de programação, no entanto, não pode ser demonstrada sem a utilização de um formalismo lingüístico. Foram selecionados inicialmente três formalismos centrados na sintaxe: as Tree Adjoining Grammars (Gramáticas de Adjunção de Árvores ou TAGs), as Extraposition Grammars (Gramáticas de Extraposição ou XGs) e as Bound-Movement Grammars (Gramáticas de Movimentação e Ancoragem, ou BMGs). Estes formalismos, apesar de já terem sido utilizados com sucesso no tratamento do português, não foram suficientemente adaptados para o paradigma de multi-agentes. As dificuldades encontradas durante esta adaptação nos levaram a definir um formalismo híbrido simplificado a partir das TAGs, mas com características agregadas das XGs e

¹PROTEM-CC 680081/95-0. <http://www.inf.pucrs.br/links/grupos/nalamas/index.html>.

BMGs. Este novo formalismo não deve ser visto como uma tentativa de superar a capacidade de reconhecimento ou de descrição de nenhum dos três formalismos estudados, mas como uma tentativa de preservar ao máximo suas habilidades ao mesmo tempo em que facilita a distribuição da análise sintática em um ambiente cooperativo.

Este trabalho conta, ainda neste capítulo, com uma breve introdução ao PLN. O capítulo 2 contém um breve histórico do surgimento das linguagens artificiais e mostra sua inadequação ao PLN. Os formalismos lingüísticos selecionados (TAG, XG e BMG) são descritos, assim como os mecanismos computacionais que originalmente os implementavam. Para demonstrar que expandir ou simplificar um formalismo para fins específicos é um procedimento comum, também apresentamos algumas variantes destes formalismos. Para demonstrar como uma destas variantes pode ser implementada utilizando mecanismos diferentes dos originais também nos referimos a alguns trabalhos correlatos.

O capítulo 3 apresenta o paradigma multi-agentes através de uma abordagem evolutiva, descrevendo as arquiteturas que vêm sendo utilizadas no PLN no decorrer dos anos, dos sistemas tradicionais aos arcabouços², passando pela orientação a objetos. Também são descritas algumas tentativas de reaproveitamento tanto do conhecimento lingüístico quanto computacional, assim como as dificuldades de integração, manutenção e expansão encontradas nas diversas abordagens. Ainda neste mesmo capítulo são apresentados dois estudos de caso de sistemas multi-agentes desenvolvidos para o processamento do português e, ainda, uma técnica para descrever formalmente um sistema multi-agentes.

O capítulo 4 descreve o formalismo híbrido definido e apresenta o modelo de agente para o qual foi desenvolvido. É importante frisar que a definição de um formalismo original não é gratuita. Ela foi necessária porque o modelo de agente e sua capacidade de comunicação foram definidos antes da seleção do modelo lingüístico, o que impediu a utilização de algumas características dos formalismos estudados e abriu novas possibilidades de processamento. Para testar o comportamento do formalismo e do modelo de agente propostos, foi construído um protótipo em ANSI C++ 2.0 capaz de analisar todos os exemplos utilizados. A construção de uma gramática ampla, no entanto, não pode ser concluída e, por isso, fornecemos um apêndice contendo exemplos de derivação e de tratamento de alguns fenômenos lingüísticos para ilustrar o funcionamento do analisador.

Note que ao definirmos primeiro o mecanismo computacional desejado para somente depois decidir qual o formalismo lingüístico adequado, estamos subvertendo a ordem tradicional de desenvolvimento. De uma forma geral os sistemas computacionais são desenvolvidos para acompanhar uma definição lingüística e não o contrário. O capítulo 5 traz esta e as demais conclusões deste trabalho, além da descrição dos trabalhos futuros relacionados com o modelo.

²Do inglês *frameworks*.

1.1 Uma breve introdução ao PLN

Uma língua dita natural (ao contrário das artificiais ou formais) é aquela que já existe, e que supre várias funções em nossa interação com outras pessoas [44], tendo surgido antes de sua representação formal. Desta forma os modelos lingüísticos tentam explicar um fenômeno natural, ao contrário do que ocorre com as linguagens de programação onde, por exemplo, a estrutura é definida antes de sua utilização, sem espaços para ambigüidades.

Não é recente a preocupação do homem em processar, através de máquinas, sua língua natural de comunicação. Desde o século XIX lingüistas vêm trabalhando o que se poderia chamar de “o enfoque matemático” a ser aplicado à língua, visando a aproximação de modelos que reflitam o comportamento desta. No início do presente século, trabalhos referentes à elaboração de modelos matemáticos formalizando a linguagem começaram a ser conduzidos, com interesse em alguma forma de processamento automático [27].

No início da guerra fria surgiu a primeira aplicação computacional que tratava da língua humana: sistemas de tradução automática para textos de física do russo para o inglês. Segundo [61] esta atividade marca a primeira das três etapas do que hoje se convencionou chamar de Processamento da Língua Natural (PLN). Essa etapa foi denominada pré-teórica e caracterizava-se pela aplicação de mecanismos rudimentares - baseados em dicionários - em problemas complexos, com resultados freqüentemente desapontadores. A etapa posterior, voltada para a experimentação de teorias lingüísticas, foi chamada de “experimental”; neste período surgiram protótipos restritos³ que aplicavam diferentes teorias lingüísticas — com ênfase na sintaxe, semântica ou lógica — para demonstrar princípios específicos da língua analisada ou da representação do conhecimento. Hoje, ainda segundo [61], entramos numa etapa empírica, avaliativa e centrada no usuário, onde as teorias e aplicações são julgadas de acordo com seu sucesso na solução de problemas — inclusive pelo mercado.

Ainda é provável que a tradução automática de textos de física russos atraia interesse, mas por motivos diferentes dos originais. Durante as últimas décadas surgiram novas aplicações, linhas de financiamento e incentivos à pesquisa na área. A proliferação dos computadores nas mais diversas atividades humanas — e suas decorrências — não apenas manteve, como aumentou o interesse em PLN. O uso da língua humana parece ser a melhor maneira de tornar a comunicação entre homem e computador mais ampla [56] e é um passo adequado, depois da utilização das interfaces gráficas [22], isto porque o ser humano é predominantemente visual e verbal [45].

Também é economicamente interessante aproximar a máquina do usuário. Além da redução dos custos de treinamento, aplicações como [7]:

- sistemas de consulta, onde a língua natural é utilizada para obter informações de

³Do inglês *toy systems*.

um banco de dados, incluindo busca de documentos sobre um determinado tema;

- identificação de tópicos e sumarização de documentos;
- tradução automática;
- sistemas de atendimento automático ao cliente via telefone, tais como transações bancárias ou compras em catálogo;
- ensino assistido por computador, onde a máquina interage com o estudante verbalmente ou em textos com formato livre;
- dispositivos para controle através da voz (ajustar o relógio do vídeo-cassete, por exemplo); e
- sistemas cooperativos de resolução de problemas como em um sistema que ajude a planejar e agendar envio de encomendas.

permitem que novos mercados sejam explorados ou que tenham seus custos reduzidos.

A necessidade de programas de computador para PLN também vem aumentando nas últimas décadas pelo efeito conjugado do desenvolvimento da informática e da necessidade de tratar volumes cada vez maiores de informação escritas e faladas. Precisamos também de programas velozes e eficazes para compor, classificar, armazenar, analisar, questionar e traduzir os documentos já produzidos [27]. Os setores econômicos [38] clamam por ganhos de produtividade e a língua é um elemento primordial da produção industrial. Uma central nuclear, por exemplo, representa 100.000 páginas de documentação técnica. A língua também carrega um elemento político, ideológico e cultural: “as línguas que não se industrializarem deixarão de ser veiculares, de ser línguas de civilização”⁴.

1.2 Problemas intrínsecos ao PLN

A natureza da língua, entretanto, não é simples⁵ e a construção de sistemas para seu tratamento automatizado enfrenta tanto problemas de natureza lingüística quanto computacional. O leitor que possui conhecimento técnico em computação pode tentar relacionar o PLN com o processamento das linguagens artificiais de computador como PASCAL ou C++. Esta associação, apesar de intuitiva, não é adequada por uma série de fatores [10]:

⁴Declaração do secretário geral do Conselho da Europa, 1995 [27].

⁵Nos anos 60 era comum acreditar que a tradução de textos de forma automática era uma questão relativamente simples quando comparada com a construção de um computador capaz de derrotar um humano campeão mundial de xadrez.

- as línguas naturais lidam com ambigüidade. A sentença “A bela casa na fazenda”, por exemplo, tem dois significados mais comuns. Pode identificar uma construção bonita no campo ou uma moça bonita casando no campo. Um analisador com uma gramática suficientemente ampla pode ainda considerar que a sentença signifique que há uma bela estampa de casa num tecido ou, mais forçosamente, que alguém costurou uma bela casa de botão num tecido ou, ainda, que temos um desenho de uma moça bonita casando estampado num tecido [43];
- a estrutura das sentenças, nas línguas naturais, normalmente é muito mais complexa do que nas artificiais como, por exemplo, em “o cão que dormia quando Maria resolveu sair de casa ao nascer do dia parecia cansado, entretanto nada se poderia afirmar uma vez que aquele cão ali estirado estava apenas dormindo” tem uma estrutura que dificilmente é encontrada nas linguagens artificiais;
- parece não existir uma maneira simples e universal de representar o significado de sentenças de uma linguagem natural [63] [10] [7] como, por exemplo, o da sentença “o vinho se dá na paciência dos cachos”;
- estrutura e significado estão necessariamente interligados nas linguagens naturais, enquanto que nas artificiais freqüentemente podem estar separados.

O objetivo do estudo do tratamento automático das línguas é a concepção de lógicas (programas) capazes de tratar de maneira automática os domínios lingüísticos desta língua [38]. Estes domínios podem delimitar parcelas de uma língua, como os textos escritos ou uma área de conhecimento com vocabulário e gramáticas limitados, e estar divididos em diferentes níveis de tratamento. Uma possível divisão em níveis pode ser [7] [61]:

- Acústico/prosódico;
- fonológico;
- morfológico;
- léxico;
- sintático;
- semântico;
- discurso;
- pragmática;

A segmentação em níveis é tradicionalmente aceita pelos lingüistas e estes níveis são considerados por alguns como independentes tanto em relação à estrutura quanto à organização e à função [39], permitindo o tratamento isolado de suas características e problemas.

1.3 Contornando dificuldades

A divisão da língua em níveis, com uma organização seqüencial de análise, está refletida em muitos dos sistemas de processamento automático [83] que assumem os resultados de uma etapa de processamento como a entrada da próxima. Este enfoque seqüencial pode ser observado no projeto LEXIS⁶, cujo objetivo era o desenvolvimento de ferramentas para o tratamento automatizado da língua portuguesa⁷.

A abordagem linear, no entanto, vem sendo questionada [39] [19] [83] [84] [26] tanto porque a solução de alguns fenômenos lingüísticos envolve mais de um nível lingüístico, quanto em razão da limitação computacional de uma arquitetura seqüencial. A interdependência dos níveis lingüísticos, com a conseqüente necessidade de troca de informação entre eles, motivou a busca por alternativas em trabalhos que analisam fenômenos específicos, tais como as anáforas [66], elipses [60] e ambigüidade categorial [21], assim como também nos trabalhos mais genéricos [84] [24].

A arquitetura seqüencial também afeta o custo e o período de desenvolvimento dos sistemas [94] [20] e dificultam o retrocesso⁸ abrindo a possibilidade de se perder informações. Sua concepção fechada e suas estruturas de controle exigem conhecimento centralizado e grandes quantidades de recursos computacionais [22].

Para contornar estas limitações arquiteturais surgiram técnicas que envolvem, por exemplo, a utilização de dados empíricos (estocásticos) [15], gramáticas lexicalizadas [77], arcabouços⁹ lingüísticos [94] e combinações entre as abordagens.

Outra alternativa ao enfoque tradicional são os sistemas multi-agentes. Nestes sistemas a tarefa de manipular uma língua natural é executada com base no comportamento individual dos componentes do sistema e na estratégia de comunicação entre estes componentes [19]. Este enfoque é adequado para o tratamento de problemas grandes e complexos que requerem conhecimentos de muitos domínios diferentes, pois favorece o controle da complexidade e da evolução do sistema [81].

⁶<http://www.inf.pucrs.br/links/grupos/lexis/homepage.htm>. Instituto de Informática da PUCRS.

⁷Foram desenvolvidos diversos sistemas neste projeto, mas os elementos centrais são um léxico com informações morfo-sintáticas e um analisador sintático baseado nas Tree Adjoining Grammars [47].

⁸Do inglês *backtracking*. Significa voltar a uma etapa anterior do processamento quando da identificação de um equívoco.

⁹Do inglês *frameworks*.

Capítulo 2

Formalismos Lingüísticos

"A formal parsing algorithm should
not always be used."

D. Gries

"An interpretation 'I' satisfies a sentence in the table language
if and only if each entry in the table designates the value of
the function designated by the function constant in the
upper-left corner applied to the objects designated
by the corresponding row and column labels."

Genesereth & Nilsson

"Logical Foundations of
Artificial Intelligence"

Quando pensamos em reconhecer as estruturas de uma língua natural, especialmente a sintaxe, é natural buscar auxílio nas gramáticas e dicionários tradicionais. No entanto, os fatores que tornam uma determinada gramática adequada não são os mesmos para o leitor humano e para o computador. Em um ambiente computacional alguns dos fatores determinantes são a generalidade, a seletividade e a facilidade de compreensão [7].

Como generalidade entendemos a quantidade de sentenças que uma gramática reconhece. As gramáticas tradicionais são especialmente competentes neste aspecto pelo contínuo refinamento que vêm sofrendo durante os anos. Isto não impede que seus mecanismos careçam de elegância que, apesar de ser um sintoma por demais difuso para ser considerado decisivo, é um bom indicativo de problemas durante uma adaptação para o ambiente computacional.

Já a seletividade é a capacidade de reconhecimento de sentenças inválidas. Este problema é freqüentemente tratado através da enumeração de exceções que, apesar de per-

mitir uma maior liberdade normativa, insere novos fatores de análise que muitas vezes sobrecarregam a descrição de uma gramática e dificultam sua compreensão.

A facilidade de compreensão é exatamente o terceiro fator desejado numa gramática em ambiente computacional. Ela é especialmente importante porque a construção de um programa de computador lingüístico é uma tarefa complexa, demorada, cara e propensa a erros [94].

2.1 Sistemas de reescrita

O interesse, tanto econômico quanto acadêmico, pelos sistemas de computação em conjunto com o desenvolvimento de teorias que modelavam matematicamente as línguas permitiu o surgimento de novas formas de programação, mais próximas da comunicação humana que do mecanismo interno da máquina.

Os modelos matemáticos foram utilizados para o mapeamento de estruturas sintáticas e permitiram a construção de línguas artificiais, mais conhecidas como linguagens de programação. Estas linguagens possuem uma estrutura mais simples do que a das línguas naturais e, ao contrário destas, não devem permitir construções ambíguas. Esta precisão descritiva provém dos esforços dos desenvolvedores e do fato que, quando geramos algo a partir de um modelo, o objeto gerado se encaixa perfeitamente neste modelo. Este encaixe perfeito não ocorre com as línguas naturais porque aqui estamos tentando explicar um fenômeno anterior ao próprio modelo.

As linguagens artificiais, assim como os modelos lingüístico e matemático que as embasam, foram aprimorados ao longo do tempo. As linguagens foram categorizadas de acordo com sua generalidade, seletividade e, conseqüentemente, complexidade de sua descrição. Para compreendermos esta classificação, suas limitações e as expansões propostas, é importante definirmos formalmente os aspectos envolvidos no problema.

2.1.1 Definição formal

A notação matemática é adequada para gerar descrições e definições livres de ambigüidade e por isso será utilizada em grande parte deste trabalho. No entanto a compreensão de uma definição matemática não é natural para a maioria dos leitores e por isso serão dadas explicações informais, assim como exemplos práticos.

Quando nos deparamos com um problema qualquer é adequado procurar a unidade básica que o compõe. Quando construímos um muro podemos utilizar tijolos e podemos tratar o próprio muro como uma coleção adequadamente arrumada de tijolos e cimento. Quando falamos de uma língua escrita nossos tijolos podem ser as palavras e nossa

gramática algumas das regras de arrumação destas palavras¹.

Assim como em um muro podemos ter diferentes tipos de tijolos, numa língua podemos ter palavras diferentes. Não existe um nome adequado para a enumeração de tijolos, mas a enumeração de palavras é comumente chamada de onomástico. Como estamos trabalhando com uma simplificação onde as palavras são entidades atômicas, podemos substituí-las por letras e trocar nosso onomástico por um alfabeto.

Quando construímos um muro queremos que ele fique sólido e por isso usamos palanques de concreto para a estrutura. Quando queremos nos referir a uma parede não dizemos “aquele retângulo de tijolos com dez de largura e trinta de altura colocados horizontalmente” ou o que valha; nós simplesmente dizemos “aquela parede” para que a explicação fique mais simples e concisa. Quando escrevemos uma gramática para ambiente computacional também queremos solidez (generalidade e seletividade) mas principalmente queremos que ela nos traga informações suficientemente detalhadas do que descreve, ou seja, que seja clara e precisa sem que se perca em detalhes. Para isto separamos o nosso alfabeto em dois grupos, um de palavras² (nossos tijolos) e outro de símbolos especiais usados para facilitar a descrição da língua desejada (nossas paredes e palanques). As palavras desta gramática serão chamadas de símbolos terminais e as estruturas especiais serão chamadas de símbolos não terminais. O porquê desta nomenclatura ficará mais claro adiante.

Um complicante na nossa analogia com os muros é o símbolo vazio. Ele corresponde aproximadamente a um elemento vazado no muro, um espaço onde poderia existir um tijolo mas que o engenheiro optou por manter vazio para entrar ar ou luz. Na nossa gramática esse “tijolo” especial será chamado de símbolo vazio e sua representação gráfica será a letra grega épsilon (ϵ). Quando temos um conjunto Σ composto por letras mais ϵ , temos que $a\epsilon = \epsilon a = a$, qualquer que seja $a \in \Sigma$.

Neste texto as enumerações de símbolos serão notadas por letras gregas maiúsculas, os símbolos não terminais por letras maiúsculas de nosso alfabeto e os símbolos terminais serão notados por letras minúsculas; quando não quisermos diferenciar símbolos terminais de não terminais usaremos letras gregas minúsculas. Uma língua, isto é, nossa coleção arrumada de palavras, será representada por letras maiúsculas ocidentais — geralmente L .

2.1.2 Concatenação e fechamento

Quando temos um conjunto de símbolos Σ , definimos a operação de concatenação como a justaposição de dois ou mais elementos deste conjunto. Sendo $a, b \in \Sigma$, a concatenação de a e b será ab ; correlatamente a concatenação de a , a e b será aab .

¹ Ao considerarmos a palavra como unidade básica estamos assumindo, em prol da clareza, uma simplificação do problema.

² Usado como sinônimo de letra neste escopo.

Esta definição intuitiva — e suspeitosamente desnecessária — permite que definamos o fechamento de Kleene de um conjunto, que no caso de Σ é notado como Σ^* . O fechamento de Kleene é a união de todas as concatenações possíveis de todos os elementos de Σ , inclusive ϵ . Isto quer dizer que todas as seqüências possíveis de elementos de Σ , não importa quantas vezes estejam repetidos, pertencem a Σ^* , inclusive a sentença vazia formada apenas por ϵ .

2.1.3 Língua

Agora que temos uma explicação preliminar, podemos definir formalmente uma língua como um conjunto de palavras sobre um alfabeto finito. Para qualquer conjunto finito Σ de símbolos (Σ é um alfabeto), temos que $\Sigma^* = \{\omega_1 \dots \omega_n | n \geq 1, \text{ para cada } \omega_i \in \Sigma\} \cup \{\epsilon\}$, onde ϵ identifica uma sentença vazia, ou seja, uma sentença de comprimento zero. Um conjunto L é uma língua se e somente se (sse) existe um alfabeto finito Σ_L de forma que $L \subseteq \Sigma_L^*$. Desta forma uma língua é um conjunto de sentenças de comprimento finito que pode ser expresso pela concatenação de um número finito de palavras [12].

Mais informalmente, isto significa que as línguas são conjuntos de sentenças, uma sentença é uma seqüência de palavras e um alfabeto é o conjunto de palavras válidas. A linguagem de programação Pascal, por exemplo, é o conjunto de todas as seqüências de palavras que constituem programas válidos nesta linguagem; a linguagem dos números primos é o conjunto de todas as seqüências de dígitos decimais que representam números primos; a linguagem das palavras reservadas em C^{++} é o conjunto de seqüências de símbolos que só podem ser utilizadas em situações especiais. Nos dois primeiros casos os conjuntos são infinitos, enquanto que no último é finito. Para este estudo o que importa é que, quando falamos das línguas desta forma, não associamos significados às seqüências de símbolos, apenas as classificamos como pertencentes ou não a uma dada linguagem de acordo com uma arrumação previamente definida [9].

Esta definição é suficientemente ampla para abranger tanto as línguas naturais como as linguagens de programação [12], mas carece da definição de um mecanismo de geração e/ou reconhecimento automático ou, pela definição adotada, um mecanismo de concatenação de palavras do alfabeto. Este mecanismo, voltando ao exemplo do muro, seria uma máquina capaz de cimentar corretamente os tijolos e palanques e/ou reconhecer como foi montada a estrutura de um muro pronto, dizendo se ele está corretamente construído ou não.

2.1.4 Mecânica

O mecanismo abordado neste trabalho é o dos sistemas de reescrita, cuja definição é utilizada para auxiliar na demonstração de algumas propriedades das línguas. O sistema de reescrita é capaz de identificar por meio de regras onde uma estrutura pode ser trocada

por outra equivalente. Uma regra para nosso muro de tijolos poderia ser um bilhete para o mestre de obras onde dizemos que ele pode trocar grupos de quatro tijolos pequenos sem furos por um outro tijolo maior, desde que com tamanho e propriedades equivalentes às do conjunto trocado.

No caso das línguas um sistema de reescrita é um par (Σ, P) , onde Σ é o nosso *alfabeto* e P é um conjunto ordenado de pares sobre Σ [74]. Os elementos $(\rho, \gamma) \in P$ formam uma *produção* ou regra de reescrita geralmente notada como $\rho \rightarrow \gamma$ ³ para indicar que ρ pode ser reescrito (substituído) por γ através da aplicação de $\rho \rightarrow \gamma$.

Derivação

Para decidirmos se uma determinada sentença pertence a uma dada língua efetuamos um processo denominado derivação. Uma derivação é a aplicação consecutiva de regras de reescrita até que consigamos reconhecer a sentença ou que tenhamos informação suficiente para decidir que a sentença de entrada não pertence à linguagem.

A derivação depende diretamente do sistema de reescrita e por isso dizemos que, dado um sistema de reescrita, a relação de derivação $\Rightarrow$ sobre o conjunto Σ^* é definido como se segue: para quaisquer palavras ρ e γ , $\rho \Rightarrow \gamma$ se aplica sse existirem palavras $\rho_1, \rho_2, \alpha, \beta$ de forma que

$$\rho = \rho_1 \alpha \rho_2 \quad \gamma = \rho_1 \beta \rho_2$$

e $\rho \rightarrow \gamma$ é uma produção do sistema. O fechamento transitivo reflexivo da relação $\Rightarrow$ é notada como $\Rightarrow^*$ e $\rho \Rightarrow^* \gamma$ se aplica sse existirem $n \geq 1$ palavras $\rho_1, \dots, \rho_n$ tais que $\rho = \rho_1, \gamma = \rho_n$ e $\rho_i \Rightarrow \rho_{i+1}$ se aplica para todo $i = 1, \dots, n-1$ [74]. Uma forma estendida de notar a mesma regra de derivação $\rho \rightarrow \gamma$ é $\rho_1 \alpha \rho_2 \rightarrow \rho_1 \beta \rho_2$ que nos leva a generalização

$$\rho_1 \alpha_1 \dots \rho_n \alpha_n \rho_{n+1} \rightarrow \rho_1 \beta_1 \dots \rho_n \beta_n \rho_{n+1},$$

onde cada $\rho_i, \alpha_i, \beta_i \in (\Sigma_N \cup \Sigma_T)^*$.

O fechamento transitivo reflexivo é a capacidade do sistema aplicar as produções até obter um símbolo especial que indica reconhecimento da sentença ou, aplicar regras sobre este símbolo especial até transformá-lo na sentença de entrada.

Gramática

A existência de um símbolo especial que nos indica quando uma derivação termina deixa claro que é necessária uma quantidade maior de informação para que o sistema funcione. Estas informações estão contidas nas gramáticas. No caso do muro de tijolos ela conteria

³Algumas vezes vamos nos referir às partes de produção como “lado esquerdo” e “lado direito”. No caso de $\rho \rightarrow \gamma$ o lado esquerdo é ρ e o lado direito é γ .

as regras de como combiná-los para construir uma casa e de como reconhecer que a casa está pronta.

Uma gramática é uma quádrupla $G = (R, \Sigma_N, \Sigma_T, \Sigma_S)$ onde $R = (\Sigma, P)$ é um sistema de reescrita, Σ_N e Σ_T são alfabetos disjuntos tais que $\Sigma = \Sigma_N \cup \Sigma_T$. Já Σ_S é um subconjunto de Σ_N . Os elementos de Σ_N , Σ_T e Σ_S são mais conhecidos como *não terminais*, *terminais* e *símbolos iniciais*, respectivamente [74].

Mais informalmente podemos definir uma gramática como um conjunto de produções na forma *símbolo* $\rightarrow$ *símbolo símbolo ... símbolo* onde existem zero ou mais símbolos do lado direito. Cada símbolo pode ser um terminal, que está presente em Σ_T , ou um não terminal, que deve estar presente no lado esquerdo de alguma produção (e em Σ_N). Um terminal não pode aparecer sozinho no lado esquerdo de uma produção [9], ou seja, não pode ser derivado por si só e por isso mesmo se chama terminal.

Uma derivação pode partir de um símbolo da gramática e tentar derivar este símbolo até que a sentença de entrada seja encontrada, num processo chamado derivação descendente⁴. Também pode tentar agrupar palavras da sentença de entrada em estruturas maiores, isto é, substituir esta seqüência pelo lado esquerdo de alguma produção cujo lado direito seja exatamente esta seqüência, até atingir um símbolo especial, num processo denominado derivação ascendente⁵. Em ambos os casos o símbolo especial origem/destino deve pertencer ao conjunto de símbolos iniciais (Σ_S).

Derivação em uma gramática

Agora que sabemos qual o princípio básico de reconhecimento das sentenças de entrada e de como as regras que regem este processo estão organizadas, podemos definir como aplicar as regras de reescrita — produções — de uma gramática específica. Esta restrição sobre uma gramática específica vão nos permitir trabalhar com um conjunto de regras organizadas de forma a reconhecer ou produzir uma determinada língua.

Uma *derivação* em uma gramática G é qualquer seqüência finita de sentenças

$$\theta_0, \theta_1, \dots, \theta_n \in (\Sigma_N \cup \Sigma_T)^*,$$

onde para $i = 1, \dots, n$, existe uma derivação usando as regras desta gramática (notada como $\xRightarrow{G}$) $\theta_{i-1} \xRightarrow{G} \theta_i$. Neste caso também podemos adotar a notação mais sucinta $\theta_0 \xRightarrow{*G} \theta_n$, indicando que uma série de passos foi utilizada para chegarmos a θ_n a partir de θ_0 . A *linguagem gerada* por G é [12]:

$$L(G) = \{w \in \Sigma^* \mid \text{existe uma derivação } S \xRightarrow{G} \dots \xRightarrow{G} w \text{ em } G\}.$$

⁴Do inglês *top-down*.

⁵Do inglês *bottom-up*.

Suponha que desejamos reconhecer todas as sentenças da linguagem onde existam apenas as palavras a e b e que o número destas palavras numa mesma sentença seja igual. Sabendo quais as palavras de nossa língua já podemos definir o conjunto de terminais como $\Sigma_T = \{a, b\}$. Como símbolo inicial usaremos a letra S , de sentença; como não precisamos de nenhuma descrição mais detalhada definiremos o conjunto de não terminais como $\Sigma_N = \{S\}$.

Ainda nos faltam as produções da gramática, ou seja, seu sistema de reescrita. Usando uma lógica simplista consideraremos que as sentenças desta língua possuem sempre um a e um b correspondentes, não importando a ordem em que apareçam. Isto nos fornece as regras

$$S \rightarrow aSb, \quad S \rightarrow bSa \quad e \quad S \rightarrow \epsilon$$

mas ainda nos falta algo que permita a repetição das combinações, o que é fornecido pela regra

$$S \rightarrow SS.$$

É freqüentemente muito difícil provar que uma determinada gramática gera a língua desejada. Logicamente é um problema indecidível [74]. Vejamos como uma sequência de derivações nesta gramática satisfaz a sentença $aabbba$, que obviamente pertence à língua desejada:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow aaSbbS \Rightarrow aabbS \Rightarrow aabbbSa \Rightarrow aabbba.$$

Outra forma de representar este processo é através de uma árvore, como nos mostra a figura 2.1. Este processo de derivação também é conhecido como *análise sintática*⁶.

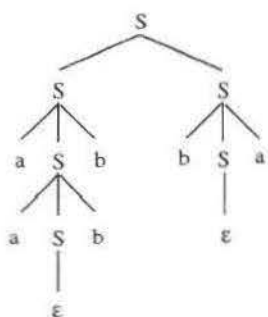


Figura 2.1: Representação em forma de árvore da derivação de $aabbba$.

Quanto mais complexa uma língua, mais complexa a gramática que a descreve. Quanto mais complexa a gramática, mais complicado fica interpretar suas regras de reescrita e mais difícil implementar um modelo computacional que faça esta interpretação automaticamente.

⁶Em inglês *parsing*.

2.1.5 Hierarquia de gramáticas

Para simplificarmos uma gramática podemos usar algumas restrições para o formato das regras de reescrita. Uma restrição comum é proibir que elementos de Σ_T sejam reescritos, forçando as regras a obedecer ao formato

$$w_1 X_1 \dots w_n X_n w_{n+1} \rightarrow w_1 \gamma_1 \dots w_n \gamma_n w_{n+1},$$

onde cada $w_i \in \Sigma_T$, cada $X_i \in \Sigma_N^*$ e cada $\gamma_i \in (\Sigma_N \cup \Sigma_T)^*$ [12]. Outra restrição comum é limitar Σ_S a um conjunto unitário [74].

A qualidade e quantidade de restrições aplicadas sobre as regras de derivação delimitam o poder de geração e/ou reconhecimento de uma gramática. Classicamente estas restrições permitem dividir as gramáticas, e conseqüentemente as línguas, em quatro categorias. As definições apresentadas a seguir foram sintetizadas a partir de [12] e [74].

A primeira categoria de gramática, conhecida como gramática de tipo 0, pode descrever as linguagens *recursivamente enumeráveis*. Estas gramáticas permitem qualquer tipo de produção, inclusive as que retiram símbolos terminais da sentença de entrada e definem o contexto onde podem ser aplicadas. Não existem restrições para as regras de produção. Note que nesta categoria uma sentença derivada não é necessariamente maior do que a sentença original porque o lado esquerdo da produção não é necessariamente menor ou igual ao lado direito.

A segunda classe de gramáticas, de tipo 1, é a das que podem descrever as *linguagens sensíveis ao contexto*. Nestas gramáticas as produções obedecem a forma $w_1 \alpha w_2 \rightarrow w_1 w w_2$, onde w_1 e w_2 são palavras arbitrárias sobre Σ , e $\alpha \in \Sigma_N$. Observe que a presença das palavras arbitrárias w_i permite especificar em que contexto uma produção pode ser utilizada, o que nos habilita a descrever linguagens com grande riqueza de detalhes.

A terceira classe de gramáticas, de tipo 2 ou *livre de contexto*, é a mais utilizada em linguagens de programação. Suas produções obedecem ao formato $\alpha \rightarrow w$, com $\alpha \in \Sigma_N$. Basicamente uma gramática desta classe permite que um não terminal seja substituído pelo lado direito de uma produção qualquer onde este não terminal seja o lado esquerdo, sem considerar os demais símbolos da sentença, ou seja, o contexto. Note que uma sentença derivada é sempre maior ou igual em número de palavras do que a sentença original, exceto quando se puder derivar um símbolo não terminal em ϵ .

A última e mais restrita categoria é a das gramáticas *regulares*, ou de tipo 3. As produções obedecem a forma $\alpha \rightarrow a\beta$, onde $\alpha, \beta \in \Sigma_N$ e $a \in \Sigma_T$.

Esta classificação, comumente denominada “hierarquia de Chomsky”, organiza as línguas em conjuntos e subconjuntos. Uma língua que pode ser descrita por uma gramática de nível 3 também pode ser descrita por uma gramática de nível 2, ou seja, as línguas de nível 3 são um subconjunto das de nível 2. Da mesma forma as de nível 2 formam um subconjunto das de nível 1 e estas um subconjunto das de nível 0.

Este poder de representação também se reflete aproximadamente nos mecanismos computacionais de reconhecimento. Um programa de computador capaz de manipular uma gramática de nível 3 também é capaz de manipular gramáticas de tipo 2; um programa para o tipo 2 é capaz de manipular as gramáticas de tipo 1; um para o tipo 1 é capaz de manipular as de tipo 0 e os programas para tipo 0 podem manipular as de todos os demais tipos. O contrário não é verdadeiro, ou seja, os programas que reconhecem linguagens de tipo 2 não possuem poder suficiente para reconhecer todas as linguagens de tipo 1, por exemplo.

Por outro lado, um programa mais complexo, capaz de manipular gramáticas mais complexas, dificilmente será mais eficiente que um mais simples, construído para os tipos mais básicos de gramática, mesmo quando ambos estiverem usando as mesmas descrições. Um analisador para gramáticas livres de contexto, por exemplo, será mais eficiente que um desenvolvido para gramáticas sensíveis ao contexto quando ambos estiverem usando uma gramática livre de contexto.

Como vimos, o reconhecimento de uma sentença como pertencente ou não à uma linguagem é determinado pela derivação das produções da gramática que a descreve. Existem diferentes maneiras de representar um processo de derivação mas, para nossos propósitos, é suficiente considerar uma árvore como aquela apresentada na figura 2.1.

2.2 A necessidade de formalismos específicos para o PLN

Enquanto o tipo da gramática determina sua *capacidade gerativa fraca*, ou seja, o quão especificamente ela pode descrever o conjunto de sentenças que pertencem a uma dada língua, ainda devemos considerar sua *capacidade gerativa forte*, ou seja, o quão detalhada pode ser a descrição da estrutura resultante do processo de derivação.

Isto porque, quando consideramos o processamento de uma língua natural, o significado está sempre relacionado com a estrutura gerada durante o processo de derivação, ao contrário do que ocorre com as linguagens artificiais, onde o significado frequentemente independe desta estrutura [10].

Quando analisamos a sentença “João viu Maria com o telescópio”, por exemplo, é possível que obtenhamos as estruturas descritas na figura 2.2. Na primeira alternativa a estrutura da árvore de derivação sugere que Maria carregava um telescópio, enquanto que na segunda a estrutura sugere que João usou um telescópio para ver Maria.

Como vemos, a capacidade gerativa fraca deve ser encarada como o menor denominador comum entre gramáticas, tendo significado apenas quando uma gramática *falha*

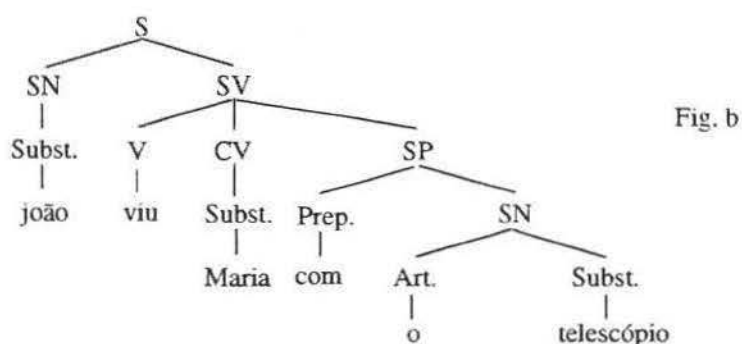
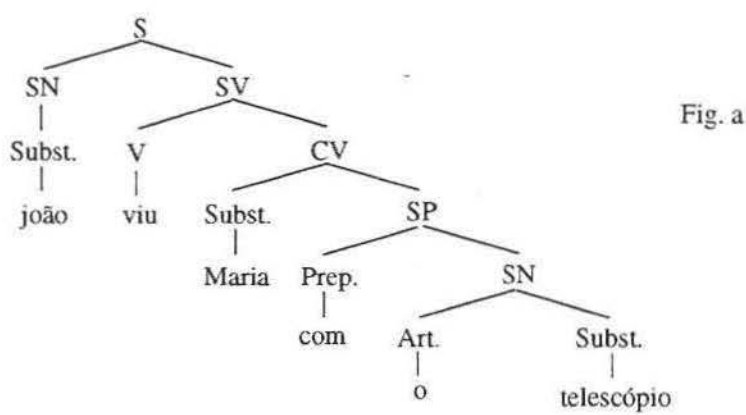


Figura 2.2: Duas possíveis descrições estruturais para “João viu Maria com o telescópio”.

ao reconhecer sentenças de uma língua cuja categoria é conhecida [16]⁷. O problema mais comum nos sistemas de PLN não está em sua capacidade de identificar as sentenças válidas de uma língua, mas na descrição estrutural destas sentenças, o que pode passar despercebido quando analisamos apenas a capacidade gerativa fraca [53].

Além do problema da qualidade da descrição, devemos considerar que o ramo da matemática discreta que trata da análise sintática — Teoria das Linguagens Formais — não aborda uma série de problemas importantes [54]:

- 1 não cobre nenhum tipo de semântica explícita;
- 2 não especifica a relação entre as estruturas sintáticas e os princípios de interpretação semântica;
- 3 analisa as sentenças fora de seu contexto, não permitindo definir a fonte ou a audiência, de forma que não existe um conhecimento mútuo assumido para ajudar na interpretação semântica, além de não prover nenhuma base para definir qual a intenção do autor ou qual o impacto que a sentença terá sobre os ouvintes/leitores;

⁷Extraído de [53].

- 4 as sentenças não estão ancoradas em nenhuma referência espaço-temporal;
- 5 a sentença é a unidade máxima de análise, impedindo o reconhecimento de organizações formais ou semânticas extraídas do discurso;
- 6 a palavra é a menor unidade de processamento, impedindo que porções menores como prefixos e radicais sejam utilizados para extrair informação; e
- 7 assume que as entradas de um léxico são palavras, impedindo que expressões idiomáticas ou frases feitas sejam tratadas corretamente.

Nem sempre é necessário analisar todos estes fatores num sistema de PLN, nem esta lista é exaustiva, mas para cada fator que se queira tratar será necessário providenciar uma extensão na teoria, o que nos leva a afirmar que são necessários formalismos e mecanismos específicos para o tratamento da língua natural.

Mesmo para as linguagens de programação uma atenção especial é dada aos itens 2 e 3 desta lista. Por exemplo, o item 2 é frequentemente tratado através de uma estrutura de pilha auxiliar chamada pilha semântica; o item 3 é tratado quando, nas linguagens fortemente tipadas, temos sobrecarga de operadores e a mesma sentença (que, no caso, é um comando) possui diferentes efeitos de acordo com o tipo de objeto que serve como operando, ou seja, de acordo com a “audiência” do operador.

2.3 Formalismos para o PLN

Nesta seção discutiremos alguns dos formalismos especificamente desenvolvidos para o PLN. Não é nossa intenção neste trabalho sermos exaustivos no assunto, até porque a quantidade de trabalhos relevantes é demasiado grande, mas sim destacar algumas características importantes que serão utilizadas mais tarde.

2.3.1 Gramáticas de adjunção de árvores (TAGs)

As gramáticas de adjunção [49] [47] fornecem um poder de reconhecimento ligeiramente superior ao das gramáticas livres de contexto. Este efeito é obtido através da expansão da estrutura das produções permitidas na gramática tanto em relação ao objeto de análise — com a troca da operação de concatenação de palavras pela composição de árvores — quanto nas operações permitidas — com a adoção da adjunção como operação fundamental.

Operar com adjunção de árvores, ao invés de reescrever e concatenar palavras, permite ao formalismo manipular diretamente a capacidade forte de geração, que é lingüisticamente mais relevante que a capacidade fraca de geração [51]. As línguas de adjunção de árvores

(TALs) englobam algumas línguas estritamente sensíveis ao contexto⁸ e fazem com que as TAGs se coloquem na categoria das gramáticas “meio sensíveis ao contexto” [47] [52] [91]. Este poder de reconhecimento aumentado, no entanto, é apenas pouco superior ao das gramáticas livres de contexto, como veremos mais adiante.

Uma TAG é uma quintupla $(\Sigma_T, \Sigma_N, I, A, \Sigma_S)$, onde Σ_T contém os símbolos terminais, Σ_N os símbolos não terminais e Σ_S é um conjunto unitário contendo o símbolo inicial. I e A são conjuntos finitos de árvores denominadas *elementares*. As árvores em I também são chamadas *árvores iniciais* e as em A de *árvores auxiliares* [47].

Em uma árvore inicial os nodos interiores são não terminais e os nodos nas fronteiras podem ser tanto terminais quanto não terminais; quando um nodo de fronteira for não terminal ele será marcado para substituição [75] [51].

Em uma árvore auxiliar os nodos interiores são não terminais e os nodos na fronteira podem ser tanto terminais quanto não terminais; todos os nodos não terminais da fronteira são marcados para substituição exceto um nodo obrigatoriamente presente, chamado nodo “pé”: este nodo deve ter o mesmo nome que a raiz da árvore [75] [51]. Um esboço esquemático das árvores pode ser visto na figura 2.3.

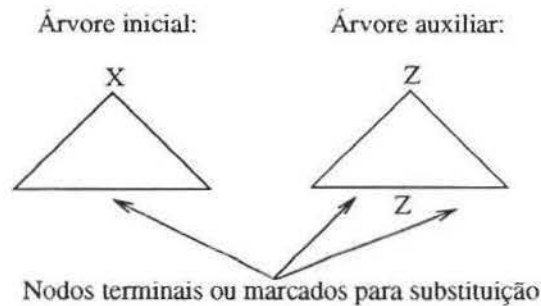


Figura 2.3: Esboço esquemático das árvores iniciais e auxiliares.

Apesar da definição das árvores elementares não ser muito restritiva, a idéia é que elas sejam de alguma forma mínimas. Nesta filosofia, uma árvore inicial corresponderia a uma árvore sentencial (ou estrutura argumental) mínima, sem recursão nos não terminais. Por sua vez, uma árvore auxiliar, com raiz e nodo pé rotulados X , corresponderia à estrutura mínima capaz de fornecer recursão durante a derivação, caso exista recursão em X [47]. Uma árvore obtida a partir da composição de outras árvores é chamada *árvore derivada* e é sobre esta árvore que utilizamos os operadores de adjunção e substituição.

A definição original do formalismo previa apenas a adjunção como operação de composição de árvores, mas em [76] passou-se a contar também com a operação de substituição.

⁸Línguas que não podem ser geradas pelas gramáticas livres de contexto.

Adjunção

A operação de adjunção compõe uma nova árvore a partir da combinação de uma árvore auxiliar e outra árvore qualquer. Sendo β uma árvore auxiliar cuja raiz é rotulada X e α uma árvore qualquer que contenha um nodo n não marcado para substituição também rotulado X , a árvore resultante γ , obtida através da adjunção de β em α no nodo n (figura 2.4) é construída como se segue [51]:

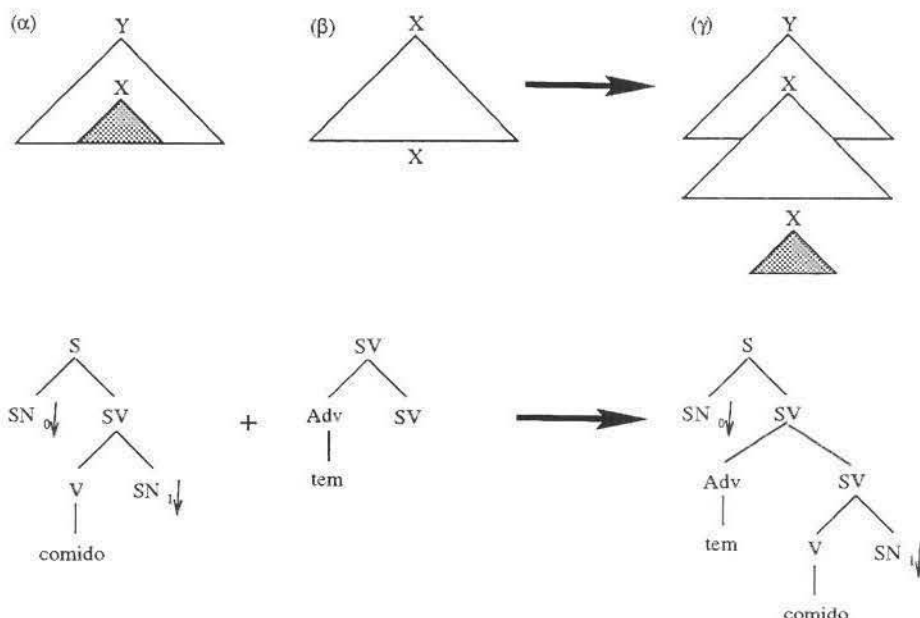


Figura 2.4: Adjunção: representação esquemática e um exemplo.

- A sub-árvore t de α , dominada⁹ por n , é separada, deixando uma cópia de n em α ;
- A árvore auxiliar β é fixada na cópia de n deixada em α ;
- O nodo pé de β é substituído pela sub-árvore t , ou seja, o nodo raiz de t , n , é afixado sobre o nodo pé de β .

Substituição

A operação de substituição não altera as propriedades fundamentais das TAGs [51] e ocorre apenas nos nodos não terminais da fronteira da árvore derivada. Um exemplo de substituição é dado na figura 2.5. Por convenção os nodos marcados para substituição

⁹Para árvores, a relação de dominância ocorre quando, para alcançarmos um determinado nodo P_1 a partir da raiz, obrigatoriamente passamos pelo nodo P_0 . Neste caso, P_0 domina P_1 .

estão assinalados com $\downarrow$. Quando uma substituição ocorre no nodo n , este nodo é substituído pela árvore adequada. Quando um nodo estiver marcado para substituição, apenas árvores derivadas a partir de árvores iniciais podem ser utilizadas na substituição. Por definição nenhuma adjunção é permitida num nodo marcado para substituição [51].

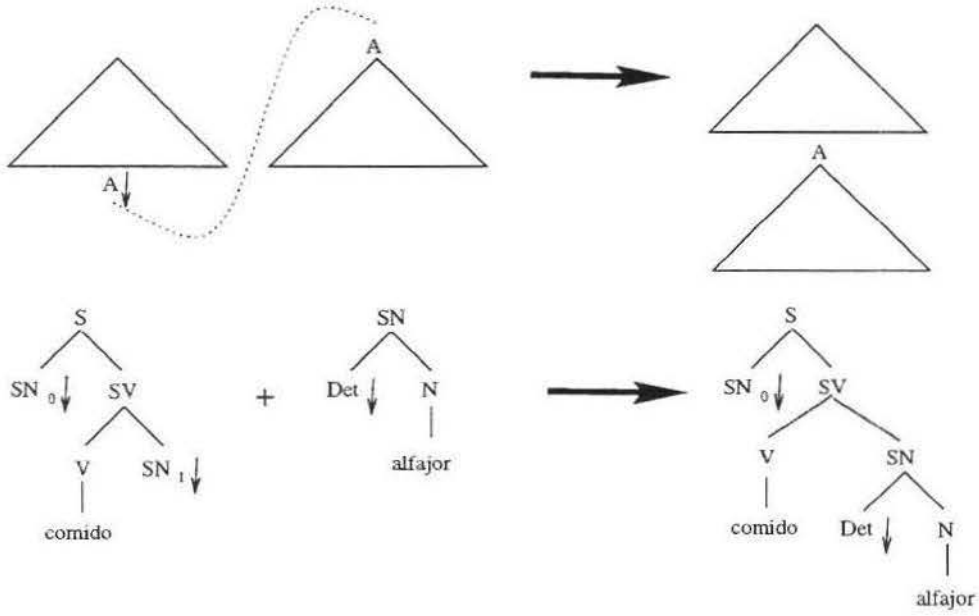


Figura 2.5: Substituição esquemática e um exemplo.

TAGs como uma extensão das gramáticas livres de contexto

Para melhor compreendermos o mecanismo das TAGs vamos iniciar utilizando-as como uma generalização das regras de produção que vimos até agora. Vamos partir então da gramática livre de contexto G'

$$S \rightarrow aTb$$

$$T \rightarrow Sa$$

$$T \rightarrow TT$$

$$T \rightarrow ab$$

com $\Sigma_S = \{S\}$, $\Sigma_N = \{S, T\}$ e $\Sigma_T = \{a, b\}$ (extraído de [47]).

Uma TAG tanto fortemente quanto fracamente equivalente pode ser definida como $G_{(a)} = (I, A)$ onde os conjuntos I e A estão representados na figura 2.6. Veremos mais detalhadamente adiante que uma gramática é fracamente equivalente a outra quando

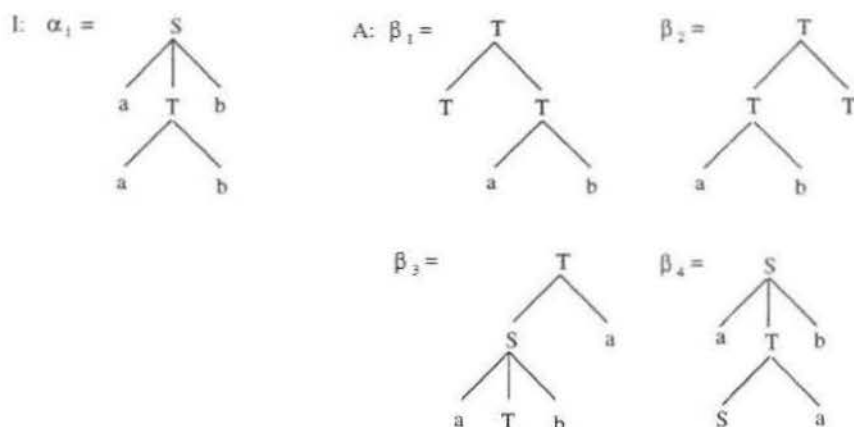


Figura 2.6: Gramática TAG $G_{(a)}$ equivalente a G' .

ambas podem reconhecer apenas as mesmas sentenças de entrada. Uma gramática é fortemente equivalente a outra quando, além de reconhecer as mesmas sentenças de entrada, fornece as mesmas descrições estruturais, ou seja, constrói as mesmas árvores de derivação.

Vejamos algumas derivações em $G_{(a)} = (I, A)$ (figura 2.6). Começemos com a seleção das estruturas iniciais (figura 2.7).

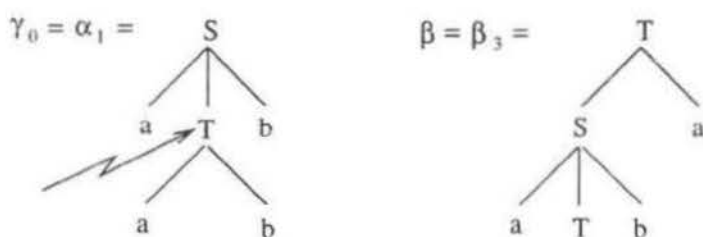


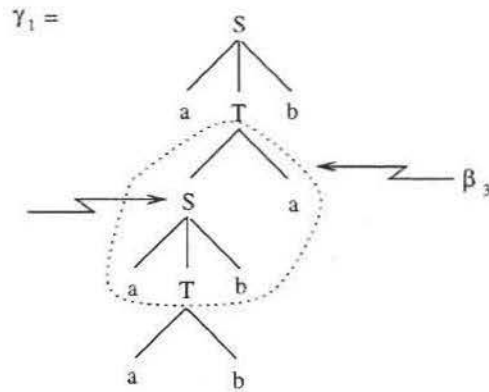
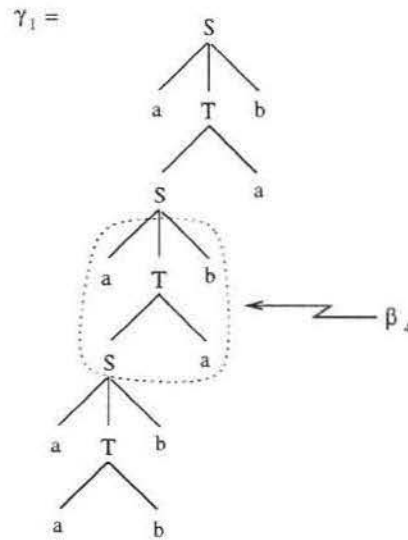
Figura 2.7: Seleção de α , β e γ .

β_3 será adjuntada em γ_0 sobre T como indicado em γ_0 . A árvore derivada resultante γ_1 pode ser vista na figura 2.8.

Pode-se continuar a derivação adjuntando, por exemplo, β_4 em γ_1 sobre S como indicado em γ_1 . A árvore resultante γ_2 é a mostrada na figura 2.9.

Para qualquer gramática livre de contexto existe uma gramática de adjunção de árvores (TAG) tanto fracamente quanto fortemente equivalente [47]. O exemplo da figura 2.6 é fortemente — e conseqüentemente fracamente — equivalente a G' . Em relação às gramáticas livres de contexto podemos afirmar que existe alguma TAG G tal que [49] [47]:

- (a) existe uma gramática livre de contexto G' que é tanto fracamente quanto fortemente equivalente a G ;

Figura 2.8: γ_1 : Adjunção de α_1 em β_3 sobre T .Figura 2.9: γ_2 : adjunção de β_4 em γ_1 sobre S .

- (b) existe uma gramática livre de contexto G' que é fracamente equivalente a G mas não é fortemente equivalente a G ; e
- (c) não existe gramática livre de contexto equivalente a G .

O item (a) foi exemplificado pela gramática exposta na figura 2.6. Agora demonstraremos as afirmações (b) e (c).

Seja $G_{(b)}$ a TAG descrita na figura 2.10. Podemos observar algumas derivações na figura 2.11 e notar que a linguagem gerada por G ,

$$L(G) = a^n e b^n | n \geq 0$$

é uma linguagem livre de contexto. Isto quer dizer que existe uma gramática G' livre de contexto fracamente equivalente a $G_{(b)}$ [47]. Mas segundo [13] também sabemos que

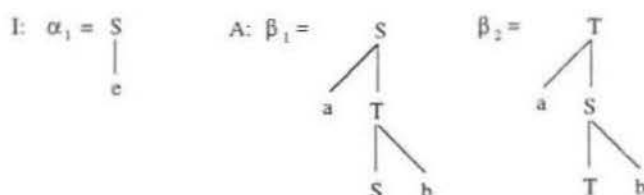


Figura 2.10: Uma TAG sem gramática livre de contexto fortemente equivalente.

não existe uma gramática livre de contexto fortemente equivalente a $G_{(b)}$. Isto porque uma TAG é capaz de assinalar descrições estruturais a uma língua livre de contexto que não pode ser obtida por nenhuma gramática livre de contexto, confirmando (b). Para denotarmos esta diferença dizemos que uma TAG gera não apenas uma linguagem — um conjunto adequado de sentenças $L(G)$ — mas também um conjunto de árvores denotado como $T(G)$ e, no caso de $G_{(b)}$, $T(G_{(b)})$ não pode ser gerado por nenhuma gramática livre de contexto, apesar de $L(G_{(b)})$ poder ser reconhecida.

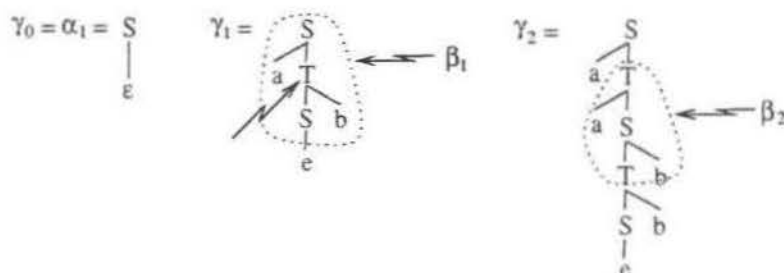


Figura 2.11: Derivação em uma TAG sem gramática livre de contexto equivalente.

Seja $G_{(c)}$ a TAG descrita na figura 2.12. Podemos caracterizar a linguagem gerada por $G_{(c)}$ como se segue. Iniciemos com uma linguagem livre de contexto:

$$L_1 = \{(ab)^n ec^n | n \geq 0\}.$$

$L(G_{(c)})$ pode ser obtida tomando sentenças de L_1 e deslocando alguns 'a's para a esquerda. A definição precisa de $L(G_{(c)})$ é a seguinte:

$$L(G) = L_1 \{wec^n | n \geq 0\}, \text{ onde } w \text{ é uma cadeia de caracteres de 'a's e 'b's tal que}$$

- (i) o número de 'a's = número de 'b's; e
- (ii) para qualquer subcadeia de caracteres inicial de w , o número de 'a's $\geq$ número de 'b's

$L(G_{(c)})$ é uma linguagem estritamente sensível ao contexto [47] e portanto não pode ser descrita por uma gramática livre de contexto. Isto pode ser demonstrado fazendo a

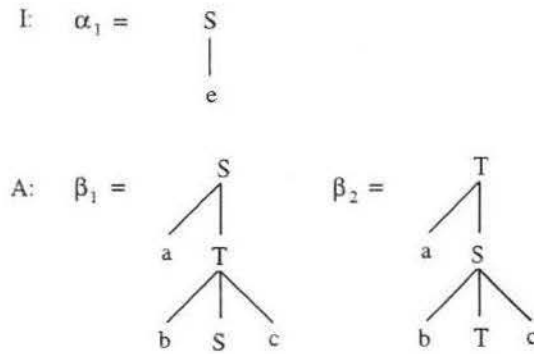


Figura 2.12: TAG $G_{(c)}$ sem gramática livre de contexto equivalente.

interseção de L_1 com a linguagem regular $a^*b^*ec^*$ que resulta na linguagem

$$L_2 = \{a^n b^n e c^n | n \geq 0\} = L_1 \cap a^* b^* e c^*.$$

L_2 é um exemplo bastante conhecido de linguagem sensível ao contexto. O resultado da interseção de uma linguagem livre de contexto com uma expressão regular é sempre uma linguagem livre de contexto, ou seja, L_1 não é uma linguagem livre de contexto mas sim uma linguagem estritamente sensível ao contexto [47].

Adjunção com contexto

Até agora descrevemos como uma árvore auxiliar β pode ser adjunta em um nodo quando este tem o mesmo rótulo que a raiz e o nodo pé de β é não terminal e não está marcado para substituição. Do ponto de vista lingüístico, no entanto, é interessante que possamos definir quais conjuntos de árvores podemos adjuntar num determinado nodo, o que pode ser feito através da especificação de contexto na adjunção. A noção de contexto aqui é muito restrita e talvez uma tradução mais adequada seria restrição na adjunção (da expressão em inglês *constraints on adjunction*).

As restrições podem ser especificadas em qualquer nodo de uma árvore elementar onde a adjunção seja permitida e estão divididas em três grupos mutuamente exclusivos [51]:

- Adjunção seletiva (*Selective Adjunction* ou $SA(T)$): somente os membros do conjunto $T \subseteq A$ de árvores auxiliares podem ser adjuntados neste nodo. A adjunção não é obrigatória.
- Adjunção nula (*Null Adjunction* ou NA): proíbe a adjunção neste nodo. Equivale a $SA(T)$ onde T é um conjunto vazio.
- Adjunção obrigatória (*Obligatory Adjunction* ou $OA(T)$): uma árvore auxiliar membro de $T \subseteq A$ é obrigatoriamente adjunta neste nodo.

Este tipo de restrição visa facilitar o desenvolvimento de regras para o processamento de uma língua natural e não estão relacionadas à natureza matemática do formalismo. Existem no entanto algumas propriedades formais de fechamento que exigem um outro tipo de restrição na adjunção [47]. Para diferenciar as regras descritas das que se seguem, estas últimas são denominadas *restrições locais* (da expressão em inglês *local constraints*).

Uma TAG $G = (I, A)$ com restrições locais é uma TAG onde I é o conjunto de árvores iniciais e A o conjunto de árvores auxiliares e para cada árvore auxiliar existe uma restrição (possivelmente nula) [47]. A definição formal destas restrições foge ao escopo deste trabalho, mas pode ser encontrada em [50]. Aqui vamos nos concentrar na idéia central, exemplificando o funcionamento do processo de análise das restrições.

Seja G a TAG descrita na figura 2.13. A árvore auxiliar β_1 possui uma restrição especificada para a direita de β_1 . Esta restrição pode ser tratada como um predicado que deve ser satisfeito em um nodo n da árvore derivada t para que β_1 possa ser adjuntada em t sobre n . Neste exemplo — retirado de [47] — o predicado $LR(\phi - \psi)$ é um predicado de contexto adequado¹⁰ que será satisfeito se no nodo n da árvore t existir um contexto adequado (um corte) da árvore t que passe através do nodo n e que possua a forma

$$\rho_1 \phi_1 A \psi_1 \rho_2,$$

onde ρ_1 e ρ_2 são seqüências arbitrárias de terminais e não terminais e A é o rótulo de n . Se o predicado $LR(\phi - \psi)$ for satisfeito em n isto quer dizer que existe um contexto ϕ_1 à esquerda de n e um contexto ψ_1 à direita de n [47].

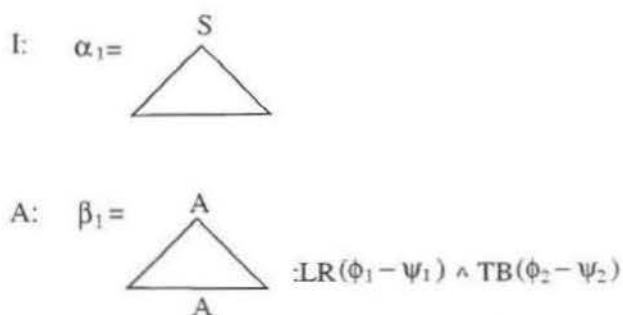


Figura 2.13: TAG G com restrições locais na adjunção.

O predicado $TB(\phi_2 - \psi_2)$ é chamado predicado de dominância e é satisfeito se no nodo n da árvore t existir um caminho a partir da raiz até a fronteira passando por n na forma

$$\rho_1 \phi_2 A \psi_2 \rho_2,$$

¹⁰Do inglês *proper analysis*.

onde ρ_1 e ρ_2 são seqüências arbitrárias de terminais e não terminais e A é o rótulo de n . Se o predicado $TB(\phi_2 - \psi_2)$ for satisfeito em n isto significa que existe um contexto ϕ_2 acima de n e um contexto ψ_2 abaixo de n [47].

A aplicação de restrições locais e de restrições na adjunção aumentam o poder gerativo das TAGs sem, no entanto, equipará-lo ao das gramáticas sensíveis ao contexto. Uma análise detalhada de como as restrições locais afetam o poder de reconhecimento de uma TAG pode ser encontrada em [47] e [48]. Em [48], [92] pode ser verificado como as restrições na adjunção afetam o poder de reconhecimento.

TAGs com *links*

A operação de adjunção nos permite movimentar sub-árvores específicas da árvore derivada, simulando a movimentação dos constituintes de uma sentença em língua natural. Enquanto esta movimentação é interessante do ponto de vista lingüístico, ela é um complicador do ponto de vista formal. Quando analisamos uma derivação em uma gramática livre de contexto, podemos determinar a maneira pela qual uma solução, ou árvore derivada final foi construída, o que não ocorre quando a árvore derivada foi construída usando TAGs.

Determinar que sub-árvores foram geradas a partir de quais nodos da nossa árvore elementar original pode ser interessante para explicitar qual o “sentido” da estrutura final, ou seja, qual a relação original entre os nodos das árvores elementares que deram origem à solução. Além disso, as árvores elementares são o domínio adequado para caracterizar algumas dependências, tais como dependências de subcategorização ou preenchimento de vazios da estrutura¹¹ [47].

Apesar desta informação não estar completamente disponível nas TAGs, alguma informação pode ser obtida com a introdução de um tipo especial de relação entre pares específicos de nodos das árvores elementares denominado ligação ou *link*. Graficamente esta relação será representada como um arco pontilhado de um nodo até o outro, assim como na figura 2.14.

Uma ligação é uma relação direcional e assimétrica, ou seja, se um nodo n_1 está ligado a um nodo n_2 em uma árvore elementar então [47]:

- n_2 c-comanda¹² n_1 . Isto quer dizer que n_2 precede n_1 e que existe um nodo m que imediatamente domina n_2 e também domina n_1 .

¹¹Elementos que, apesar de previstos na estrutura completa inicial, podem não estar presentes na solução final. Quando isto ocorre diz-se que o constituinte que falta não foi realizado.

¹²Este conceito está relacionado com a teoria de Government and Binding [17] e será novamente abordado ao tratarmos as Bound Movement Grammars.

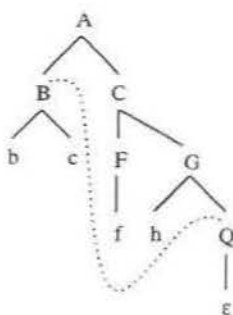


Figura 2.14: TAG G com nodos B e Q ligados por um *link*.

- n_1 domina uma cadeia de caracteres nula, que é representada por um símbolo terminal nos exemplos formais e por ϵ nos lingüísticos.

Lingüisticamente falando, tanto n_1 quanto n_2 pertencem à mesma categoria gramatical e somente n_1 vai dominar uma cadeia de caracteres nula. Uma TAG com *links* é uma TAG onde algumas das árvores elementares podem ter ligações determinadas *a priori* e serão referidas apenas como TAGs.

Dependências cruzadas e encaixadas

Os *links* são definidos nas árvores elementares, mas o ponto importante é que eles são preservados pelas operações de substituição e adjunção [47] [51]. Durante o processo de derivação, no entanto, estes *links* podem ser “esticados” ou deslocados e explicitam dependências tanto cruzadas quanto encaixadas. Para exemplificar este processo vamos tomar a gramática descrita na figura 2.10 e inserir *links* (figura 2.15).

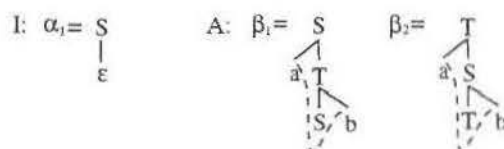


Figura 2.15: TAG G da figura 2.10 com *links*.

Agora observemos a seqüência de derivações da figura 2.16 onde as dependências estão assinaladas abaixo das árvores. A árvore derivada γ_1 mostra uma dependência simples enquanto γ_2 mostra dependências encaixadas que surgem após a adjunção de β_1 . Já γ_3 mostra tanto dependências encaixadas como cruzadas após as adjunções consecutivas de β_1 e β_2 . Mesmo sem a presença dos *links* as dependências existiriam; eles apenas as tornam explícitas.

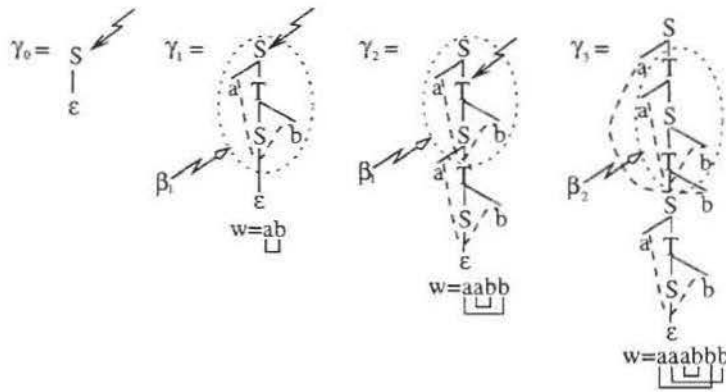


Figura 2.16: Dependências cruzadas e encaixadas durante uma derivação.

Como podemos observar, apesar dos *links* serem definidos nas árvores elementares, eles são preservados durante a derivação. No exemplo dado, uma dependência cruzada surgiu através de uma adjunção, mas ela poderia ter sido definida diretamente na gramática.

Apesar de não utilizarmos substituições no exemplo, é fácil entender porque esta operação também não afeta os *links*: ela somente pode ocorrer em um nodo não-terminal da fronteira da árvore que não pode ser adjuntado (por definição). Isto quer dizer que a referência original do *link* se mantém posicionalmente fixa, ou seja, onde havia um nodo rotulado X sobre o qual se executa uma substituição, continua havendo um nodo rotulado X . Se o nodo é realmente o mesmo ou uma cópia é uma questão irrelevante para o formalismo.

TAGs nas línguas naturais

As TAGs vêm sendo utilizadas para o processamento, entre outras línguas, do inglês [2], francês [1] e alemão [71]. Também foram utilizadas para o português em [56], onde toda uma gramática normativa do português foi mapeada para uma TAG equivalente. Utilizamos estas evidências para indicar que o formalismo é adequado para o processamento da língua natural. Seria muito difícil provar a adequação do formalismo porque não existe um consenso sobre quais propriedades tornam um formalismo adequado para o PLN. Uma referência interessante sob este aspecto é [37] onde se aborda a implementação da teoria de *Government and Binding* de Chomsky utilizando-se TAGs.

Alguns outros indícios tornam as TAGs atrativas, dentre eles podemos destacar a proliferação de publicações e de variações em torno do formalismo e ainda a natureza matemática de sua definição e de suas características. Mais tarde também veremos que existem estudos disponíveis sobre como implementar uma ferramenta de análise sintática eficiente.

Por hora vamos aplicar o poder descritivo das TAGs sobre alguns exemplos lingüísticos

para que possamos instanciar adequadamente a definição formal dada.

A figura 2.17 contém as árvores iniciais de uma TAG e a estrutura destas árvores reflete uma estrutura argumental “mínima” de verbo transitivo direto (α_1) e uma estrutura mínima de verbo intransitivo (α_2). α_1 reflete diretamente a estrutura de sentenças simples como “o vendedor era um chato”.

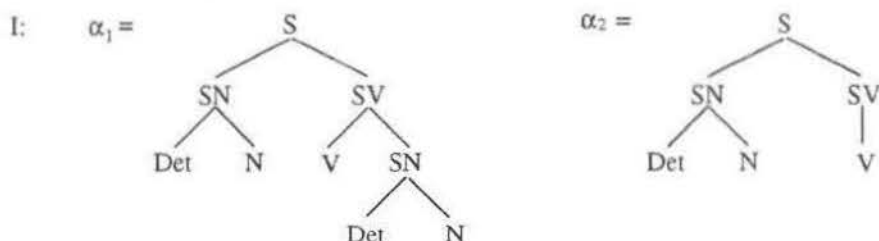


Figura 2.17: Árvores iniciais que representam a sentença mínima de um verbo transitivo direto e de um verbo intransitivo.

A figura 2.18 possui duas representações da mesma estrutura, uma como árvore inicial (α_3) e outra como auxiliar (β_1); mesma estrutura no sentido de ambas representarem um dos formatos de sentença ‘*qu*’¹³. A utilização destas estruturas, no entanto, se dá em operações diferentes — uma na substituição e outra na adjunção — e sofre restrições características de cada uma destas operações. α_3 pode ser utilizada para representar a sentença “quem perturbou Ivan?”, enquanto β_1 pode representar “que perturbou Ivan”. Note que existe um *link* determinando a relação da partícula ‘*qu*’ com o constituinte previsivelmente não realizado da estrutura.

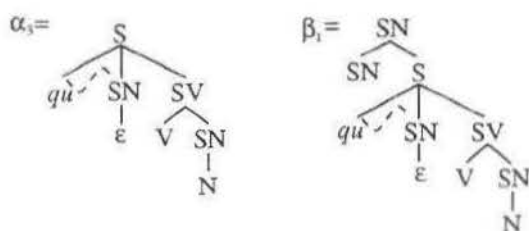


Figura 2.18: Uma árvore inicial e uma árvore auxiliar representando uma das estruturas ‘*qu*’.

Instanciando-se as árvores das figuras 2.17 e 2.18 com as sentenças exemplo fornecidas podemos obter as derivações descritas nas figuras 2.19 e 2.20. Pode-se observar como as TAGs agrupam os constituintes, mas também que é possível manipular uma gramática livre de contexto para se obter a mesma resposta.

Neste ponto deve-se considerar o quão complexo foi descrever e obter a derivação desejada da sentença com uma ou outra técnica.

¹³Perguntas que contém “que”, “quando”, “quanto”, etc.

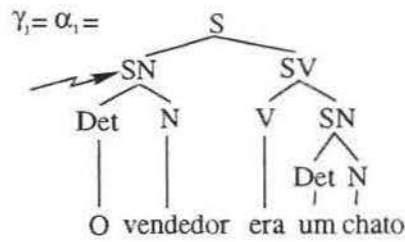
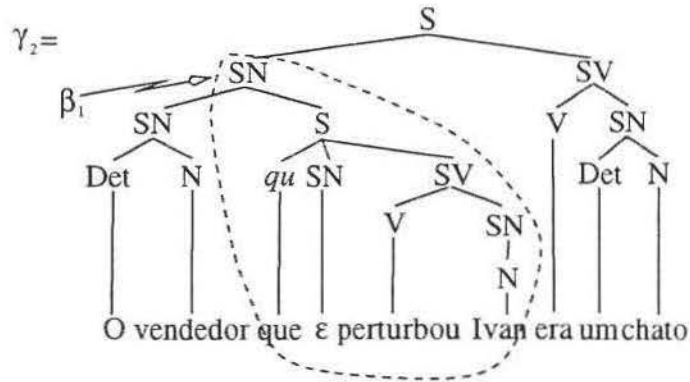
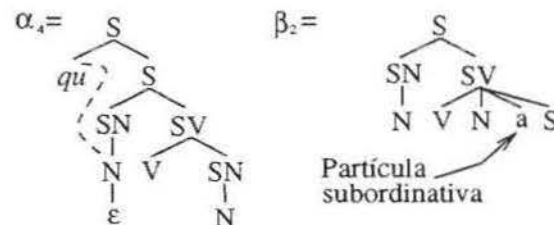
Figura 2.19: Instanciação de α_1 .

Figura 2.20: Derivação com itens instanciados.

A literatura aponta que as TAGs possuem uma característica denominada *domínio de localidade estendida*, referindo-se em parte à própria maneira como as estruturas gramaticais são definidas (como árvores), à capacidade de deslocar parcelas significativas da árvore derivada sem perder as referências locais, e também à capacidade de manter os *links* entre nodos de uma árvore elementar como anteriormente demonstrado. Esta característica se reflete, por exemplo, na análise do português durante a manipulação de uma sentença ‘qu’ que envolva, por exemplo, uma oração subordinada, coordenada ou mesmo um aposto. Consideremos uma nova árvore inicial α_4 e uma nova árvore auxiliar β_2 na nossa gramática (figura 2.21).

Figura 2.21: Árvores elementares que complementam G .

Podemos acomodar a frase “quando compraste caqui?” na estrutura de α_4 e referenciar

o constituinte suprimido pela partícula ‘qu’ diretamente. A figura 2.22 mostra como os *links* se preservam na adjunção de β_2 no nodo indicado de $\alpha_4 = \gamma_1$ para acomodar a oração “quando João convenceu Ivan a comprar caqui?”.

TAGs com *features* — FB-TAG

Os verbos da sentença anterior se encontram propositalmente no infinitivo para facilitar a construção do exemplo, mas é importante que possamos manipular as informações dos itens léxicos para garantir concordância verbal e nominal ou qualquer outro tipo de consistência deste tipo.

As TAGs tratam deste problema através da agregação de informações léxico-morfológicas aos itens léxicos que são propagadas pela árvore de derivação à medida em que ela é gerada. Este processo de propagação também é chamado *tree climbing* e as informações léxico-morfológicas também podem ser chamadas simplesmente de *traços*.

Com a informação disponível é necessário também definir como manipular os dados para verificar se as restrições lingüísticas adequadas são satisfeitas, como no caso da concordância verbal do nosso exemplo.

Uma idéia simples que pode ocorrer ao leitor é inserir regras de concordância nos nodos das árvores e fazer com que elas se refiram aos demais nodos posicionalmente. Por exemplo no caso da figura 2.22 poderíamos aproveitar o *tree climbing* dos traços e obrigar o nodo rotulado *SN* a concordar com o nodo imediatamente a sua esquerda (*SV*). Idéias simples geralmente levantam suspeitas e neste caso fundadas. Com esta técnica rudimentar não poderíamos especificar regras que envolvessem nodos em níveis diferentes da árvore porque uma adjunção pode modificar esta relação. Mesmo que garantíssemos que a adjunção não ocorreria pela adoção de uma restrição *NA* (*Null Adjunction*) nos nodos, este tipo de regra seria mais um complicador na referência posicional. Ainda mais grave é que a combinação de traços durante o *tree climbing* pode tornar a informação complexa demais para ser útil. Se um sintagma nominal *SN* domina imediatamente dois nomes (N_1, N_2) — como José e Mariana, por exemplo — *SN* terá traços masculinos, femininos ou ambos?

Outra idéia simples e que soluciona o problema da referência é identificar univocamente cada nodo da árvore na gramática. Neste caso bastaria instanciar os identificadores dos nodos e das regras durante a derivação para termos regras sem problemas de referência posicional e sem problemas de combinação de traços. Neste caso o que ganharíamos sem perceber é um espaço de busca do tamanho da própria árvore e boa parte do custo associado às gramáticas sensíveis ao contexto sem nenhum de seus benefícios.

A solução encontrada na literatura, apesar de inicialmente não parecer simples, é elegante e, além de fornecer os mecanismos de concordância, ainda facilita o próprio processo de derivação. Inicialmente apresentada em [87] [89], as *feature based TAGs*,

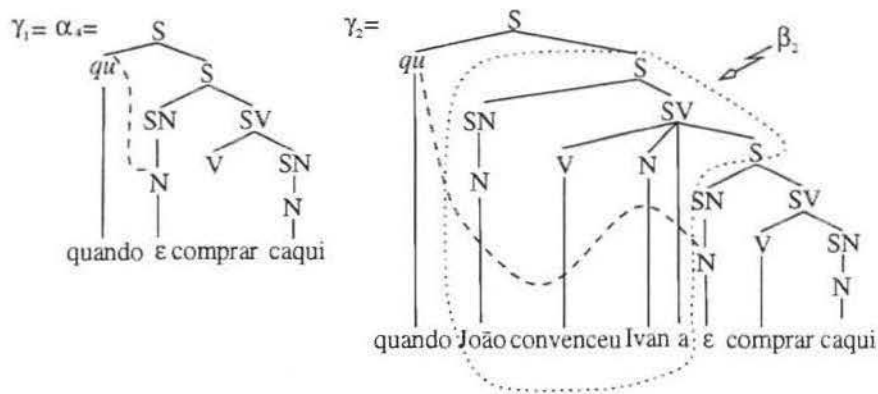


Figura 2.22: Controle de localidade estendida numa sentença ‘qu’.

ou simplesmente FB-TAGs, apresentam o conceito das *feature structures*. Para uma apresentação mais detalhada remetemos o leitor para [90].

Uma *feature structure* é um conjunto de predicados que devem ser satisfeitos num determinado nodo para que a derivação neste nodo seja considerada correta. Cada nodo de uma árvore elementar contém duas *feature structures* denominadas *top* e *bottom*.

A estrutura *top* de um nodo n contém as regras de unificação — os predicados a serem satisfeitos — que fazem referência aos nodos irmãos de n , isto é, àqueles nodos que também são imediatamente dominados pelo mesmo nodo que domina imediatamente n . A estrutura *bottom* de n contém as regras de unificação que fazem referência aos nodos dominados por n .

Quando a derivação é completada os traços *top* e *bottom* de todos os nodos são unificados. Se os traços *top* e *bottom* de um nodo não unificam então uma árvore auxiliar deve ser adjuntada naquele nodo [75]. O comportamento dos traços durante uma adjunção pode ser observado na figura 2.23.

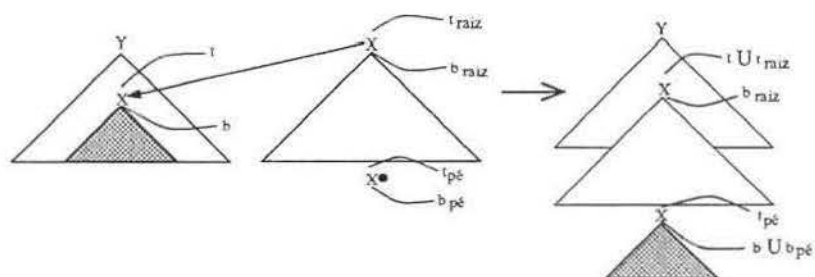


Figura 2.23: Comportamento dos traços durante uma adjunção.

Para cada nodo marcado para substituição podemos associar apenas uma *feature structure* que age como uma estrutura *top* [75]. O comportamento dos traços durante uma substituição pode ser observado na figura 2.24.

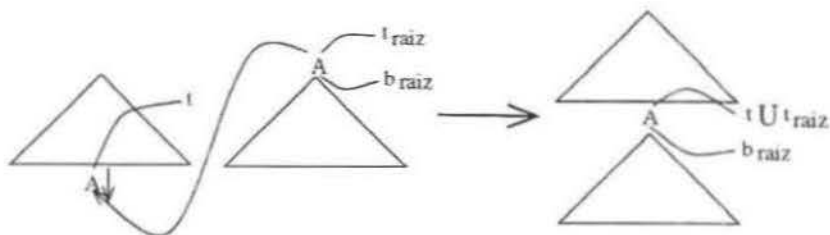


Figura 2.24: Comportamento dos traços durante uma substituição.

O comportamento dos conjuntos de traços, que prevê tanto a combinação quanto o deslocamento dos mesmos, permite que se mantenha o controle de como as regras de unificação serão aplicadas. Um exemplo de árvore inicial com as *features* à mostra está na figura 2.25.

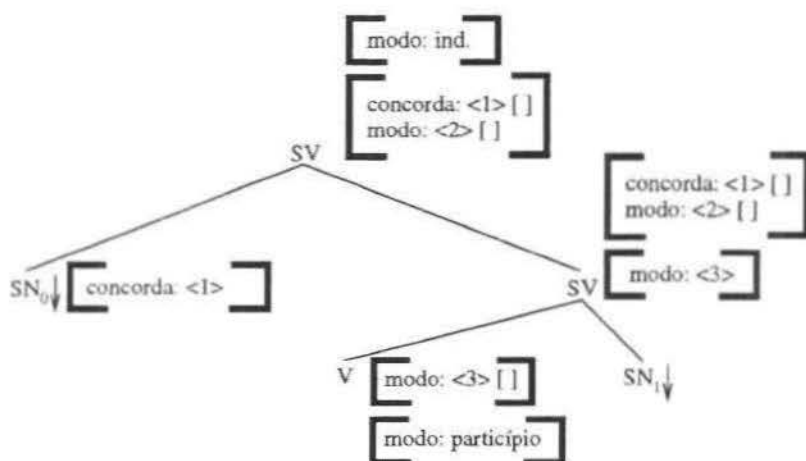


Figura 2.25: Parte de uma árvore inicial com *features* à mostra.

Neste exemplo a concordância nominal é checada entre os nós S , SN e SV . O modo é checado entre SV e V gerando um problema: V tem traço *bottom* indicando particípio passado e VP deve concordar com ele e com S_r que tem um traço *top* indicando modo indicativo. Por definição, uma adjunção deve ocorrer em VP , que é o nó onde os traços não unificam.

O traço “particípio” de V obrigatoriamente vem do item léxico nele afixado e que foi omitido na figura 2.25. Suponhamos que seja a flexão do verbo “comer” no particípio passado, ou seja, “comido”. Quando completamos a estrutura com a sentença “João comido banana” vemos que realmente a sentença é agramatical e que a restrição OA (adjunção obrigatória) deduzida a partir da unificação dos traços está correta.

Considere a árvore auxiliar mostrada na figura 2.26. Executando-se uma adjunção desta árvore no nó VP da figura 2.25, é gerada como resultado a árvore descrita na

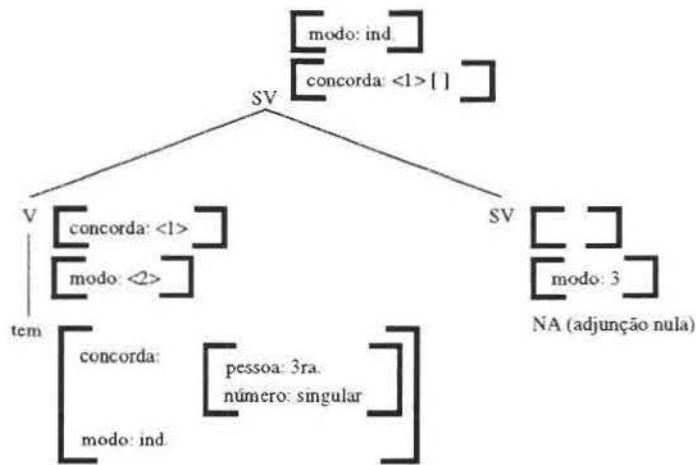
Figura 2.26: Árvore auxiliar com *features*.

figura 2.27 que representa a sentença “João tem comido banana”, perfeitamente gramatical.

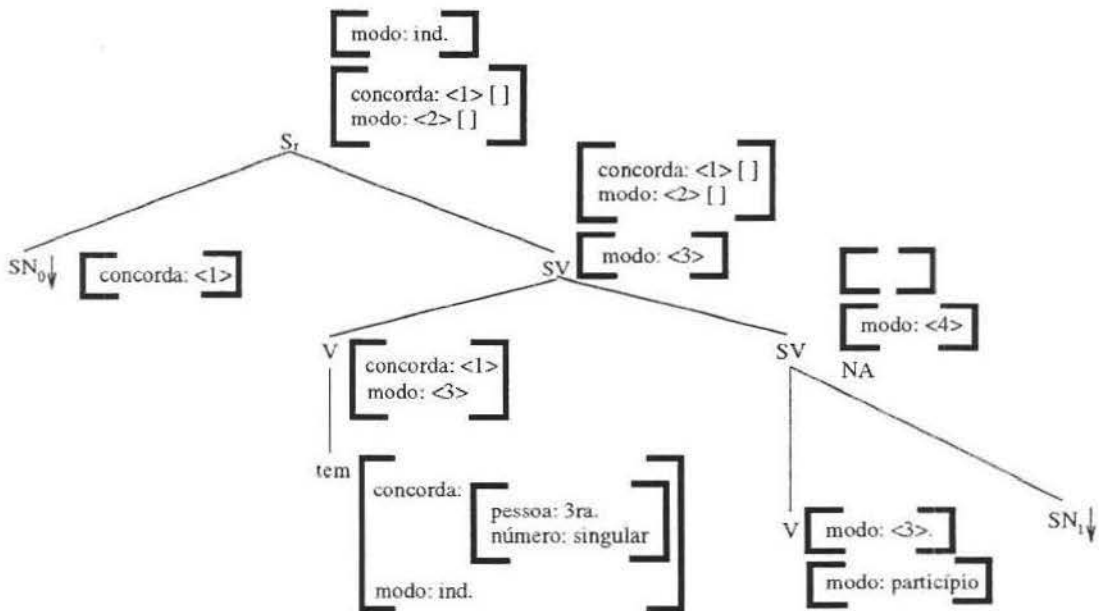


Figura 2.27: Derivação de “João tem comido banana” com unificação correta.

Enquanto a definição das TAGs dada até agora permite que se selecione quais estruturas podem ser adjuntadas sobre quais nodos, ela ainda determina uma solução de força bruta para determinar o espaço de busca das estruturas da gramática quando recebemos uma determinada sentença para análise. Os exemplos dados, por outro lado, se aproveitam do conhecimento sobre quais itens podem ser afixados sobre quais estruturas. A

maneira encontrada para modelar computacionalmente este conhecimento sobre a relação dos itens léxicos com as árvores elementares é chamada *lexicalização gramatical* que, no caso das TAGs, gera a variação *lexicalized TAG* ou L-TAG.

TAGs lexicalizadas — L-TAG

Como vimos, é necessário garantir que os itens léxicos da sentença de entrada sejam alocados na árvore elementar adequada, para que tenhamos a derivação desejada como solução.

Originalmente isto era garantido pela verificação de uma série de restrições — concordância e subcategorização, por exemplo — para cada árvore elementar [47]. Na figura 2.17 apresentamos as estruturas mínimas para um verbo transitivo direto e uma para um verbo intransitivo. Nós não poderíamos encaixar a frase “o maluco caiu sorrisal” na primeira estrutura porque a subcategorização do verbo é inadequada, nem poderíamos associar “os problema surgiram” com a segunda estrutura porque, como toda a estrutura está disponível para análise desde o início do processo de derivação, pode-se verificar que não existe concordância.

A formalização do processo de verificação de concordância veio com as FB-TAGs como vimos anteriormente, mas a formalização da verificação da subcategorização, assim como a operação de substituição, veio com as TAGs lexicalizadas — L-TAGs [76], [75].

Uma gramática lexicalizada é composta por [75]:

- um conjunto finito de estruturas, cada qual associado a um item léxico que será então chamado de âncora desta estrutura; a estrutura define o domínio de localidade sobre o qual as restrições são especificadas; e
- uma ou mais operações de composição das estruturas.

A primeira parte desta definição é na verdade um léxico, ou seja, um conjunto finito de estruturas, cada qual associada com uma âncora¹⁴. As estruturas definidas diretamente no léxico são chamadas de estruturas elementares; estruturas construídas pela composição de outras são denominadas estruturas derivadas. A diferença entre uma estrutura elementar e uma árvore elementar é a presença da âncora.

A categoria de uma palavra no léxico é dada pela estrutura que ela ancora, ou seja, a estrutura que ela seleciona. Uma gramática é dita lexicalizada quando todas suas estruturas estão ancoradas por um ou mais itens léxicos [75]. Note que quando dois ou mais itens ancoram a mesma estrutura, eles pertencem à mesma categoria. Também é possível que uma determinada estrutura seja ancorada por mais de um item simultaneamente, ou

¹⁴Na literatura sobre gramáticas lexicalizadas a âncora também é chamada de cabeça (*head*).

seja, que exista mais de um terminal alocado na estrutura. Neste caso podemos selecionar um dos itens léxicos como âncora ou assumir que existe uma âncora multi-componente, ou seja, para que a estrutura seja utilizada mais de um item léxico deve estar presente¹⁵.

Mesmo desconsiderando a ambigüidade categorial, também podemos ter um mesmo item ancorando mais de uma estrutura. Na verdade isto é bastante comum e, neste caso, diz-se que o item ancora uma *família* de árvores. Uma família de árvores é um conjunto de árvores sentenciais que compartilham a mesma estrutura argumental, abstraída da instância da âncora. Em outras palavras, cada árvore de uma família pode ser vista como uma transformação da mesma estrutura argumental [92]. Uma possível família de árvores pode ser vista na figura 2.28 e corresponde à família de árvores dos verbos que pedem apenas um argumento [75]. O símbolo $\diamond$ indica a posição onde deve ser afixada a âncora da estrutura.

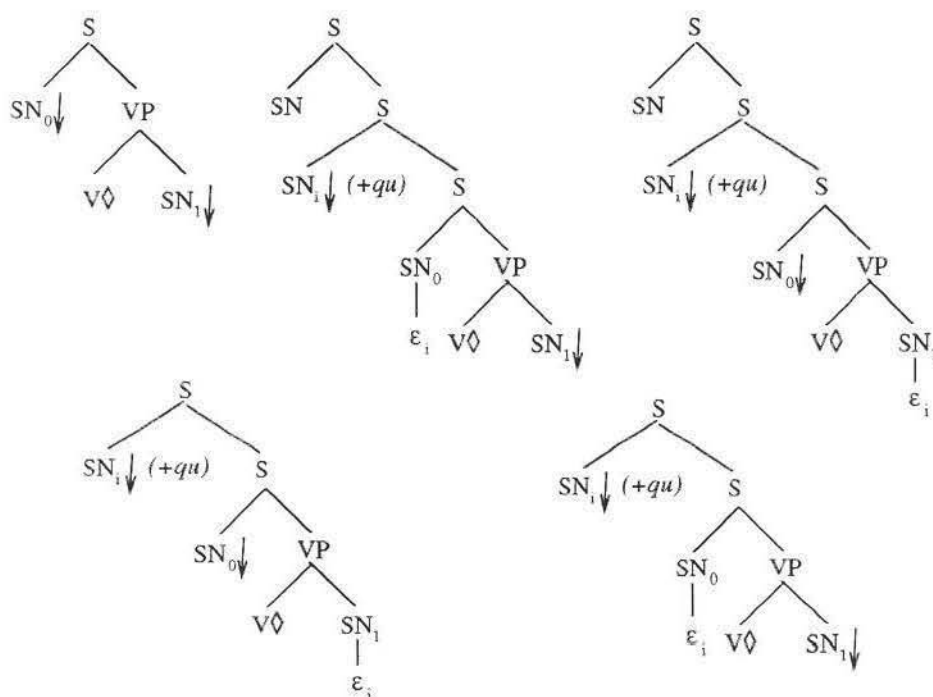


Figura 2.28: Possível família de árvores de verbos que pedem apenas um argumento.

Se podemos ter famílias de árvores que independem da âncora então isto quer dizer que as informações de unificação — *features* — independentes do valor da âncora podem ser definidas diretamente nas árvores da família, assim como nas FB-TAGs. Por exemplo a concordância entre o sujeito e o verbo da figura 2.25 independe da âncora, ao contrário do tempo verbal do nodo *V*, que depende do item léxico (suprimido na figura).

¹⁵ Expressões idiomáticas como “bateu as botas” — morreu — poderiam formar uma âncora multi-componente.

A definição de gramática lexicalizada dada em [76] também exige que as estruturas do léxico tenham um tamanho finito e que as operações de composição gerem um número finito de combinações possíveis. Esta definição torna a gramática finitamente ambígua, ou seja, não existe nenhuma sentença de tamanho finito que possa ser analisada de infinitas maneiras. Conseqüentemente decidir se uma sentença pertence ou não a uma dada gramática é um problema decidível para estas gramáticas [75].

Sob o aspecto lingüístico uma gramática lexicalizada explicita a relação entre os núcleos de uma sentença e os adjuntos. À medida que associamos os itens léxicos percebemos que as árvores iniciais representam uma estrutura argumental completa enquanto que as árvores auxiliares representam os adjuntos — adjetivos, advérbios ou cláusulas relativas, verbos auxiliares e predicados que exigem complementos sentenciais, entre outros [75]. As cláusulas sentenciais são representadas por árvores elementares com raiz do tipo *S*, ancoradas pelo elemento predicativo que, na maioria dos casos, é o verbo principal.

O comportamento de uma gramática lexicalizada depende do tipo das estruturas definidas. Em [75] e [2], mas principalmente, em [92] temos informações sobre as gramáticas desenvolvidas para o inglês. Em [56] é descrita uma gramática para o português que não considera a lexicalização gramatical e possui apenas uma versão limitada das *features*. A construção de uma TAG robusta para o português continua uma questão em aberto.

Estratégias de análise sintática

A gramática para o inglês do projeto XTAG [92] conta com mais de 1000 árvores divididas em 82 famílias diferentes. Kipper [56] aponta indícios de que uma descrição para o português sem lexicalização atingiria a casa das 500 árvores. Como no processo de lexicalização o número de árvores cresce rapidamente — e este é um dos fatores que determinaram a adoção da operação de substituição pelo formalismo [76] [75] — nada leva a crer que uma gramática equivalente a proposta em [92] para o português tenha menos de 1000 árvores.

Existe um analisador sintático estilo CKY¹⁶ [55], [93] com custo computacional de $O(n^6)$ no pior caso [88] definido para as TAGS. No entanto segundo [75] este também é o custo do melhor caso para este analisador. Além disso algumas restrições são feitas em relação às estruturas gramaticais que são pouco realísticas, tais como supor que um nodo possui no máximo dois filhos.

Dois analisadores estilo Earley [31], [32], e [6] foram apresentados em [75], um deles com custo computacional $O(n^6)$ e outro com custo $O(n^9)$, mas ambos têm com comportamento médio melhor, chegando a ser linear em alguns casos, variando de acordo com a gramática. Em [75] foi apresentado um limite superior $O(n^4)$ para TAGs não ambíguas. Também foram encontradas referências de analisadores sintáticos paralelos para TAGs em [64], [65]

¹⁶Cocke-Kasami-Younger.

e [79].

Independentemente do custo do analisador sintático pode-se aproveitar a lexicalização das estruturas para definir uma estratégia em dois passos para o processo de análise [75]:

- no primeiro passo o analisador seleciona o conjunto de estruturas correspondentes a cada palavra da sentença de entrada. Cada estrutura pode ser encarada como a codificação de um conjunto de regras. Uma palavra seleciona uma estrutura quando é sua âncora.
- no segundo passo, o analisador tenta verificar se as estruturas podem ser combinadas para montar uma solução bem formada. O analisador sintático tenta colocar as estruturas correspondentes aos argumentos nas estruturas correspondentes aos predicados e então adjuntar, se necessário, as estruturas auxiliares correspondentes aos adjuntos que as selecionam.

A primeira etapa seleciona a parcela da gramática que é relevante para a sentença de entrada, uma vez que apenas as estruturas associadas às palavras são selecionadas para o processamento. A quantidade de estruturas selecionadas depende da gramática, do número de entradas no léxico e do grau de ambigüidade lexical.

Na segunda etapa o analisador sintático é capaz de utilizar informações ascendentes¹⁷ não locais para guiar sua busca, uma vez que as estruturas selecionadas durante a primeira etapa incorporam as informações léxico-morfológicas das âncoras que as selecionam. Esta codificação é utilizada para limitar o número de combinações possíveis entre as estruturas.

Apesar de grande parte das técnicas de derivação para TAGs estar explicada em [75] o leitor também pode se referir a [77], onde um conjunto significativo de técnicas é resumido.

Sumário sobre TAGs

Apesar de não ser um formalismo essencialmente lingüístico, as TAGs são consideradas adequadas para o tratamento da língua natural tanto pelo seu poder de reconhecimento como pelo seu domínio de localidade estendido em relação às gramáticas livres de contexto.

Possuir poder de reconhecimento superior ao das gramáticas livres de contexto vem se tornando comum entre os formalismos para o processamento da língua natural. As TAGs possuem não apenas esta característica, mas também variantes que fornecem restrições na adjunção, aproximando-se das gramáticas de unificação (FB-TAG) e das abordagens lexicalistas (L-TAG), tornando o formalismo uma alternativa atraente para o PLN.

A natureza matemática da definição das TAGs também é um atrativo, assim como a disponibilização das provas e dos algoritmos de análise sintática. É curioso notar que

¹⁷Do inglês *bottom-up*.

as restrições locais, como proposto em [47], perderam destaque em relação às restrições na adjunção cujas características são mais lingüísticas do que propriamente matemáticas, deixando perceber a mudança de foco nos trabalhos sobre o formalismo.

No estágio atual a bibliografia sobre TAGs vem tratando tanto as *features* quanto a lexicalização como parte integrante do formalismo, sem fazer distinção entre as variantes e a TAG simples. Isto quer dizer que as FB-TAGs, L-TAGs e combinações como L-FB-TAGs são denominadas simplesmente TAGs.

2.3.2 Uma abordagem centrada na lingüística

Outra maneira de abordar o PLN é começar pelo estudo dos fenômenos lingüísticos, deixando o tratamento matemático para um segundo momento. As Bound Movement Grammars¹⁸ [58] possuem esta característica e são apresentadas como uma extensão das Extraposition Grammars (XGs) [67]. Estes dois formalismos (BMGs e XGs) baseiam-se em cláusulas lógicas e permitem o tratamento de várias movimentações de constituintes em sentenças do português, tais como, por exemplo, a topicalização e a formação do modo interrogativo. Ambos os formalismos estão embasados lingüisticamente na teoria de *Government and Binding* [17] e na teoria das barreiras (*Barriers*) [18].

Como as BMGs, a princípio, derivam das XGs, descreveremos como funcionam as XGs primeiro. Como ambas baseiam-se nas cláusulas lógicas das DCGs (*Definite Clause Grammars*), também apresentaremos um breve estudo sobre elas. Estas explicações serão precedidas por uma introdução à teoria de *Government-Binding* e das barreiras.

A teoria de *Government and Binding* assume um conjunto de premissas biológicas e sobre aprendizado que não serão discutidas aqui. Esta introdução tem como objetivo apresentar da forma mais simples possível os conceitos de “movimento”, “barreira” e “restrição” no contexto lingüístico que moldou a formulação das BMGs e XGs. Para uma discussão mais aprofundada sobre como a biologia influenciou na criação desta teoria remetemos o leitor a [69].

2.3.3 Introdução à *Government and Binding*

Government and Binding (GB) é uma teoria que tenta explicar o que significa conhecer uma língua. Uma de suas premissas é a existência de um módulo biológico que determina a capacidade inata de uma pessoa falar e compreender uma língua qualquer.

O funcionamento deste módulo lingüístico é regido por princípios e parâmetros que podem assumir um conjunto finito de valores. Cada combinação de valores destes parâmetros, associado a uma configuração dos princípios, instancia uma língua natural. O

¹⁸Também conhecidas como GMA — Gramáticas de Movimentação e Ancoragem.

conhecimento de um falante da língua inglesa, por exemplo, é tão somente uma configuração do módulo biológico com valores diferentes dos da configuração de um falante da língua portuguesa.

Isto implica em aceitar a existência de uma espécie de gramática universal codificada no módulo biológico; também implica em aceitar que a aplicação dos valores dos princípios e parâmetros instancia uma espécie de gramática superficial mais diretamente relacionada com a estrutura diretamente observável de uma língua natural e mais próxima do que tradicionalmente se descreve nas gramáticas normativas.

O conhecimento necessário para dominar uma língua é organizado em quatro níveis de representação: D-Structure (DS - estrutura profunda), S-Structure (SS - estrutura superficial), Logical Form (LF - forma lógica) e Phonological Form (PF - forma fonológica).

Nesta organização, a estrutura profunda abrange a representação “pura” da estrutura argumento-predicado das sentenças da língua e serve como ligação com o léxico. A forma lógica é utilizada para a interpretação semântica e resolve, por exemplo, problemas de quantificação e escopo. A forma fonológica trata da ligação entre a fonologia e a sintaxe. A estrutura superficial representa a forma sintática e é o ponto de interação entre os demais níveis. A relação destes níveis pode ser visto na figura 2.29.

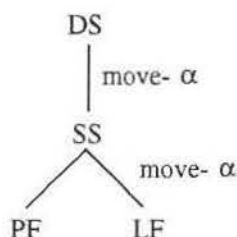


Figura 2.29: Organização do conhecimento necessário para instanciar uma língua natural segundo GB.

Cada domínio nesta organização possui um conjunto de regras que determina um nicho de competência, ou seja, é responsável por coordenar o funcionamento de um aspecto lingüístico tanto individualmente quanto em relação aos demais.

As representações existentes em DS e SS estão relacionadas por uma operação genérica de movimentação de constituintes chamada α -move que permite que qualquer um dos constituintes de DS seja deslocado para qualquer outra posição em SS ou vice-versa. As representações de SS e LF também estão relacionadas pelo α -move. Já entre SS e PF existem regras mais restritas de movimentação e remoção de constituintes [37].

A operação genérica α -move permite a movimentação de qualquer parte da estrutura que representa uma sentença para qualquer outra posição. Este movimento deixa no local original da estrutura uma marca referencial e pode ser restringido pelas regras gramaticais, ou seja, pelos princípios. Idealmente cada um destes princípios é simples e especifica

os requisitos das estruturas sintáticas aceitáveis através da manipulação das regras de governo (*government*), de c-comando e de ligação (*binding*) entre os itens. Mais tarde demonstraremos como os conceitos de movimentação evoluíram até a definição do *move- α* e como foram desenvolvidas as regras que restringem este movimento.

Uma das maneiras de definir a gramaticalidade de uma sentença é aceitar que ela é gramatical quando possuir representação em todos os quatro níveis e quando estas representações forem coerentes entre si e com as regras que restringem as transformações do *move- α* . Cada nível possui seus próprios princípios e parâmetros e a coerência entre as representações não necessariamente implica em homogeneidade.

Outro requisito de gramaticalidade é o *princípio de projeção*, que determina que os constituintes de uma sentença sejam a realização de sua representação no léxico. Esta projeção permite que as regras sintáticas observem as propriedades de subcategorização, impedindo a alteração ou eliminação arbitrária de constituintes.

As tentativas de descrever, mesmo que introdutoriamente, a teoria de *Government and Binding* em geral resultam numa grande quantidade de material. Para tentar evitar que esta parte introdutória acabe se tornando por demais extensa, optaremos por uma descrição evolutiva da teoria centrando a descrição nos pontos utilizados pelas BMGs. Mais adiante voltaremos a Teoria de *Government and Binding* com menos alusões biológicas e mais especificações lingüísticas.

2.3.4 Teoria Transformacional

As gramáticas transformacionais foram abordadas anteriormente sob o aspecto matemático quando tratamos dos sistemas de reescrita e das TAGs. Lingüisticamente falando, a Teoria Transformacional introduz a noção de transformação lingüística pela determinação de quais constituintes são comuns aos núcleos das frases, pela construção das estruturas frasais básicas e pela especificação de qual é o conjunto mínimo necessário destas estruturas para gerar as diferentes sentenças da língua aplicando-se uma ou mais transformações [78].

Uma gramática transformacional, ainda segundo [78], contém:

- *um componente de base:*

- regras sintagmáticas¹⁹: as gramáticas tradicionais distinguem dois tipos de categorias: lexicais (ex: N para nomes) e sintagmáticas (ex: SN para sintagma nominal); as regras sintagmáticas especificam a estrutura das frases em sintagmas e dos sintagmas em categorias lexicais (ex: $P \rightarrow SN\ SV$).

¹⁹Em textos de ciência da computação é usado o termo “sintático” como sinônimo de sintagmático. Isto nem sempre é adequado porque assim não se respeitam as diferenças de classificação lingüística dos componentes gramaticais, simplificando a categorização dos constituintes.

- um léxico: lista das palavras de uma língua; para cada palavra fornece diversas informações como, por exemplo, a categoria léxica, especificações morfo-sintáticas, informações sobre subcategorização, etc. Pode conter regras de redundância lexical e de reestruturação.
- uma regra de inserção lexical: indica como uma palavra do léxico pode ser manipulada utilizando-se as informações lexicais para ser inserida na árvore de derivação de uma frase que está sendo analisada.

Este componente gera as árvores sintáticas das estruturas profundas das frases (*D-Structures*).

- *um componente transformacional*

- as transformações são regras de reescrita de árvores que, aplicadas sobre as *D-Structures*, geram as estruturas superficiais cujos nodos de fronteira formam as sentenças de uma língua. Várias regras transformacionais podem ser aplicadas sobre uma mesma estrutura, tanto sobre seus constituintes quanto sobre a estrutura da árvore sintática que os contém.

As regras transformacionais obedecem ao esquema apresentado na figura 2.30.

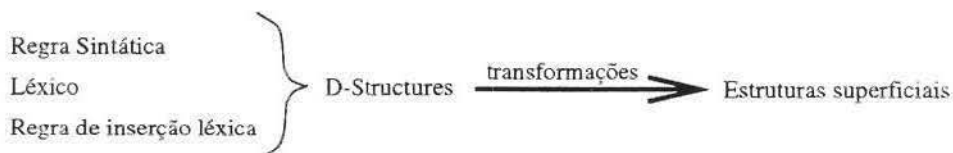


Figura 2.30: Esquema das regras transformacionais.

2.3.5 Sintaxe X-Barra

A sintaxe X-Barra [46], definida nos anos 70, permitiu resolver dois problemas inerentes à sintaxe sintagmática clássica:

- um número de categorias muito limitado: ausência de categorias intermediárias entre os objetos lexicais e os sintagmas que a experiência empírica provou existir. Em outras palavras, proveu princípios de subcategorização.
- uma sintaxe permissiva demais em relação ao formato das regras de formação de frases. Por exemplo, a regra $VP \rightarrow DetAdjN$ não pode ser rejeitada.

A partir da versão mais recente da teoria X-Barra, as principais categorias foram redefinidas como uma combinação de dois traços, o nominal (N) e o verbal (V). Uma estrutura nominal, ou seja, um substantivo ou um sintagma nominal, passou a ser codificado como $[+N, -V]$, ou simplesmente N, assim como um verbo ou sintagma verbal passou a ser codificado como $[-N, +V]$, ou simplesmente V. A combinação $[+N, +V]$ (ou apenas A) representa os adjetivos e advérbios enquanto $[-N, -V]$ (ou apenas P) representam os sintagmas preposicionais²⁰. Existem ainda mais duas categorias não lexicais: COMP (complemento) e INFL (inflexão: tempo, voz, etc.) que substituem o símbolo de sentença (S em geral).

Para cada categoria X existem categorias relacionadas: X^0 ou X (categoria ou papel de inserção lexical), X^1 ou $\bar{X}$, X^2 ou $\bar{\bar{X}}$, e assim por diante. O número de índices ou barras é controverso mas frequentemente fica entre 0 e 2. As categorias com barras ou com índices estritamente positivos são denominados projeções-barra (*bar-projection*) de X . A categoria com o maior número de barras ou seja, com o maior índice (pode-se supor que seja 2), é a projeção maximal de X .

Para cada categoria X , as regras de composição básicas são:

$$X^2 \rightarrow \text{spec}, X^1.$$

$$X^1 \rightarrow X^0, \text{cpt.}$$

$$X^1 \rightarrow \text{cpt}, X^0.$$

$$X^1 \rightarrow \text{adjunto}, X^1.$$

$$X^1 \rightarrow X^1, \text{adjunto.}$$

Aqui, *spec* é um especificador opcional de tipo X^2 (projeção maximal) específico de cada categoria. Por exemplo, para um determinante é um nome e para os auxiliares, um verbo; *cpt* é um complemento específico de cada categoria; *adjunto* é um adjunto de tipo X^2 ; em cada regra o X^i do lado direito é denominado centro da regra. Por exemplo:

$$\begin{array}{l} \text{INFL}^2 \rightarrow \text{N}^2 \text{INFL}^1 \\ \text{INFL}^1 \rightarrow \text{INFLV}^2 \end{array} \} \text{substituem } P \rightarrow \text{SN SV}.$$

A sintaxe X-Barra é interessante para o PLN porque permite especificar um conjunto de regras bastante restritivas com um alto grau de parametrização. Também permite compreender facilmente o desenrolar de uma análise. Fica evidente que nas gramáticas deste tipo o léxico assume um papel primordial.

2.3.6 A evolução do conceito de movimentação

A Teoria Transformacional apresenta um grande número de movimentações, aplicáveis ciclicamente, que complementam as regras de composição gerativa básicas. Como exemplo podemos citar as regras de formação de perguntas, da forma passiva, montagem do sujeito

²⁰Estas duas últimas combinações ainda geram debate, com alguns autores defendendo que $[+N, +V]$ deve representar os advérbios. Esta discussão, no entanto, foge ao escopo deste trabalho.

e topicalização. Estas movimentações são freqüentemente complicadas, cheias de detalhes e são específicas de uma língua natural em particular.

A sintaxe X-Barra, no entanto, permite que apliquemos estas movimentações sobre as regras composicionais básicas. De fato, os lingüistas tentam generalizar as movimentações originalmente especificadas e reagrupá-las de acordo com a subcategorização dos constituintes. Definir qual o conjunto final de transformações é um trabalho em eterno andamento, mas podemos enumerar algumas transformações obtidas no processo de generalização e reagrupamento:

Movimentação do SN: descreve como um SN pode ser deslocado dentro de uma frase. Esta transformação substitui a apassivação e a construção do sujeito de uma frase. Aplicando-a sobre a frase “parece que João come uma maçã” podemos obter “João parece comer uma maçã”. Este tipo de movimentação ocorre por *substituição*. O ramo adequado da árvore de derivação é substituído por uma sub-árvore e deixa na estrutura um *traço* e_i , que coindexa sua nova posição com o índice i (teoria dos traços). A figura 2.31 mostra um exemplo desta operação.

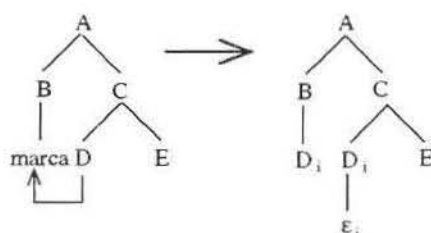


Figura 2.31: Substituição com coindexação.

A sentença “João parece comer uma maçã” surge após aplicarmos a movimentação do SN sobre a estrutura apresentada na figura 2.32.

Movimentação ‘qu’: descreve como construir uma sentença interrogativa a partir de uma sentença afirmativa. Para transformar “João come uma maçã” em uma interrogação é necessário transformar o sintagma em pronome e depois deslocar este pronome na frase. O resultado obtido “o que_i come João e_i?” apresenta uma inversão do verbo e de seu sujeito (e surge um traço de coindexação).

Este movimento é executado pela adjunção, à esquerda do COMP da frase, do pronome interrogativo. Aqui pode-se considerar dois tipos de adjunção: a clássica (adotada nas TAGs) e a definida por Chomsky. As diferenças entre ambas são mostradas na figura 2.33.

A figura 2.34 mostra um exemplo de movimentação ‘qu’.

A última etapa na evolução das transformações é apresentada na teoria de *Government and Binding* onde, como dissemos anteriormente, todas as movimentações são reduzidas à regra genérica *move- α* : “deslocar não importa qual constituinte, não importa para onde”. Aqui não existem mais transformações específicas para uma determinada categoria ou

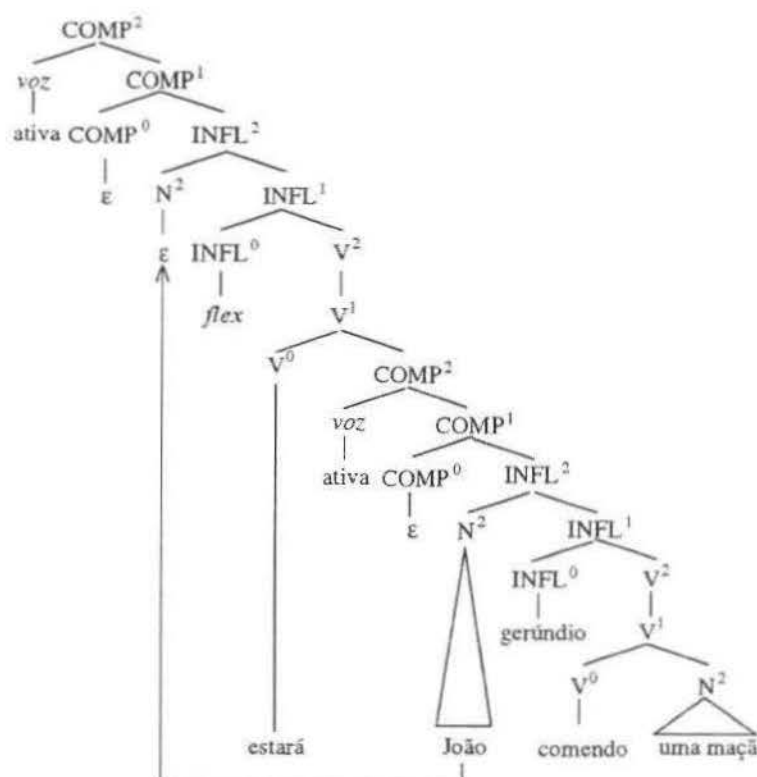


Figura 2.32: Movimentação de SN sobre estrutura original.

construção. Esse grau de generalismo do *move- α* torna necessária a introdução de algum princípio de aplicação como, por exemplo, unificação de traços (como nas FB-TAGs) ou restrições estruturais de movimentação, que veremos a seguir.

2.3.7 Restrições de movimentação

Uma restrição é uma condição que deve ser satisfeita pela representação sintática de uma frase. Estas restrições podem apresentar um aspecto matemático que orienta a derivação (as regras gerativas), como no caso das TAGs, ou podem assumir formatos mais relacionados com a lingüística, sem preocupação com o aspecto computacional.

Uma restrição pode definir *ilhas*, ou seja, subdomínios dentro de uma árvore de derivação de onde não se pode extrair elementos quando utilizamos as regras de movimentação. Se em uma etapa da derivação um constituinte C aparecer dentro de uma ilha e, após a aplicação das regras de movimentação, estiver fora dela, então a derivação resultante (estrutura superficial) estará mal formada.

O trabalho de dois lingüistas se destaca no estudo de restrições deste tipo: Ross [72] e Chomsky. A seguir apresentaremos as restrições de Ross, adaptadas para a sintaxe X-Barra por [78]. Trataremos do trabalho de Chomsky quando voltarmos a falar sobre a

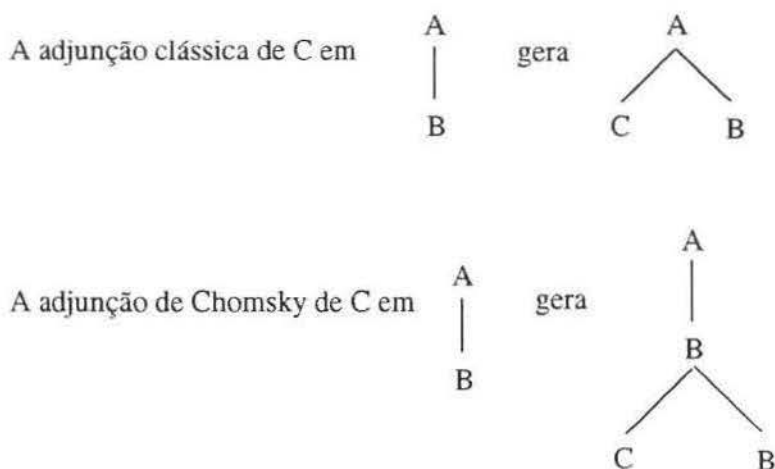


Figura 2.33: Dois tipos de adjunção.

teoria de *Government and Binding*.

A. Restrição para sintagmas nominais complexos

Expressão: não se pode deslocar constituintes de um $COMP^2$ para um N^2 quando este $COMP^2$ for dominado por este N^2 e precedido por um N^0 .

Esquema: veja a figura 2.35

Exemplo: a frase “eu creio que João tem este arquivo” pode ser transformada em “este arquivo eu creio que João tem”, *mas* “eu acredito no fato de João ter este arquivo” não pode ser transformado em “este arquivo eu acredito no fato de João ter” porque a frase inicial possui a árvore sintática equivalente a

eu creio [N^2 [N^1 no [N^0 fato]] $COMP^2$ de João ter este arquivo]]

B. Restrição sobre estruturas coordenativas

Expressão: nenhum dos termos de uma coordenação pode ser deslocado para fora da estrutura coordenativa, assim como nenhum componente destes termos individualmente, conforme esquematizado na figura 2.36.

Esquema:

Exemplo: “João está comendo uma maçã e uma pêra” não pode ser transformado em “que maçã o João está comendo e uma pêra”

C. Convenção de Pied-Piping

Expressão: toda transformação sobre um N_0^2 — precedido e seguido por variáveis de indexação estrutural que indiquem a adequação desta regra — pode ser aplicada a este N_0^2 e a todo N_i^2 não coordenado que o domine desde que não exista nenhuma coordenação e nenhum nodo $COMP^2$ no caminho que liga estes dois N^2 , conforme esquematizado na figura 2.37.

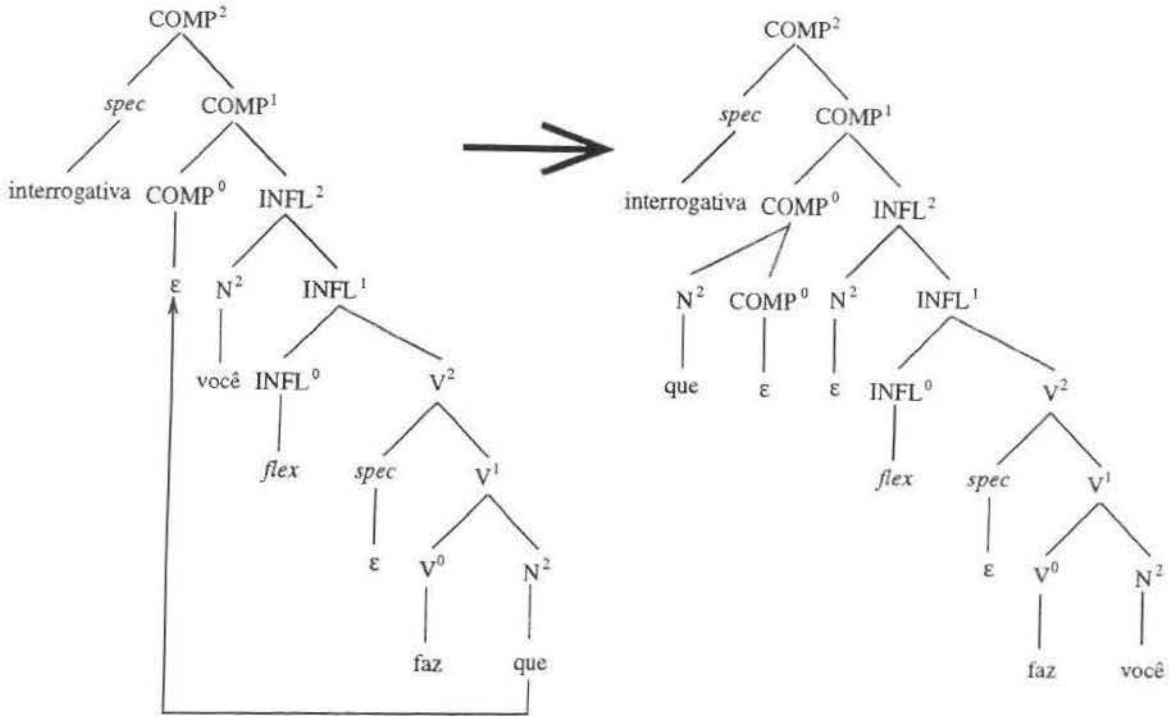


Figura 2.34: Exemplo de movimentação 'qu'.

Esquema:

Exemplo: "Eu protegi o chapéu da goteira":

→ "a goteira da qual eu protegi o chapéu"

→ "o chapéu que da goteira eu protegi".

A figura 2.38 mostra que o N² "a goteira" pode sofrer extraposição, assim como o N² "o chapéu da goteira" porque não existe nenhum nodo coordenativo e nenhum nodo COMP² entre os dois nodos N².

A presença de um nodo coordenativo impediria a transformação, como no exemplo "eu protegi o chapéu e a pasta da goteira":

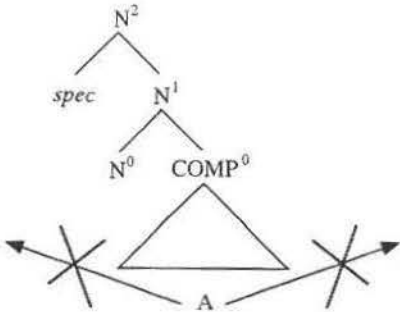


Figura 2.35: Restrição para sintagmas nominais complexos.

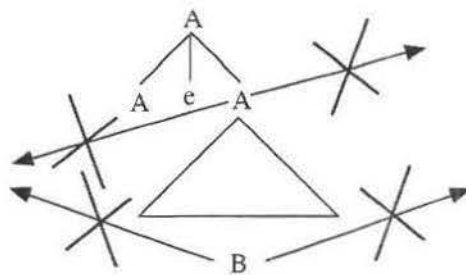


Figura 2.36: Restrição de termos coordenados.

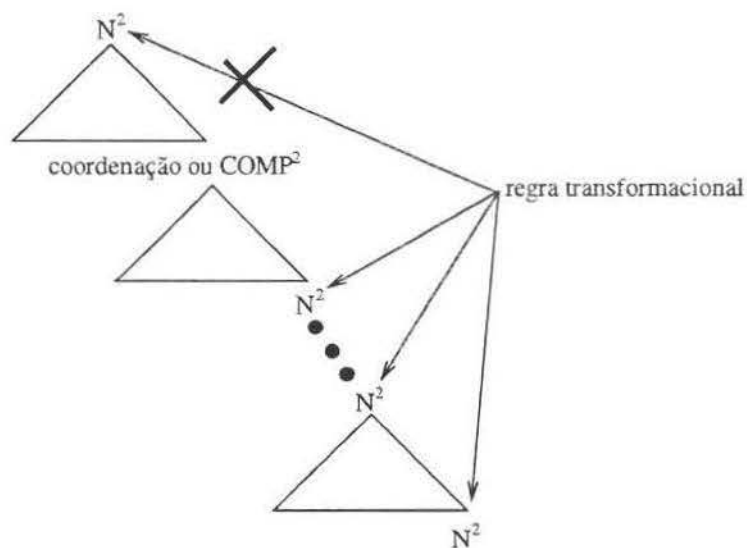


Figura 2.37: Esquemática do "pied-piping".

→ "a goteira da qual eu protegi o chapéu e a pasta" (gramatical);

→ "o chapéu que da goteira eu protegi e a pasta" (agramatical).

D. Restrição sobre a projeção mais à esquerda

Expressão: não se pode retirar de um N^2 o grupo nominal mais à esquerda que ele domina.

Esquema:

Exemplo: "Eu admiro a coragem dos filhos do mar"

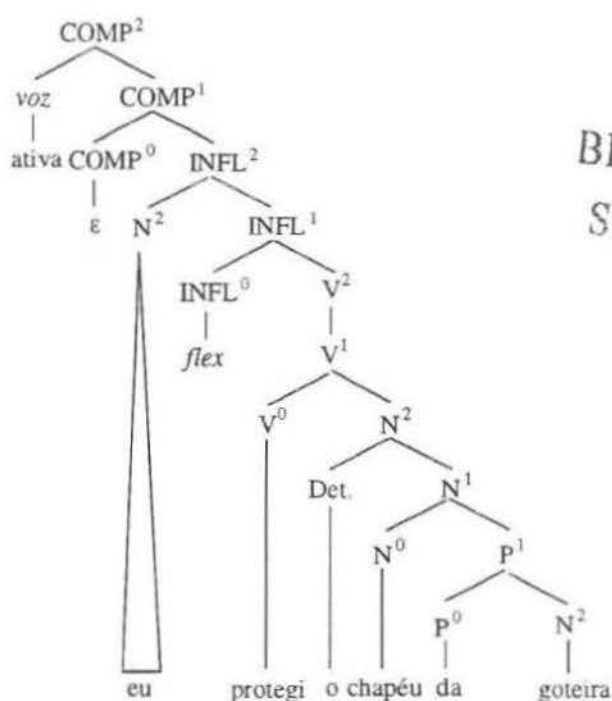
→ "a coragem que eu admiro dos filhos do mar" (agramatical);

→ "a coragem dos filhos que eu admiro do mar" (agramatical);

→ "a coragem dos filhos do mar que eu admiro" (gramatical);

→ "dos filhos do mar eu admiro a coragem" (gramatical).

A figura 2.39 nos mostra de forma esquematizada como a projeção nominal mais à esquerda, que no caso do exemplo é "a coragem", fica impedida de deixar a sub-árvore dominada pelo N^2 . Isto não ocorre com a projeção mais à direita, que no caso é "dos



UNICAMP
BIBLIOTECA CENTRAL
SEÇÃO CIRCULANTE

Figura 2.38: Exemplo de Pied-Piping.

filhos do mar”.

E. Restrição sujeito-frase

Expressão: nenhum constituinte dominado por um $COMP^2$ pode ser movimentado para fora deste $COMP^2$ se este for dominado por um N^2 e se este N^2 for imediatamente dominado pela estrutura da figura 2.40.

Exemplo: “o livro que eu comprei era interessante”:

→ “eu o livro que comprei era interessante” (agramatical);

→ “comprei o livro que eu era interessante” (agramatical);

→ “eu comprei o livro que era interessante” (gramatical mas com sentido alterado).

Estas movimentações se aplicam apenas aos constituintes dominados pelo $COMP^2$

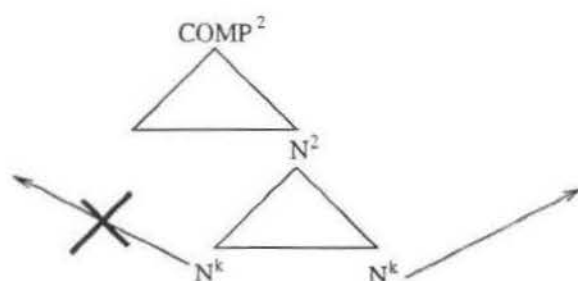
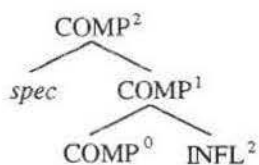


Figura 2.39: Restrição sobre projeção nominal mais à esquerda.

Figura 2.40: Estrutura que domina N^2 na restrição sujeito-frase

e não impedem movimentações mais complexas. Uma destas movimentações pode ser observada na sentença “eu comprei o livro, que era interessante”. A figura 2.41 nos mostra a estrutura da sentença exemplo e destaca o COMP² cujos subconstituintes não podem ser deslocados para fora; note como esta configuração é similar a uma ilha de onde os componentes não podem ser retirados.

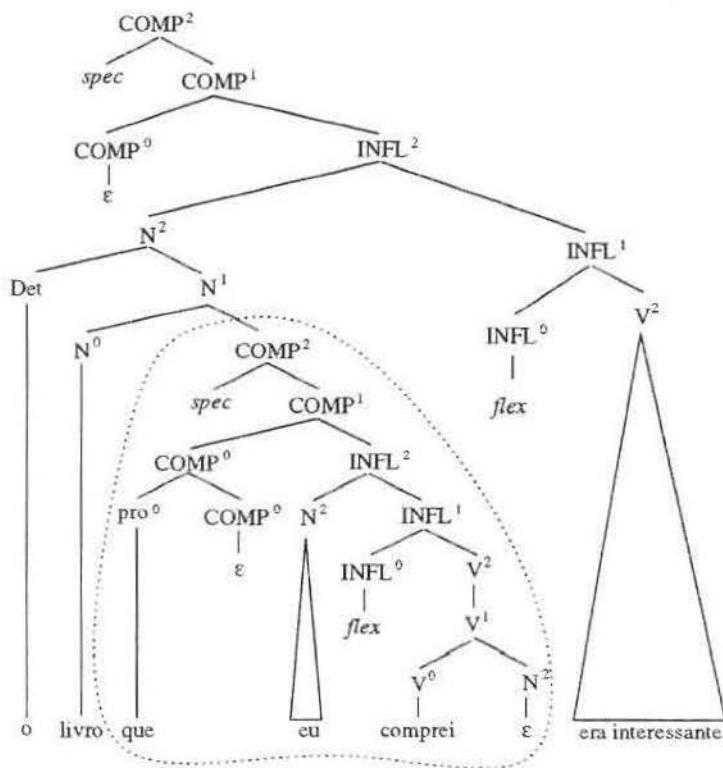


Figura 2.41: Exemplo de restrição sujeito-frase

Posteriormente Chomsky introduziu novas restrições e algumas considerações genéricas como, por exemplo, explicações de porque um determinado constituinte pertence a uma ilha e não à outra. Algumas restrições definidas por Chomsky são a restrição de subjacência e as restrições sobre os complementos [17], [18].

Transformações podem ser aplicadas sucessivamente sobre uma estrutura. Um constituinte deslocado deixa em sua posição original um traço que coindexa sua nova posição;

se for deslocado novamente cria-se então uma *cadeia de coindexação*. Cada transformação deve ser analisada independentemente e satisfazer todas as restrições.

Government and Binding como a última evolução das movimentações

As últimas evoluções da teoria transformacional permitem uma grande economia quanto ao número de regras que descrevem as frases de uma língua. Na teoria de *Government and Binding* [17], [18] as regras de composição gerativa são apenas cinco e as transformações se reduzem a apenas um movimento (*move- α*). A estas regras se soma um pequeno número de princípios gerais que filtram as frases agramaticais aceitas devido à grande generalidade do *move- α* .

Segundo a GB a gramaticalidade de uma frase é decidida quando se obtém uma estrutura anotada de superfície bem formada — como por exemplo $[COMP^2 \dots [N^2 PRO_i t_i] \dots]$. Nesta lógica, *move- α* pode ser visto como uma função de coindexação entre o elemento deslocado e sua posição antes da movimentação. Isto quer dizer que podemos deixar de encarar a transformação como um movimento de verdade e passar a tratá-la como uma relação entre constituintes distantes.

Usar os resultados obtidos pela GB no tratamento automático de línguas naturais traz algumas vantagens: a teoria é bastante clara (se bem que alguns aspectos em relação à ordenação ainda precisam de maiores esclarecimentos), tem um grande poder expressivo; além disso é modular, concisa, declarativa e possui alto grau de parametrização.

Ao contrário da teoria transformacional clássica, GB não define regras específicas para cada tipo de movimento. Estes movimentos, por adjunção ou substituição como vimos anteriormente, resultam da aplicação da regra única *move- α* e da interação de vários princípios e teorias que limitam seu poder e rejeitam as frases incorretas.

Algumas das teorias relacionadas com a restrição de movimento são a própria teoria X-Barra, *Case Theory*, *Theta Theory*, *Movement Theory*, *Government Theory*, *Binding Theory* e o *Extended Projection Principle*. A explicação de como estes diversos formalismos lingüísticos contribuem na verificação da gramaticalidade e restringem o *move- α* constitui grande esforço e pouca contribuição para compreendermos as BMGs, isto porque não existe vinculação formal entre as BMGs e estes formalismos.

2.3.8 Extraposition Grammars — XG

As Extraposition Grammars foram apresentadas em [67] e estendem o poder de reconhecimento das gramáticas livres de contexto com o intuito de tratar adequadamente um fenômeno lingüístico chamado *extraposição à esquerda*²¹.

²¹Left extraposition.

A extraposição à esquerda é um recurso gramatical utilizado para descrever a formação de sentenças interrogativas e cláusulas relativas em línguas ocidentais, tais como o Português, Francês, Inglês e Espanhol. Este fenômeno ocorre quando um subconstituente de um constituinte está ausente e algum outro constituinte, à esquerda do que está incompleto, representa o subconstituente que falta de alguma maneira [67].

Pode-se imaginar que um tipo de constituinte vazio, chamado traço, ocupa o “buraco” deixado pelo subconstituente ausente e que o constituinte à esquerda, que representa a parte perdida, é um marcador, indicando que o constituinte à sua direita contém um traço. O constituinte em cujo lugar está o traço é dito extraposto para a esquerda e, em sua nova posição, é representado pelo marcador. As cláusulas relativas são formadas por um marcador, que nos casos mais simples é apenas um pronome relativo, seguido de alguma sentença onde um sintagma nominal foi substituído por um traço. A sentença “O homem que_i [_S João viu t_i] era manco”, por exemplo, exhibe este comportamento.

O conceito de extraposição para a esquerda é essencial, direta ou indiretamente, para muitas descrições formais de cláusulas interrogativas ou relativas. Na descrição deste conceito existem muitas restrições globais e “restrições de ilhas” especificando exatamente onde ele pode ocorrer [67]. Por exemplo, o conceito de restrição de Ross para sintagmas complexos [72], que vimos anteriormente, implica que nenhum pronome relativo que ocorra fora de um dado sintagma nominal pode ser referido por um traço que ocorra dentro de uma cláusula relativa que é subconstituente deste sintagma nominal. Isto impossibilita configurações como, por exemplo,

$$X_1 \dots [_{np} [_{rel} X_2 [_{S} \dots f_2 \dots t_1 \dots]] \dots] .$$

Tomemos como ponto de partida a XG da figura 2.42, que está sem seus terminais para simplificar:

```

sentença → sintagma_nominal, sintagma_verbal
sintagma_nominal → determinante, nome, cláusula_relativa
sintagma_nominal → traço
cláusula_relativa → [ ]
cláusula_relativa → marcador_de_relativo, sentença
marcador_de_relativo ... traço → pronome_relativo

```

Figura 2.42: Uma XG para cláusulas relativas simples.

A construção básica das regras é similar à das *Definite Clause Grammars* (DCGs) [68] e utiliza o mesmo sistema de cláusulas lógicas (*definite clauses*) para descrever as informações lingüísticas. Estas cláusulas podem assumir a forma

$$P : -Q_1, \dots, Q_n.$$

que pode ser lida como²² “ P é verdadeiro se $Q_1, \dots, Q_n$ são verdadeiros”, ou a forma

$$P.$$

que pode ser lida como “ P é verdadeiro”. P é a cabeça da cláusula e $Q_1, \dots, Q_n$ são os objetivos que formam o corpo da cláusula. Os símbolos $P, Q_1, \dots, Q_n$ são literais. Um literal possui um símbolo predicado e possivelmente alguns argumentos (entre parênteses, separados por vírgulas), como em

$$\text{pai}(X,Y) \quad \text{falso} \quad \text{número}(0).$$

Um literal deve ser interpretado como uma relação entre seus argumentos; isto quer dizer que “ $\text{pai}(X,Y)$ ” denota a relação “pai” entre X e Y .

Argumentos são termos, ou melhor, objetos parcialmente especificados. Os termos podem ser:

- variáveis, significando objetos não especificados (os nomes das variáveis começam com maiúsculas):

$$X \quad \text{Case} \quad \text{Agreement}$$

- símbolos atômicos, significando objetos específicos:

$$\text{plural} \quad [] \quad 3$$

- termos compostos, que representam objetos complexos:

$$s(\text{SN}, \text{SV}) \quad \text{suc}(\text{suc}(0))$$

Um termo composto possui um funtor e alguns argumentos que são termos. Termos compostos podem ser melhor visualizados na forma de árvores, como na figura 2.43.

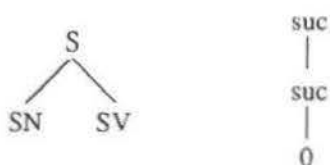


Figura 2.43: Termos compostos vistos como árvores

Um tipo particular de termo — a lista — é bastante utilizado e merece uma notação simplificada. O funtor binário ‘.’ constrói listas não vazias e o átomo ‘[]’ representa a lista vazia. Na notação simplificada,

$$[a, b] \quad [X|Y]$$

²²As XGs definem um operador “...”, o que nos leva a adotar “...” para representar uma sequência de termos.

representa respectivamente os termos

$$.(a,.(b,[])) \quad .(X,Y)$$

Nesta notação, a cláusula

$$\text{avô}(X,Z):- \text{pai}(X,Y), \text{pai}(Y,Z).$$

pode ser lida como “X é avô de Z se X é pai de Y e Y é pai de Z”; já a cláusula

$$\text{pai}(\text{joão}, \text{maria}).$$

pode ser lida como “João é pai de Maria” (o uso de minúsculas é intencional e pretende denotar constantes).

Um conjunto de cláusulas define um programa. Um programa define as relações denotadas pelos predicados que são cabeças das cláusulas. Um programa deste tipo pode ser implementado por um interpretador de cláusulas, como o PROLOG [73], através de um processo de conversão [68], [67].

No exemplo da figura 2.42 podemos notar que a última regra não é a representação de uma regra livre de contexto nem de uma cláusula DCG. A intenção do operador “...” é permitir a representação da extraposição em cláusulas relativas e aproveitar as informações obtidas quando do processamento do marcador na análise do traço. A regra do exemplo manipula cláusulas relativas simples dizendo que um pronome relativo deve ser analisado como um marcador, seguido por um conjunto desconhecido de constituintes seguidos por um traço. A figura 2.44 nos permite observar como esta informação é utilizada; mais tarde veremos como é processada uma sentença.

No exemplo da figura 2.44 o sintagma nominal “a telha” sofreu extraposição e foi representado por um traço que, ao invés de ser reescrito como uma cadeia de caracteres vazia, guarda informações da análise do marcador.

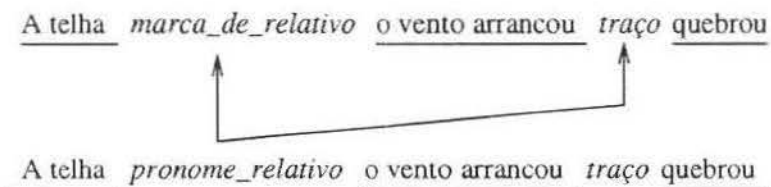


Figura 2.44: Aplicação de regra XG para cláusulas relativas simples

Outra diferença em relação às DCGs e GLCs é que o lado esquerdo de uma regra XG pode conter muitos elementos. Estes elementos são “derivados em conjunto, expandindo vários símbolos não contíguos em uma cadeia de caracteres” [67]. Alguns trabalhos

apontam que a capacidade de reconhecimento das XGs (e conseqüentemente as BMGs) é suficiente para o reconhecimento de qualquer conjunto recursivamente enumerável [40].

A forma geral de uma regra XG é

$$s_1 \dots s_2 \dots s_{k-1} \dots s_k \longrightarrow r.$$

onde cada segmento s_i (separado dos demais por $\dots$) é uma seqüência de terminais e não terminais escritos na forma das DCGs com a vírgula ',' sendo usada para concatenação. O primeiro símbolo em s_1 deve ser um não terminal. O lado direito da regra (' r ') é uma regra DCG.

Quando muitas regras XG estão envolvidas, a derivação de uma sentença se torna mais complicada que no exemplo anterior. Isto porque a aplicação de uma regra interage com as demais através dos marcadores de "ilhagem" '<' e '>'. Para representar os estágios intermediários de uma derivação XG definiu-se as cadeias de caracteres ilhadas (*brackets strings*). Estas cadeias de caracteres são formadas por:

- símbolos terminais;
- símbolos não terminais;
- símbolo de menor (<); e
- símbolo de maior (>).

Uma cadeia de caracteres ilhada está balanceada se os símbolos de maior ('>') e de menor ('<') estiverem balanceados da mesma maneira que estariam se fossem parênteses. Desta forma, uma regra XG

$$u_1 \dots u_2 \dots \dots u_n \longrightarrow v.$$

pode ser aplicada a uma cadeia de caracteres ilhada s se

$$s = x_0 u_1 x_1 u_2 \dots x_{n-1} u_n x_n.$$

e cada um dos intervalos $x_1, \dots, x_{n-1}$ estiver balanceado. A subcadeia de caracteres de s entre x_0 e x_n é o intervalo²³ da aplicação da regra. Numa derivação, s é reescrita como t substituindo-se u_1 por v seguido por $n - 1$ símbolos de menor (<) e substituindo cada $u_2, \dots, u_n$ por um sinal de maior (>). Outra forma de representar este processo é dizer que s é substituído por

$$x_0 v << \dots < x_1 > x_2 > \dots x_{n-1} > x_n.$$

²³Do inglês *span*.

A relação entre a cadeia de caracteres original s e a derivada t pode ser abreviada como $s \Rightarrow t$. Na nova cadeia de caracteres t , a subcadeia de caracteres entre x_0 e x_n é o resultado da substituição. Em particular, a aplicação de uma regra com apenas um elemento no lado esquerdo não é diferente do que seria numa gramática de tipo 0.

Tomando a regra

marcador_de_relativo ... traço $\rightarrow$ pronome_relativo.

sua aplicação sobre

marcador_de_relativo *João viu* traço

produz

pronome_relativo < *João viu* > .

Após a aplicação desta regra, não é possível aplicar nenhuma regra sobre um segmento que esteja parte dentro, parte fora de uma ilha delimitada por '<' e '>'. A aplicação desta regra dividiu a sentença em duas partes isoladas que devem ser analisadas independentemente.

Dada uma XG com símbolo inicial s , a sentença t pertence a linguagem por ela definida se existir uma seqüência de aplicação de regras que transforme s em $\bar{t}$, e $\bar{t}$ possa ser transformada em t removendo-se todos os símbolos de maior e de menor.

O efeito produzido pelas restrições de “ilhas” (*bracketing constraints*) independe da ordem com que as regras são aplicadas [67], [40]. Isto porque, se duas regras são usadas numa derivação, os símbolos '<' e '>' introduzidos por cada uma delas devem ser compatíveis com a definição de restrição dada. À medida que estes símbolos são inseridos e nunca retirados fica claro que a ordem de aplicação das regras é irrelevante. Por razões similares, quaisquer duas aplicações de regra numa derivação onde estas regras possuam mais de um segmento em seu lado esquerdo, uma e apenas uma das seguintes possibilidades se aplica:

- o intervalo de nenhuma delas tem interseção com o da outra; ou
- o resultado de uma das aplicações está completamente contido em um intervalo da outra aplicação, ou seja, as aplicações estão encaixadas.

Da mesma forma com que árvores são utilizadas para visualizar a derivação de sentenças nas gramáticas livre de contexto, grafos de derivação são usados nas XGs. Nestes grafos, cada nodo corresponde a uma aplicação de regra ou a um símbolo terminal da sentença derivada. Os arcos que deixam este nodo correspondem aos símbolos do lado direito da regra e os arcos que apontam para o nodo — que podem ser vários — correspondem a cada um dos símbolos do lado esquerdo da regra a qual o nodo corresponde.

Dos arcos que representam o lado esquerdo da regra, somente o correspondente ao símbolo cabeça é utilizado para definir a ordem (da esquerda para direita) dos símbolos da sentença cuja derivação é representada pelo grafo. Se for mantido apenas o primeiro arco de entrada de cada nodo do grafo e retirados todos os demais a representação resultante é análoga a uma árvore de derivação.

A topologia dos grafos de derivação refletem a restrição de “ilhagem”. Vamos assumir a seguinte convenção para desenhar um grafo de derivação [67]:

- os arcos entrando em um nodo são ordenados no sentido horário seguindo a sequência de símbolos que correspondem ao lado esquerdo da regra a que o nodo corresponde; e
- os arcos que partem de um nodo são ordenados no sentido anti-horário seguindo a sequência de símbolos do lado direito da regra a que o nodo corresponde.

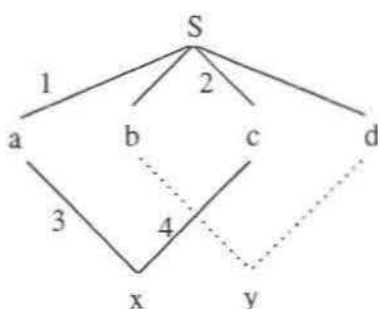


Figura 2.45: “Ilhagem” de componentes por bloqueio

O grafo obedece a restrição de ilhagem se e somente se puder ser desenhado, seguindo esta convenção, sem cruzamento de arcos [67]. O desenho da figura 2.45 mostra isto claramente. Nesta figura o ciclo formado pelos arcos 1, 2, 3 e 4 tem o mesmo efeito de uma ilha. Também vale notar que as regras aninhadas podem ocorrer no grafo de derivação como exemplificado na figura 2.46.

Limitações das XGs

A capacidade de reconhecimento de um determinado formalismo é um fator essencial para sua adequação ao PLN. Não se pode deixar, entretanto, de analisar a qualidade das descrições lingüísticas por ele geradas. As BMGs foram propostas exatamente pela incapacidade das XGs em descrever algumas relações lingüísticas e, em alguns casos, por fornecer relações incorretas. Para que compreendamos como estas associações podem ocorrer, devemos compreender o mecanismo de análise sintática do formalismo.

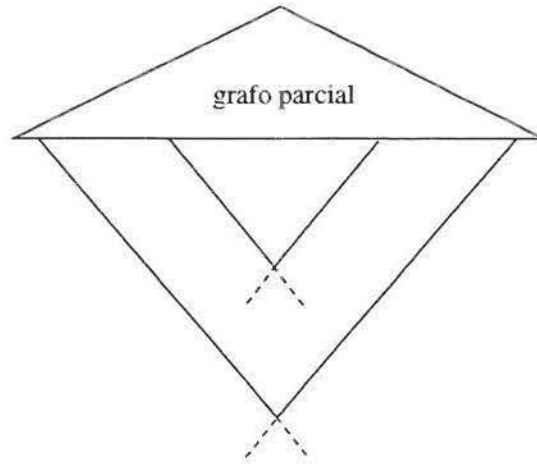


Figura 2.46: Aninhamento de derivações em uma XG

Uma XG é uma tupla (N, T, S, P) onde N e T são conjuntos disjuntos de não terminais e terminais, $S \in N$ é o símbolo inicial e P é um conjunto finito de produções na forma:

1. $A \rightarrow X_1 X_2 \cdots X_n$.
2. $A_1 \dots A_2 \dots \dots A_{n-1} \dots A_n \rightarrow X_1 X_2 \cdots X_n$.

onde $A_i \in N \wedge X_i \in (N \cup T)$.

Uma derivação em uma XG é definida sobre uma sequência “ilhada” $\bar{\alpha} \in (N \cup T \cup \{<, >\})^*$ de símbolos terminais e não terminais. Sejam $\bar{\alpha}, \bar{\beta}$ e $\bar{\gamma}$ sequências deste tipo e P um conjunto que contenha regras de tipo 1 e 2, então teremos respectivamente:

1. $\bar{\alpha} A \bar{\beta} \Rightarrow \bar{\alpha} X_1 X_2 \cdots X_n \bar{\beta}$.
2. $\bar{\alpha} A \bar{\gamma} B \bar{\beta} \Rightarrow \bar{\alpha} X_1 X_2 \cdots X_n [\bar{\gamma}] \bar{\beta}$.

desde que os símbolos de maior e menor de $\bar{\gamma}$ estejam balanceados. Uma gramática (ou programa) G deriva uma cadeia de caracteres w se existir uma sequência “ilhada” $\bar{w}$ de w de forma que $S \xrightarrow{*} \bar{w}$.

O grafo de derivação é uma maneira intuitiva de mostrar como as regras elípticas (que possuem o operador $\dots$) relacionam um elemento extraposto com seu traço. Um grafo de derivação observa as restrições de ilhagem quando possuir uma representação *top-down*, ordenada da esquerda para direita e planar, ou seja, quando puderem ser desenhados de acordo com a convenção dada sem cruzar arcos.

Computacionalmente falando, verificar se um grafo qualquer é planar não é exatamente eficiente. O processo de análise mais prático²⁴ é se utilizar um provador de cláusulas —

²⁴Sugerido em [67] e bastante difundido.

PROLOG, por exemplo — para a derivação e abordar o espaço de busca em profundidade, com estratégia *top-down* e derivação mais à esquerda, utilizando listas para simular uma estrutura de pilha auxiliar.

Tomemos a seguinte XG²⁵:

$s \rightarrow as, bs, cs.$

$as \rightarrow \epsilon.$

$as \dots xb \rightarrow [a], as.$

$bs \rightarrow \epsilon.$

$bs \dots xc \rightarrow xb, [b], bs.$

$cs \rightarrow \epsilon.$

$cs \rightarrow xc, [c], cs.$

Ela descreve a linguagem de todas as cadeia de caracteres com formato $a^n b^n c^n$ para todo $n \geq 0$, que é estritamente sensível ao contexto [67]. Primeiro mostraremos uma derivação usando uma técnica simplificada, específica para este exemplo. Trocaremos a derivação mais à esquerda pela derivação mais a direita e manteremos as cadeias de caracteres intermediárias balanceadas e a pilha auxiliar vazia. Um não-terminal cuja derivação resulte em vazio será representado por “[]” para facilitar a visualização.

$S \rightarrow as \ bs \ cs.$

$\rightarrow as \ bs \ xc \ [c] \ cs.$

$\rightarrow as \ bs \ xc \ [c] \ xc \ [c] \ cs.$

$\rightarrow as \ bs \ xc \ [c] \ xc \ [c] \ [].$

$\rightarrow as \ xb \ [b] \ bs \ <> \ [c] \ xc \ [c] \ []$

$\rightarrow as \ xb \ [b] \ xb \ [b] \ bs \ <<> \ [c] \ > \ [c] \ []$

$\rightarrow as \ xb \ [b] \ xb \ [b] \ [] \ <<> \ [c] \ > \ [c] \ []$

$\rightarrow [a] \ as \ <> \ [b] \ xb \ [b] \ [] \ <<> \ [c] \ > \ [c] \ []$

$\rightarrow [a] \ [a] \ as \ <<> \ [b] \ > \ [b] \ [] \ <<> \ [c] \ > \ [c] \ []$

$\rightarrow [a] \ [a] \ [] \ <<> \ [b] \ > \ [b] \ [] \ <<> \ [c] \ > \ [c] \ []$

$\rightarrow aabbcc$

Note que os traços esperados pelos operadores ‘...’ já estão disponíveis imediatamente nas cadeias de caracteres intermediárias porque a derivação se deu à direita e não à esquerda. Por causa desta disponibilidade, os traços são retirados da cadeia de caracteres à medida em que se fazem necessários e os marcadores ‘<’ e ‘>’ aparecem sempre aos pares, de forma balanceada.

Vejamos agora como o mecanismo genérico trabalharia a mesma derivação, usando a pilha auxiliar, com derivação à esquerda:

²⁵Note que ‘as’ não é a concatenação dos símbolos ‘a’ e ‘s’, mas apenas uma variável. A concatenação de ‘a’ e ‘s’ seria notada como ‘a,s’.

$S \rightarrow as\ bs\ cs$	
$\rightarrow [a]\ as\ <\ bs\ cs$	$[xb_1—]$
$\rightarrow [a]\ [a]\ as\ <<\ bs\ cs$	$[xb_2—xb_1]$
$\rightarrow [a]\ [a]\ []\ <<\ bs\ cs$	$[xb_2—xb_1]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ bs\ <\ cs$	$[xc_1—xb_1]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ bs\ <>\ [c]\ cs$	$[xb_1—]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ >\ [b]\ bs\ <<>\ [c]\ cs$	$[xc_2—]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ >\ [b]\ []\ <<>\ [c]\ cs$	$[xc_2—]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ >\ [b]\ []\ <<>\ [c]\ >\ [c]\ cs$	$[—]$
$\rightarrow [a]\ [a]\ []\ <<>\ [b]\ >\ [b]\ []\ <<>\ [c]\ >\ [c]\ []$	$[—]$
$\rightarrow aabbcc$	

Esta técnica é mais fiel à definição original do processo, onde se diz que o processamento se dá pelas cabeças das regras, enquanto os elementos posteriores ao operador “...” são colocados “de lado” esperando por sua realização. Na verdade, a definição original dada para as XGs é informal, quase uma mímica do processamento de cláusulas da esquerda para direita, em profundidade e com derivação mais à esquerda.

Esta organização permite que o formalismo reconheça sentenças como “João disse que vai fazer o trabalho ontem”, que exige alguma atenção de um leitor humano, seja tratada corretamente. Por outro lado, sentenças como “A pessoa que a uma amiga deu um livro desapareceu” gera um problema: mesmo que através de alguns artifícios²⁶ se possa escrever uma gramática capaz de reconhecer a sentença, o mecanismo de análise associa erroneamente “A pessoa” a “que a uma amiga” (sem sentido) ao invés de “que deu um livro” (o sentido pretendido).

Esta associação ocorre porque a estrutura utilizada para guardar as informações linguísticas colhidas durante a análise da sentença — uma pilha — não é adequada para o tratamento de referências cruzadas, apenas de referências encaixadas. O analisador sintático sempre força a associação encaixada, mesmo quando tentamos manipular as restrições de ilhas.

A proposta das BMGs

As BMGs foram propostas para superar as dificuldades encontradas pelas XGs e ampliar a gama de fenômenos linguísticos passíveis de tratamento. Além da extraposição à esquerda, foram considerados neste formalismo fenômenos como a formação de interrogativas, topicalização, clíticos e cláusulas relativas, além da restrição do escopo dos movimentos

²⁶Movimentando-se as restrições de ilha para cercar apenas símbolos terminais e artificialmente posicionando, ou suprimindo, as regras com o operador “...” [40].

permitidos [58]. Um aspecto elegante deste formalismo é a capacidade de tratar um conjunto de transformações, não apenas um tipo, sem efetivamente deslocar os constituintes durante a análise, apenas guardando as referências de maneira adequada.

Vimos que as XGs não conseguem tratar alguns fenômenos lingüísticos devido a sua estrutura de armazenamento baseada em pilha. As duas alternativas de expansão mais imediatas são a troca da estrutura de pilha por algo mais poderoso (e complexo) ou a proliferação de estruturas, isto é, o uso de mais de uma pilha ao mesmo tempo. As BMGs adotam a segunda técnica —ou seja, a multiplicação de estruturas— com mais ênfase, mas também redefinem a estrutura de pilha como diferenciação de listas. Apesar dos exemplos encontrados na literatura utilizarem estas listas como pilhas, não existe uma obrigação formal para isto.

O problema que surge ao multiplicarmos estruturas de armazenamento é decidir qual a estrutura de dados onde se deve armazenar e mais tarde buscar as informações. Para evitar este problema, as BMGs especificam que cada uma das estruturas de dados pode armazenar informações de apenas um tipo de movimento. Isto permite identificar quais regras tratam um determinado fenômeno verificando qual estrutura manipulam mas, por outro lado, obriga que se saiba exatamente onde são armazenadas, e posteriormente buscadas, as informações da análise.

A esta multiplicação de estruturas corresponde o surgimento de diferentes tipos de barreiras que substituem ' $<$ ' e ' $>$ ' e impedem a correferência imprópria de material lingüístico movimentado. Estas barreiras, ou ilhas, são especificadas nas regras através de identificadores unívocos que substituem o operador ' $\dots$ '.

O lado esquerdo de regra BMG pode ser igual ao de uma regra DCG, apenas um não terminal ou, devido a estas modificações, assumir a forma

NT Movimento Cat

em substituição à notação da regra " $Cat_1 \dots Cat_2 \rightarrow$ " das XGs. Neste caso *NT* representa um não terminal, *Cat* um terminal ou não terminal e *Movimento* é o nome da estrutura de dados que vai guardar as informações.

Pode-se verificar por esta especificação que a BMG é mais uma nova generalização das DCGs do que propriamente uma expansão das XGs. Isto porque a definição de lado esquerdo de regra não permite uma sequência de símbolos, como ocorre nas XGs.

Apesar do número de estruturas de armazenamento depender apenas do usuário, sem limites impostos pelo formalismo, cada uma delas está relacionada com um fenômeno lingüístico diferente, acompanhando o número de movimentações tratadas. Um dos possíveis conjuntos de valores que pode ser assumido é [58]:

relativo: Se *NT* puder ser classificado como uma cabeça de cláusula relativa (um pronome relativo ou uma preposição seguida de pronome relativo);

*corte*²⁷: se *NT* for uma categoria sintática que pode ser topicalizada (um sintagma preposicional, uma cláusula subordinada, um núcleo verbal ou um sintagma verbal); e

interrogativo: se *NT* puder ser categorizado como um sintagma nominal interrogativo ou preposicional.

O lado direito de uma regra BMG é composta por uma regra DCG prefixada por barreiras. Cada um dos tipos de barreiras tem um nome e está diretamente relacionada com uma das estruturas de dados de armazenamento. A forma geral do lado direito de uma regra BMG é

Tipo_de_barreira *NT*

onde *NT* é um não terminal. Para o conjunto de estruturas dado anteriormente, o *Tipo_de_barreira* pode assumir os valores *ilha*²⁸, *ilha_corte*, *ilha_relativa* e *ilha_interrogativa*.

Um termo com o formato *ilha NT* indica que *NT* deve ser analisado sem que se utilize nenhum material lingüístico colhido anteriormente; também não é permitido que o material identificado entre seus subconstituintes seja usado no processamento do resto da sentença. Cláusulas relativas e terminais, por exemplo, são precedidos por uma barreira de tipo *ilha*.

Um termo com o formato *ilha_corte NT* indica que *NT* deve ser analisado sem que o material armazenado na estrutura *corte* seja correferenciado por nenhum de seus subconstituintes. Ao contrário da barreira de tipo *ilha*, no entanto, o material contido na estrutura de dados *corte* antes do início da análise de *NT* deve ser obrigatoriamente referenciado por um termo que precede ou sucede *NT*. Todo o material armazenado na estrutura *corte* durante a análise de *NT* deve ser consumido por algum de seus subconstituintes. Pode-se utilizar este tipo de barreira para prefixar sintagmas nominais, como em

$s \rightarrow \text{ilha_corte, Sintagma_nominal, Sintagma_verbal.}$

Esta barreira não impede que o material interrogativo referencie o traço na posição de sujeito, como no caso da informação associada ao pronome “quem”, armazenado no marcador *is*, e do traço *t_i* no exemplo abaixo:

Quem_{is}[*t_i*deu um livro à Maria]_s

Como as informações ocupam estruturas distintas, também é possível que uma cláusula relativa coindexe seu traço em uma posição de sujeito, como em

O cão_{relativo}[que_{is}[*t_i*latiu]_s]_{relativo}morreu.

²⁷ Traduzido livremente de *slash*.

²⁸ *Ilha* é o resultado da composição de cada uma das barreiras existentes, e é equivalente a uma sequência contendo todas as barreiras, sem importar a ordem, uma vez que as ilhas são comutativas por definição.

Um termo no formato *ilha_relativa NT* é equivalente ao previamente definido. O conteúdo do material contido na estrutura *relativo* antes do início da análise de *NT* deve correferenciar um traço não dominado por *NT*. A cabeça de uma cláusula relativa (possivelmente um pronome relativo) deve correferenciar um termo dominado por *NT*.

Da mesma forma, em um termo no formato *ilha_interrogativa NT* o conteúdo da estrutura de dados *interrogativa* presente antes da análise de *NT* não pode ser referenciado por um termo dominado por *NT*. A cabeça de cláusula interrogativa (provavelmente também um pronome) deve correferenciar um termo dominado por *NT*.

A composição comutativa associativa à direita de duas barreiras, prefixando um não-terminal, vai restringir o uso das duas estruturas de armazenamento correspondentes. Sejam *B1* e *B2* as barreiras e *St1* e *St2* as estruturas de armazenamento correspondentes; sejam *Mv1* e *Mv2* os dois tipos de movimento cujas descrições requerem o uso destas estruturas e barreiras. O termo *B1 B2 NT* vai ser interpretado como se segue: o conteúdo das estruturas *St1* e *St2* antes da análise de *NT* devem correferenciar um traço não dominado por *NT*. Qualquer material armazenado durante a análise de *NT* deve ser correferenciado por um traço dominado por *NT*.

De acordo com as definições anteriores, uma BMG é uma tupla $\langle NT, T, I, P, St, B, M, \alpha, \beta \rangle$ onde *NT* é um conjunto de não terminais, *T* é um conjunto de terminais, *I* é o símbolo inicial da gramática, *P* é um conjunto de regras BMG como definidas, *St* o conjunto de estruturas de armazenamento de dados (*stores*), *B* o conjunto de barreiras, *M* o conjunto de movimentos tratados e α e β duas funções bijetivas tais que: $\alpha: St \rightarrow B$, $\beta: B \rightarrow M$, $\beta \bullet \alpha: St \rightarrow M$.

2.3.9 Bound-Movement Grammars — BMGs

A proliferação de estruturas permite que algumas referências cruzadas possam ser tratadas. Mais especificamente, permite que referências cruzadas sejam tratadas quando pertencerem a fenômenos —e movimentações— diferentes. Esta capacidade ampliada em relação às XGs permite que a sentença “o livro que a uma amiga o João deu desapareceu” possa ser interpretada corretamente. A figura 2.47 mostra uma possível árvore de derivação.

Não existe um estudo formal sobre a capacidade forte do formalismo e não é possível aplicar-se diretamente os estudos sobre as XGs. Esta impossibilidade não deriva da substituição do operador ‘...’ pelas estruturas de armazenamento e barreiras identificadas, mas da impossibilidade do lado esquerdo de uma regra BMG conter seqüências de símbolos como definido para as XGs. Caso as modificações se restringissem ao operador ‘...’, poderíamos considerar as XGs como um caso especial das BMGs onde apenas um fenômeno lingüístico é tratado e afirmar que elas poderiam reconhecer todos os conjuntos recursi-

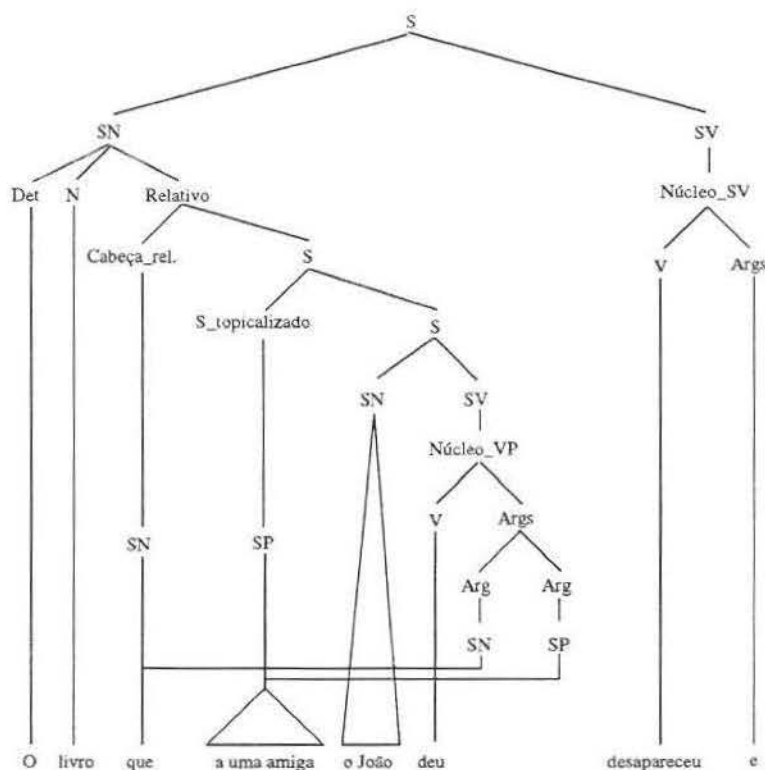


Figura 2.47: Possível árvore de derivação BMG para a sentença “o livro que a uma amiga João deu desapareceu”.

vamente enumeráveis [40].

Com relação às BMGs, sabemos que as referências anotadas são indiretas. Isto ocorre porque seu armazenamento em estruturas é mapeado através de funções que devem ser consultadas pelo analisador sintático, ao contrário do que ocorre com as TAGs, que mantém apontadores para os traços correferenciados e coindexados. Não se sabe até onde esta característica afeta a clareza estrutural das soluções, mas sabe-se que as regras contidas na gramática devem especificar diretamente como as referências são tratadas sem uma regra geral que coordene um funcionamento padrão (*default*).

A proliferação de estruturas permite sobrepujar parte dos problemas das XGs com referências cruzadas e cobrir uma ampla gama de fenômenos linguísticos. Alguns tipos de referências possíveis nas TAGs, no entanto, não são nas BMGs, em especial quando se trata de coordenações. Isto ocorre porque nas coordenações é possível encontrar o mesmo fenômeno linguístico —ou seja, movimentação— em mais de uma posição da sentença e com referências cruzadas. A sentença “o pote e a cadeira, vermelho e amarela respectivamente, estavam à venda”, por exemplo, não é corretamente analisada utilizando-se BMGs. Isto porque o formalismo limita o número de estruturas de armazenamento

a uma por fenômeno, o que nos leva de volta aos problemas das XGs. Na sentença o analisador sintático erraria ao associar “pote” com “amarela” e “cadeira” com “vermelho”, ficando o usuário do formalismo impossibilitado de tratar os problemas trazidos pelo “respectivamente”.

É necessário que as BMGs sofram uma formalização antes que um estudo sobre as capacidades forte e fraca de geração possam ser analisadas e comparadas com outros formalismos. Isto não impede, no entanto, que apontemos indícios de uma capacidade gerativa fraca e complexidade computacional equivalentes à das XGs, obtendo descrições melhores das derivações e simplificando o processo de escrita das gramáticas. Também não nos impede de apontar a persistência de alguns problemas com as referências cruzadas.

Capítulo 3

Mecanismos de implementação

“An algorithm must be seen to be believed.”

Donald Knuth

“Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary.

No such faith comforts the software engineer.”

Fred Brooks

No capítulo anterior abordamos três dos formalismos lingüísticos já utilizados com sucesso no processamento automático da língua portuguesa. Também procuramos levantar algumas características computacionais que afetam o desempenho dos sistemas que os implementam. Antes de querer exaurir o assunto, buscamos enumerar um conjunto de características que auxiliem na determinação da abordagem mais adequada de implementação.

Neste capítulo trataremos das alternativas de implementação sob o ponto de vista dos paradigmas de programação e desenvolvimento. Abordaremos breve e informalmente a metodologia tradicional de desenvolvimento procedural imperativo, centrada nos processos de resolução, o paradigma de orientação a objetos, centrado nas estruturas de dados e suas características, os arcabouços, que tentam fornecer modelos gerais de organização e, finalmente, o paradigma multi-agentes. O texto busca mostrar que, quando aplicadas ao PLN, estas abordagens possuem um caráter evolutivo.

3.1 Sistemas tradicionais

Os sistemas computacionais tradicionais de PLN, em geral, seguem a teoria lingüística que divide o processamento da língua em níveis de tratamento isolados. Esta divisão vai se refletir na organização dos sistemas computacionais implementados [26] [22], que adotam um fluxo de informação linear ou cíclico, sendo a saída de uma etapa de processamento a entrada da etapa subsequente.

Esta abordagem serial dificulta a utilização das informações de um nível lingüístico nos demais, podendo exigir múltiplas análises sobre um mesmo conjunto de dados. Em outras palavras, a informação colhida em uma etapa do processamento deve ser armazenada em estruturas de dados auxiliares que serão utilizadas nas próximas etapas. No caso de uma abordagem cíclica, a própria etapa que coleta a informação pode ter que esperar um novo ciclo de atividade para se utilizar dos dados coletados. Uma primeira execução coletaria os dados e uma segunda execução, em um novo ciclo, os utilizaria.

É bastante comum no português que as informações de um nível lingüístico sejam necessárias para resolver ambigüidades ou fenômenos específicos de outros níveis. Esta característica pode ser verificada, por exemplo, na resolução de anáforas [66], elipses [60], análise semântica [4], ou mesmo para a resolução da ambigüidade categorial [22].

A abordagem linear também acarreta efeitos negativos no desempenho computacional à medida em que as informações lingüísticas coletadas são transportadas para fora de seu contexto. Para manter o significado da informação intacto, o projetista fica obrigado a definir previamente para quais etapas e em que momento os dados coletados devem estar disponíveis. Quando se determina que uma etapa necessita de uma determinada informação, os dados devem se propagar por todos os níveis de processamento entre esta etapa e a fonte da informação. Cada uma das fases de análise depende diretamente da organização da anterior e manipula todo o conjunto de informações que circula pelo sistema, quer faça uso dela, quer seja apenas o conduíte para a próxima etapa.

A filosofia que rege este tipo de organização supõe que os processos são mais importantes e duradouros do que os dados que são coletados. Esta escola de pensamento foi a dominante na década de 1970 e nas anteriores, não apenas para o processamento das línguas naturais, mas no desenvolvimento de sistemas em geral.

Esta abordagem torna o desenvolvimento de programas de computador lingüísticos um processo caro e propenso a erros [94]. A propagação das informações dificulta a reutilização de código à medida em que as modificações em um determinado módulo do sistema gera efeitos nem sempre previsíveis. A ampliação de um sistema desta natureza também é dificultada, assim como sua manutenção.

Estas críticas não são uma exclusividade dos autores relacionados à inteligência artificial, mas bastante freqüentes na literatura de engenharia de *software* e de orientação a

objetos.

3.2 Sistemas orientados a objetos

A alternativa mais comum aos sistemas tradicionais são os sistemas orientados a objetos. Neste paradigma a atenção se volta aos dados envolvidos no problema, tornando os processos coadjuvantes na análise da informação. Neste caso se assume que os dados são mais duradouros que os processos.

Este efeito é obtido pela representação abstrata de objetos reais e pelo agrupamento de processos que manipulam estes objetos em uma única entidade. Esta entidade, definida como um conjunto de dados e processos, é genericamente chamada de objeto e reúne uma série de características que a torna adequada para a reutilização de código.

Um objeto encapsula as informações que contém; seus dados só podem ser acessados através dos processos que contém. Estes processos agregados também são chamados de métodos do objeto. O acesso aos métodos é feito através de um mecanismo elementar de controle de acesso.

Quando proibimos a manipulação direta da informação contida em um objeto, ganhamos a liberdade de modificar a estrutura de dados que a representa sem comprometer nenhuma outra parte do sistema que não sejam os métodos encarregados de manipular esta informação que, por sua vez, são codificados dentro do mesmo objeto (próximos dos dados).

A relação entre os objetos pode ser representada em uma estrutura "é-um", ou seja, é possível criar uma hierarquia de objetos que compartilham características comuns, através de um processo denominado herança. Quando queremos que um determinado objeto compartilhe características e métodos de outro tipo (ou classe) de objetos, podemos estipular que o objeto herde os dados e os métodos que desejemos. Também podemos fazer com que um objeto herde características de várias classes distintas de objetos, o que é denominado herança múltipla. Como pode-se notar, quando nos referimos a somente um objeto, com dados específicos, nos referimos a ele simplesmente como objeto; quando queremos nos referir a todas as entidades similares, que compartilham a estrutura de dados e os métodos, usamos o termo classe.

O mecanismo de controle de acesso às partes de uma classe também afeta a herança. Podemos definir três tipos de dados e métodos em relação à proteção:

- *público*: este dado ou método pode ser acessado diretamente por qualquer parte do programa. Quando uma classe herda as características de outra, todos os dados e métodos definidos como públicos são repassados e continuam públicos;

- *protegidos*: este dado ou método pode ser acessado apenas pelos demais métodos do objeto. Quando ocorre uma herança, este dado ou método é repassado como privado;
- *privado*: este dado ou método só pode ser acessado pelos métodos do próprio objeto e fica oculto no caso de uma herança.

Quando escondemos a informação e a propagamos em conjunto com os métodos que a manipulam, diminuimos a complexidade do sistema porque não tratamos mais a informação fora de seu contexto e não dependemos mais da estrutura de dados que a representa. Temos apenas que manter a estrutura de solicitação, ou seja, a rotina de chamada dos métodos de cada objeto. A chamada de um método é similar a uma função: no lugar dos argumentos da função temos os parâmetros do método mais um argumento implícito que é o próprio objeto dono do método que está sendo invocado.

Cada um dos métodos de um objeto possui um nome que o identifica. Este nome, no entanto, não é um identificador unívoco. É possível definir métodos diferentes com o mesmo nome desde que possuam parâmetros de tipos diferentes ou um número diferente de parâmetros. Esse processo, chamado sobrecarga, permite que utilizemos uma política de nomes homogênea para tratar objetos diferentes. Podemos, por exemplo, tratar a transferência de qualquer tipo de dados com um método “*transfere(de, para, o_que)*” ao invés de proliferarmos nomes especiais como “*transfere_arquivo_binário(de, para, nome_arq)*” ou “*transfere_mensagem_texto(origem, destino, mensagem)*”. Apesar de não liberar o programador de escrever um método para cada tipo de transferência, esta técnica torna o código mais legível, fácil de entender e manter.

Alguns objetos, como vimos anteriormente, possuem características compartilhadas, herdadas da hierarquia de classes. Quando essas características comuns são suficientes para determinar como uma tarefa deve ser realizada é desnecessário construir métodos novos. A propagação dos dados e métodos de maneira hierarquizada e acumulativa permite ao sistema decidir, em alguns casos em tempo de execução, se uma classe de objetos pode ser tratada como outra. Esta capacidade dos objetos se comportarem como se fossem de outra classe, em geral superior na hierarquia, é conhecida como polimorfismo. Um objeto da classe “*círculo*” pode ser tratado como um objeto da classe “*figura_geométrica*” desde que seja derivado dele.

É comum que alteremos um método da classe superior na hierarquia (ou mãe) nas classes derivadas (ou filhas). Algumas vezes é interessante que uma classe mãe funcione como um molde genérico do qual os filhos herdam propriedades e métodos, definindo desta forma uma interface comum com o sistema. Os métodos da classe mãe podem até mesmo ser vazios, definindo apenas os nomes e parâmetros de cada método. Cada classe filha deve implementar estes métodos vazios (virtuais) de acordo com suas características

próprias.

Quando utilizamos o polimorfismo para economizar no desenvolvimento de métodos, o sistema precisa saber qual o método a ser invocado, se o da classe mãe ou o da classe filha. Voltando ao nosso exemplo de figuras geométricas, podemos definir um método chamado "desenha()", sem parâmetros explícitos, vazio para a classe mãe e um método com o mesmo nome e parâmetros, mas com o processo de desenho, para cada classe derivada. A decisão de qual método invocar pode ser tomada em tempo de compilação com o auxílio de modificadores de tipo¹, ou em tempo de execução, em um processo denominado amarração tardia².

Estas características são interessantes à medida em que permitem tratar os diferentes tipos de informação envolvidos no PLN de forma homogênea. Outro aspecto positivo é a possibilidade de se reaproveitar código.

Podemos considerar os dados morfológicos, semânticos ou sintáticos como simplesmente dados, sem entrar no mérito de sua natureza, quando queremos passá-los adiante na cadeia de processamento. Isto não impede que os tratemos quando for interessante, desde que respeitemos o encapsulamento dos dados, o que nos obriga a utilizar a interface de comunicação prevista.

É exatamente este encapsulamento que nos permite modificar as representações internas sem propagar nenhum tipo de efeito colateral para o restante do sistema. Basta manter a consistência da interface dos métodos que, ao ser atingida, permite que um determinado objeto possa ser reaproveitado em outro sistema. Nossas classes de figuras geométricas podem ser utilizadas em diversos programas sem grandes alterações, assim como estruturas de dados abstratas como pilhas, filas, grafos ou qualquer outra classe coerente.

Em geral as maiores dificuldades relacionadas com o uso de objetos se referem à definição e implementação da hierarquia de classes e à definição de herança. Em relação aos sistemas de PLN existem mais fatores a considerar.

Quando definimos a estrutura de classes de um sistema, ou uma biblioteca de tipos, tentamos montar um conjunto flexível de objetos, maximizar o reaproveitamento de código para minimizar a repetição de trabalho e evitar a redundância de competências. Isto quer dizer que em geral os objetos pertencem a famílias de classes que por sua vez são responsáveis por um domínio específico. Não é difícil perceber que, em uma biblioteca de uso geral, classes genéricas como "desenho", "armazenamento" ou "grafos" são entidades abstratas que servem como o molde inicial de todo um conjunto de objetos utilizados na programação.

No caso da classe "desenho", podemos imaginar classes derivadas para polígonos re-

¹Do Inglês *cast*.

²Do inglês *late binding*.

gulares, círculos ou *canvas*³. No caso de uma hierarquia de armazenamento podemos prever o surgimento de classes para arquivos-texto, dicionários, arquivos binários; para os grafos podemos ter cíclicos ou acíclicos, dirigidos ou não. Uma organização inteligente pode minimizar a quantidade de classes que não derivam de nenhuma outra, raízes das hierarquias, quando for interessante que mais classes possam ser manipuladas homogeneamente. O caso extremo desta abordagem é fazer com que todas as classes derivem da classe 'objeto'.

Aumentar a profundidade da hierarquia, no entanto, também tem seu custo. Os objetos instanciados a partir de uma classe herdam os métodos e dados de toda a hierarquia (classes-mãe) aumentando o tamanho do programa e, dependendo da linguagem de programação, o custo de determinar qual função está amarrada a um objeto em momentos específicos da execução do programa⁴.

Um problema específico do domínio é que, quando tratamos de línguas naturais, o modelo global de comportamento não é conhecido. Não se sabe exatamente quais componentes participam de quais atividades e, muitas vezes, não se sabe como um determinado componente funciona. Isto não quer dizer que por não poder mimetizar a realidade não devamos usar a programação orientada a objetos. Podemos abrir mão das relações mais imediatas e abstrair uma estrutura de classes que, dentro de uma teoria lingüística ou cognitiva específica, tenha competência para resolver um conjunto determinado de fenômenos lingüísticos.

Restringir o paradigma às limitações dos formalismos retira algo do poder de reaproveitamento dos objetos. Dessa forma eles não podem mais ser retirados de um sistema e colocados em outro sem uma adaptação mais ampla. Essa adaptação é facilitada pelo encapsulamento de dados e métodos, bastando manter a consistência dos métodos.

3.3 Arcabouços

Em alguns domínios especiais, como no PLN, a hierarquia de classes é menos intuitiva do que em outros, como no exemplo de sistema de desenho. Isto advém da dificuldade em mimetizar as estruturas que resolvem o problema no mundo real e em relacionar estas estruturas hierarquicamente. Com uma hierarquia fragmentada é mais difícil que consigamos reaproveitar objetos de um sistema em outro. Também é mais comum que tenhamos dois ou mais objetos compartilhando a mesma competência.

Sobreposição de competências em um sistema pode levar ao inchamento do projeto final, à ineficiência e à ineficácia. O desempenho é afetado pelo tamanho do código e pelos

³Área da tela onde o programa pode desenhar.

⁴Algumas linguagens, como o *C++* tratam os objetos como uma categoria especial de tipo, o que não elimina o problema, apenas o transfere parcialmente para a etapa de compilação.

testes que devem ser feitos para determinar qual objeto será utilizado para qual propósito; neste caso não poderemos contar com a sobrecarga de funções porque, quando falamos de objetos reaproveitados, a política de nomes não é necessariamente homogênea. A capacidade final de resolução também pode ser afetada porque ao utilizarmos ora métodos de um objeto, ora métodos de outro para uma mesma atividade e uma mesma competência, podemos ter semânticas diferentes, mesmo que de maneira sutil.

Mesmo em sistemas bastante decantados, como no caso de um sistema de janelas, existem problemas. Por exemplo, de quem é a competência de impedir que o conteúdo de uma janela não vazze para as demais? A solução da Microsoft (MFC — Microsoft Foundation Classes) é considerar que o *canvas* pode contar com a proteção do sistema operacional. A solução da Borland (OWL—Object Windows Library) é considerar que o objeto, no caso o próprio *canvas*, deve garantir que o conteúdo não vazze. Já a solução da Sun Microsystems (xlib — Solaris OpenWindows 3.5.1) é simplesmente deixar que o conteúdo vazze, tornando o programador responsável. Estas discrepâncias ocorrem mesmo quando o objeto em questão (no caso o *canvas*) possui basicamente os mesmos métodos em todos os casos, inclusive com nomes similares.

Os arcabouços⁵ tentam disciplinar as competências das classes de um sistema especificando quais são os domínios e como devem ser preenchidos. Desta forma, ao desenvolvermos uma biblioteca de classes para um sistema que obedeça à configuração do arcabouço, poderemos reaproveitar os objetos desta biblioteca em quaisquer outros sistemas que também se baseiem no mesmo arcabouço, sem os problemas de sobreposição ou de semântica e conservando as vantagens da programação orientada a objetos.

Ao delimitar competências, os arcabouços definem implicitamente um padrão de resolução para o problema que mapeiam. Os objetos que compõem um sistema baseado neste modelo ou arquitetura têm uma estrutura pré-moldada de troca e compartilhamento de informação.

No caso do PLN, à medida em que novos fenômenos lingüísticos são estudados, novas teorias são formuladas e novos tipo de integração entre os nichos de competência são revelados. Quando tentamos resolver anáforas possessivas no português [66] podemos ter a colaboração de informações morfológicas. Por exemplo, na sentença “João jogou seu lápis e sua borracha no chão, mas Maria conservou a sua na mão” sabemos que Maria segurou a borracha e não o lápis devido ao gênero. O lápis de Maria não foi citado (nem sabemos se ele existe) mas a informação de sua existência poderia ser assumida como informação presumível pela parte do sistema que trata da análise pragmática ou semântica [4].

Em outros casos a informação morfológica não é suficiente e temos que apelar para a análise semântica e do discurso. Em uma sentença imediatamente posterior à do exem-

⁵Do inglês *frameworks*.

plo anterior poderíamos dizer “E o seu lápis ficou limpo” e poderíamos supor que Maria possuía um lápis e que, por não tê-lo atirado ao chão, permaneceu limpo. Outras interpretações também são possíveis; queremos apenas demonstrar a integração dos diversos níveis lingüísticos para mostrar que a definição do arcabouço deve levar em consideração muitos tipos de comunicação.

A definição deste tipo de modelo sofre então de um problema de dosagem. Se o arcabouço for demasiadamente genérico, pode não conseguir impedir a sobreposição da competência dos objetos; se for por demais restritivo acaba por definir qual hierarquia de classes deve ser construída.

3.4 Sistemas Multi-Agentes

Ao levantarmos as principais características de um conjunto de formalismos ou paradigmas de programação, buscamos enumerar as características desejáveis para a resolução de um problema. Quando enumeramos seus problemas estamos tentando definir quando a abordagem é adequada e quais as dificuldades que teremos ao utilizar este conjunto.

A orientação a objetos nos permite minimizar os efeitos colaterais de uma alteração pelo encapsulamento e facilita o reaproveitamento de código devido ao polimorfismo e à hierarquia de classes (herança). Os arcabouços nos trazem organização no desenvolvimento do sistema e, por conseqüência, organizam o reaproveitamento.

Se maximizarmos os caminhos de comunicação de um arcabouço teremos ligações de todos os componentes com todos os demais componentes. Isto nos permitiria usar a mesma arquitetura mesmo que descobríssemos novas dependências lingüísticas em um problema. Esta proliferação de vias de comunicação, no entanto, poderia propiciar o surgimento de interfaces desnecessárias em vários componentes do sistema, que teriam que estar programados para manipular qualquer informação, fosse para descartá-la, reenviá-la ou digeri-la. A solução ideal neste caso em que não conhecemos o modelo de resolução *a priori* seria poder definir os caminhos de comunicação dinamicamente ou tornar sua modificação fácil e com poucos (ou nenhum) efeitos colaterais.

Em relação aos objetos, a situação ideal seria definir uma maneira de transplantá-los de um sistema a outro mesmo quando os nichos de competência se sobreponham ou não resolvam todo o problema, deixando tarefas parcialmente completas. Para que isto pudesse acontecer os nomes dos métodos deveriam mudar dinamicamente de acordo com o contexto onde o objeto está inserido e os parâmetros deveriam ser interpretados da mesma forma. Esta sensibilidade ao contexto do objeto não apenas se limitaria à política de nomes, mas também consideraria a semântica de cada sistema, fazendo com que seus métodos se comportassem de acordo com o “meio-ambiente”. Esta é uma tarefa especialmente complexa porque exige uma redefinição dinâmica do que se espera de cada

objeto, ou seja, exige que se agreguem *objetivos* aos objetos além dos métodos de resolução dos problemas.

Para obter nosso objeto ideal devemos considerar que as informações disponíveis em cada sistema são diferentes. Em alguns sistemas a alteração dos nichos de competência pode fazer com que alguns dados não estejam disponíveis no momento da chamada do método por não serem mais necessários, ou por ainda não terem sido definidos. Alterando a natureza da informação e, conseqüentemente, o número e o tipo dos parâmetros que ativam um método do objeto, não poderíamos mais contar com o mecanismo de sobrecarga, que define qual método chamar de acordo com o número e tipo dos parâmetros de chamada. A conseqüência mais imediata é que os objetos devem poder “amostrar” o meio para compreender o que dele se espera, ou seja, devem ter a capacidade de “perceber”.

Uma alternativa prática seria maximizar, assim como fizemos com as vias de comunicação no caso do arcabouço ideal, a operação de sobrecarga. Definiríamos apenas um método público para o objeto e este método receberia uma pilha de parâmetros como argumento. Sua implementação se encarregaria de decidir quais métodos internos seriam chamados para resolver o problema. Desta forma seria garantida a homogeneidade da interface de comunicação não importando qual o sistema onde o objeto está sendo usado. Uma ou mais chamadas seriam utilizadas para configurar o comportamento deste objeto em relação ao meio, ou seja, seriam utilizadas para perceber o que cerca o objeto. Seria desta forma que o objeto obteria as informações sobre os demais objetos do sistema e a definição do que se espera dele.

A decisão de qual (ou quais) métodos chamar a partir de um conjunto de parâmetros é deixada a cargo do programador/projetista nos sistemas atuais. Nosso objeto ideal vai precisar de mais informações para tomar esta decisão dinamicamente. Isto quer dizer que um ou mais dos parâmetros usados na chamada de nossa função única devem conter elementos que auxiliem nesta decisão. Em outras palavras, cada chamada deve possuir um objetivo, representado, provavelmente, por uma diretiva de um protocolo de negociação.

Se agruparmos o que desejamos do nosso arcabouço ideal com o que desejamos de nosso objeto ideal e adicionarmos uma definição dinâmica de nichos de competência, teremos um sistema multi-agentes. Um agente encapsula não somente seus dados e métodos, mas também sua política de nomes para os métodos e mesmo o paradigma em que foi implementado. Ele possui apenas um ponto de entrada, que é a função que recebe as mensagens do sistema e a maneira como esta mensagem é interpretada internamente também é privada. Ele é capaz de perceber o ambiente em que está inserido, armazenar informações sobre os demais agentes, possuir e incorporar objetivos e modificar seu comportamento para realizar o que dele se espera⁶.

⁶De uma forma geral espera-se que um agente consiga realizar seus objetivos próprios sozinho ou cooperativamente e que possa colaborar na resolução de objetivos alheios ou globais.

A mudança de paradigma ocorre porque, ao definirmos nosso sistema ideal, movemos novamente o foco do processamento. Em um primeiro momento este foco deixou os processos e passou aos dados na programação orientada a objetos. Agora deixa os dados e passa para a cooperação entre módulos cooperativos que agregam dados e processos.

Esta alteração de foco traz novos problemas, como a determinação do protocolo de cooperação, de uma linguagem comum entre os agentes, de um protocolo de decomposição de tarefas e de síntese de resposta. Em uma visão mais extrema é possível considerar que esta abordagem pode trazer problemas sociais, na medida em que nossos novos objetos (agentes) podem possuir objetivos conflitantes ou comportamentos não cooperativos.

3.4.1 Definição de agente

Existem diversas definições para o termo “agente” que variam de acordo com a filosofia dos sistemas e de quem o define. Algumas destas definições foram sumarizadas [22] e serão apresentadas a seguir.

Um dos conceitos mais intuitivos de agente é uma metáfora que o relaciona com a figura de um representante que atua por outra pessoa [62]. A idéia, neste caso, é relacionar à palavra agente um mecanismo computacional que execute alguma tarefa sem a necessidade de se explicitar seu funcionamento interno.

Outra definição considera o agente uma entidade real ou virtual inserida em um ambiente conhecido [81]. Esta entidade teria a capacidade de executar ações neste ambiente, percebê-lo e representá-lo parcialmente. Um agente dentro deste ambiente deve poder comunicar-se com outros agentes e possuir um comportamento autônomo.

Um agente também foi definido como uma entidade inteligente que possui objetivos e busca atingi-los de forma racional e intencional conforme o estado atual de seu conhecimento [11]. Seu comportamento autônomo é inerente à sua própria existência e independe da existência de outros agentes. Ele é capaz de criar objetivos próprios, baseado em seu interesse motivacional, e agir de forma flexível e adaptativa para alcançá-los eficientemente.

A autonomia, no entanto, não é uma característica unânime. Ela pode ser considerada não como um elemento integrado à natureza do agente, mas um aspecto relacionado [59]. Nesta definição um agente é qualquer entidade a que se pode atribuir um estado mental. Numa comparação com objetos, podemos dizer que um agente é a instanciação de uma classe associada a um conjunto de objetivos. Já um agente autônomo seria a instanciação de um agente com um conjunto de motivações associadas.

Segundo [22] podemos definir um agente como uma unidade computacional independente, com algum grau de inteligência, representado pelos estados mentais internos, que possui a capacidade de executar ações conjuntas com outras unidades, numa intenção

mútua de atingir um objetivo global em determinado momento.

Esta última definição é a mais apropriada a nossos objetivos, mas nos leva a definir o que é inteligência e o que são intenções para uma entidade artificial. Considerando este problema e a nossa definição de objeto ideal, adotaremos uma definição derivada. Aqui, para unidades artificiais, inteligência será entendida como a capacidade de possuir objetivos, discernir quais os recursos necessários para atingir estes objetivos, interpretar um protocolo de comunicação a ponto de adquirir ou propor novos objetivos, conseguir tratar partes de um problema composto e de solicitar auxílio, além de saber comunicar suas capacidades (se apresentar) a um grupo de entidades similares. Intenção é a execução de um conjunto de mecanismos que leve a resolução do problema para o qual o agente foi programado ou o cumprimento dos objetivos adquiridos pela solicitação dos demais agentes.

Uma consequência desta definição mais rígida é a cooperação obrigatória. Não existe, dentro desta proposta, espaço para agentes que recusam a cooperação por outros motivos que não o esgotamento de recursos computacionais.

3.4.2 Tipos de agentes

Os agentes podem ser classificados quanto a maneira com que realizam suas tarefas de duas maneiras [22]:

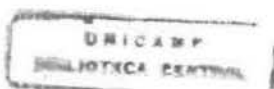
- *Agentes reativos*: sem recordações de seus atos passados ou previsão de ações futuras, o seu comportamento é baseado no processo estímulo-resposta, interagindo através da percepção de mudanças no ambiente. Não existe uma representação do ambiente ou dos agentes, nem uma comunicação de alto nível. Em uma sociedade de agentes reativos, o comportamento “inteligente” global emerge do comportamento local de uma coleção de entidades simples; por isso, uma sociedade reativa é composta por um grande número de membros. Seu objetivo fundamental é uma espécie de auto-satisfação, já que predominam os objetivos locais [35] [81].
- *Agentes cognitivos*: um agente cognitivo ou deliberativo possui uma representação explícita do ambiente e dos outros agentes podendo raciocinar sobre ações passadas e planejar ações futuras. Sua ação inteligente ocorre quando o agente possui um objetivo e sabe quais ações são necessárias para atingí-lo. De posse disto, ele “delibera” sobre aquela ação e a executa. A sociedade é composta por um pequeno número de membros [35] [81].

Estas categorias de agentes não são excludentes, mas sim graduais. Isto quer dizer que um agente pode ser definido de maneira mais reativa ou mais cognitiva de acordo com as

UNICAMP

BIBLIOTECA CENTRAL

SEÇÃO CIRCULANTE



necessidades de projeto. Um agente no qual os fatores com maior peso no processo de decisão são os dados colhidos no ambiente⁷ é mais reativo do que cognitivo. Um agente que seja mais dirigido pelos estados internos e pela memória de suas ações anteriores do que propriamente pelo ambiente⁸ é mais cognitivo do que reativo.

3.4.3 Definindo um sistema multi-agentes

Como dissemos anteriormente, ao adotarmos um sistema multi-agentes (SMA) estamos retirando o foco do projeto dos processos e dos dados e colocando-o na interação entre módulos cooperativos (agentes). Esta definição, no entanto, não é nem completa nem unânime.

Outro aspecto que deve ser considerado ao definirmos um sistema multi-agentes é a decomposição do problema que será tratado pelo sistema (ou sociedade de agentes).

Metodologia de decomposição

Segundo [85] existem quatro etapas que devem ser obedecidas no processo de resolução de um problema: decomposição do problema, alocação dos subproblemas, solução dos subproblemas e integração das soluções parciais ou síntese do resultado.

Alguns fatores determinam a qualidade da solução destas etapas:

- Cobertura: todos os aspectos do problema devem ser cobertos;
- Conectividade: a interação entre os agentes deve permitir que as tarefas definidas sejam realizadas e integradas na solução global, o que envolve tanto a organização dos próprios agentes como da sociedade. Deve existir uma organização física de comunicação entre os agentes e, a qualquer momento, uma organização lógica deve poder ser aplicada sobre esta estrutura. Isto não quer dizer que a estrutura de comunicação deva ser estática;
- Capacidade: a cobertura e a conectividade devem ser obtidas dentro das possibilidades de comunicação, computação e de recursos da rede de agentes.

A maneira como o problema foi decomposto influencia diretamente que tipo de informação cada agente contém e como estes agentes se comunicarão [86].

A cobertura do problema deve incluir alguma redundância para se obter alguma tolerância à falhas. Estas falhas podem estar relacionadas à incompletude ou incerteza da

⁷ Podem ser sensores físicos, interação com o usuário ou mesmo mensagens dos demais agentes.

⁸ Mesmo que as ações sejam disparadas por alguma mudança ambiental, interação com o usuário ou mensagem de outros agentes.

informação relacionada ao problema, o que depende do domínio, ou ao comportamento de algum agente, que pode perder a comunicação ou falhar (no sentido computacional).

Quanto à conectividade devemos considerar a topologia de interconexão entre os agentes, por onde circularão dados e sinais de controle e a estrutura de comunicação em que se baseia a comunicação.

3.4.4 Tipos de decomposição

É possível distinguir dois tipos de decomposição de um problema do ponto de vista de um agente [8]. No primeiro tipo, denominado *extrínseco*, a decomposição é externa ao agente; os agentes são capazes de resolver o problema individualmente e o uso de vários deles em paralelo busca a aceleração do processo. No segundo tipo, denominado *intrínseco*, os agentes são especialistas cujas capacidades são definidas pelo projetista ou por mecanismos de aprendizado.

Assim como no caso da divisão entre agentes reativos e cognitivos, esta divisão não é excludente [8].

Decomposição extrínseca

Neste tipo de decomposição o trabalho a ser realizado é dividido entre os agentes apenas de acordo com a capacidade computacional de resolução do problema, à medida em que todos os agentes são idênticos ou pelo menos têm o mesmo papel no processo de resolução do problema.

A divisão de toda a tarefa de resolução é feita de acordo com os critérios espacial e temporal.

- *decomposição espacial*: a decomposição espacial é a técnica mais difundida de decomposição de problemas [85] [33]. Aqui dividimos o problema em áreas geográficas e encarregamos os agentes de realizar tarefas idênticas em cada uma destas áreas, ou seja, a divisão é baseada numa divisão quantitativa dos dados de entrada e/ou saída do processo de análise [8]. O grau de interação entre os agentes depende do domínio do problema [85] e da granularidade dos agentes [41].
- *decomposição temporal*: aqui a decomposição é baseada em intervalos de tempo, ou seja, de acordo com a quantidade de tempo que os agentes levarão para solucionar o problema.

Neste tipo de decomposição os agentes são praticamente independentes e compartilham os mesmos objetivos, sem que exista uma hierarquia entre eles [8].

Decomposição intrínseca

Neste caso a decomposição se baseia na especialização dos agentes. Isto significa que cada agente tem um papel diferente no processo de resolução.

Um tipo de decomposição que se baseia nesta técnica é funcional ou baseada em tarefas. Neste caso decomparamos o processo de resolução do problema e não os dados analisados. O objetivo da sociedade é decomposto em vários processos e ações que devem ser realizados para se atingir os sub-objetivos e, desta forma, as ações determinam a decomposição do problema [85].

A decomposição funcional determina uma relação de precedência entre as sub-tarefas, ou seja, uma sub-tarefa depende da realização parcial ou total de outras sub-tarefas. Uma solução efetiva neste tipo de organização é quebrar o problema de forma que os recursos computacionais sejam aproveitados de maneira ótima [85]. Neste caso os agentes podem não executar tarefas idênticas e desta forma possuir capacidades e objetivos diferentes uns dos outros.

Apesar do grau de paralelismo oferecido por esta solução ser menor que a apresentada pela divisão extrínseca, com agentes uniformes, este modelo tem vantagens em relação à complexidade dos agentes. Numa variante denominada decomposição baseada em domínio [8], destina-se a cada agente apenas uma parte do conhecimento necessário para resolver todo o problema, ou seja, determina-se que este agente seja competente em um domínio limitado. A limitação do domínio diminui o espaço de busca de cada agente individualmente, sem afetar a cobertura do sistema como um todo. Esta característica simplifica cada agente e diminui o tempo de resolução de cada sub-tarefa [8], mas não necessariamente o tempo gasto pelo sistema como um todo, à medida que a síntese da resposta e o custo da comunicação entre agentes pode tomar o tempo ganho nas sub-tarefas.

3.4.5 Uma possível formalização

O processo de decomposição dos problemas em geral adota soluções *ad hoc* [8] [85] [41]. Uma alternativa mais esquemática, apresentada por Uma, Alvares e Demazeau [85], pode ser definida como se segue.

Seja P um problema ou descrição de tarefa denotada como

$$P(S, G, F) \rightarrow (A, E, I, O),$$

onde $S = (S_{\text{dados}}, S_{\text{restrições}})$ é o ponto inicial ou os dados iniciais e suas restrições, se houver. G é um conjunto de objetivos que devem ser atingidos pelo sistema multi-agentes (SMA), usando F . Aqui se assume que G pode ser resolvido solucionando-se G_{sub} , que é o conjunto de sub-tarefas de G .

A capacidade do SMA pode ser descrita pelos parâmetros A, E, I e O .

- um agente A_i pode ser descrito por

$$A_i = \{A_{cap}^i, A_{conf}^i, A_{disp}^i, A_{carga}^i\}, \text{ onde}$$

A_{cap}^i representa a capacidade de resolução do agente em relação aos problemas g . Neste caso g não é igual a G nem a G_{sub} , mas sim uma descrição mais detalhada de quais tarefas o agente pode realizar. Desta forma $A_{cap}^i = \{g_1, g_2, \dots, g_m\}$;

A_{conf}^i é a informação, em um tempo t , de confiabilidade do agente em relação à resolução de um problema g ;

A_{disp}^i é uma informação booleana determinando se o agente está ou não disponível no momento t ; e

A_{carga}^i é a medida de carga computacional e de comunicação do agente em um momento t .

Estes parâmetros variam no tempo, ou seja, se em um momento t qualquer tivermos $A_{disp}^i = 0$, isto não indica que o agente esteja indisponível indeterminadamente. Em um momento t_2 o agente pode estar novamente livre.

Outras informações críticas como, por exemplo, quais tarefas são prioritárias para um agente, ou se o agente é acionado por sensores, também devem ser consideradas no caso de existir alguma ambigüidade.

- E descreve o ambiente do agente. Esta é uma função parcial da função de entrada S_i^A e também a representação dos demais agentes pelo agente A_i .
- I descreve o conjunto de interações do agente nos termos dos protocolos de comunicação.
- O descreve a organização dos agentes. As ligações existentes entre os agentes A_i e A_k e as características desta ligação como, por exemplo, se por difusão⁹ ou diretamente, *duplex* ou *simplex*¹⁰, com ou sem confirmação, solicitado ou não, etc. Estas informações, combinadas com os protocolos de comunicação, nos levam à organização do agente. Também definem os poderes relativos dos agentes em relação a um determinado objetivo [85].

⁹Do inglês *broadcast*.

¹⁰Na comunicação *simplex* os agentes recebem e emitem informações alternadamente, em tempos determinados para emitir ou receber. Na comunicação *duplex* eles podem estar recebendo e enviando informação simultaneamente pelo mesmo canal de comunicação.

Alguns comentários sobre a técnica de formalização

Uma das motivações para o desenvolvimento da metodologia apresentada em [85] é a predominância de soluções *ad hoc* na decomposição de tarefas no desenvolvimento de sistemas multi-agentes.

A técnica apresentada tenta determinar não apenas os conjuntos de informação que devem ser definidos, mas basear sua construção na comunicação desejada entre os agentes que compõem o SMA. Ao determinar esta precedência, esta técnica traz problemas no reaproveitamento de sistemas existentes.

Quando desejamos a integração de um ou mais agentes pré-existentes em um novo SMA não podemos determinar a funcionalidade do novo sistema apenas considerando a comunicação desejada, mas também a comunicação possível. Assim como a utilização de objetos fora de seu sistema original pode trazer problemas semânticos (como vimos anteriormente), a utilização de agentes fora de sua sociedade original pode trazer primitivas de comunicação que não são passíveis de decomposição como, por exemplo, transações atômicas com bancos de dados. Alterar este tipo de transação para atender um nível de comunicação mais granular pode trazer conseqüências negativas para o sistema.

Ao considerarmos as técnicas de decomposição de problemas [22] [85] [41] podemos reparar que os exemplos são predominantemente determinados pela natureza dos agentes e não pelos mecanismos de comunicação e interação. Entre outras coisas, a granularidade do agente (sua competência) deve ser determinada não apenas pela comunicação desejada, mas também pela complexidade do problema tratado, pela facilidade de distribuição das tarefas definidas e pelas características do domínio para o qual o SMA foi projetado, além de considerar aspectos de reaproveitamento de módulos previamente construídos.

3.5 Dois estudos de caso

Apresentaremos agora dois SMA definidos no contexto do projeto NALAMAS [26] e bastante contrastantes no que se refere à natureza dos agentes, aos protocolos de comunicação e de interação, mas funcionais sobre um mesmo domínio, o processamento computacional do português.

3.5.1 Um Sistema Multi-Agentes para Processamento Distribuído de Linguagem Natural

Este trabalho[25] trata de um sistema multi-agentes encarregado de lidar com a sintaxe de sentenças do português de modo distribuído.

O conjunto G deste sistema, ou seja, seus objetivos, consiste na resolução de um conjunto de fenômenos da língua portuguesa:

- **Topicalização:** pode ser compreendida como a mudança de um termo para o início da oração. Ex.: Léo comeu o peixinho → O peixinho, Léo comeu.
- **Posposição do sujeito:** normalmente o sujeito da oração vem antes do núcleo do predicado, mas há situações em que ele pode vir após este núcleo. Ex.: Chegaram vários carregamentos de cachaça.
- **PP-Attachment:** ocorre quando existe mais de uma árvore sintática para uma determinada sentença, o que denota ambigüidade estrutural. Ex.: O menino viu a menina com o telescópio. Neste exemplo podemos entender que o menino usou um telescópio para ver a menina ou que a menina carregava um telescópio.
- **Construção relativa:** a construção relativa se caracteriza pela presença de um relativo (que, qual, quem, onde ou cujo) precedido às vezes por uma preposição. Também se caracteriza pela presença de uma estrutura oracional aparentemente incompleta logo após o relativo. Este relativo se articula com um elemento nominal¹¹ formando uma sequência que é um sintagma nominal. Ex.: O menino que ama a menina caiu.
- **Coordenação:** a coordenação se caracteriza pela presença de uma conjunção coordenada que liga dois constituintes de mesma classe. Estes dois constituintes formam um conjunto que é tratado como um constituinte da mesma classe. Ex.: Míriam e Tiago fazem Belas-Artes;
- **Redução:** uma oração coordenada sem sujeito. Ex. (redução simbolizada por []): César impressionava os homens e [] beliscava as mulheres.
- **Gapping:** tipo de elipse que ocorre em orações coordenadas onde o elemento verbal da segunda oração é omitido. Ex.: (*gap* representado por []): João comeu a maçã e Pedro [] a banana.

Metodologia de desenvolvimento

A partir de um conjunto inicial G de objetivos, o sistema foi baseado em um estudo comparativo entre as abordagens disponíveis de processamento lingüístico baseado em SMA.

A escolha de quais sistemas analisar partiu do tipo de decomposição escolhido para o problema de análise sintática de sentenças. As três formas de decomposição consideradas

¹¹Parte de um sintagma nominal.

foram a não decomposição, tratando sentenças em paralelo (técnica extrínseca — não há simplificação dos agentes), decomposição em sintagmas e decomposição por palavras, que foi a técnica escolhida para experimentação.

Ao assumir a decomposição do problema em termos da competência dos agentes, a sequência de definição do sistema passou do conjunto G para os subconjuntos A_{sub}^i , ou seja, a definição dos objetivos parciais dos agentes. Apesar desta escolha não necessariamente implicar na definição imediata das capacidades dos agentes (A_{cap}^i), isto foi feito em detrimento da definição da cooperação desejada, ou seja, dos protocolos de comunicação e interação.

As capacidades de cada agente baseiam-se na organização e arquitetura geral do *Word Expert Parser* (WEP) [82]. No WEP cada palavra é tomada como um agente que participa do controle global do processo de análise. Os agentes interagem até que cada um tenha determinado seu sentido na sentença ou esteja em uma situação em que não possa prosseguir sem informações adicionais. O conhecimento contido em cada agente do sistema envolve sinais de controle, mensagens para os demais agentes, mecanismo de discurso e de conhecimento de mundo, espalhados numa rede cujos arcos contém dados que permitem a determinação do sentido da palavra. No sistema apresentado, no entanto, os agentes agregam não o conhecimento de cada palavra, mas de categorias de palavras, como veremos mais adiante.

O processo de construção da solução adota o conceito das *Polaroid Words*, definidas para os sistemas *Paragram* e *Absity* [42]. Originalmente definidas para tratar a ambigüidade semântica, estas estruturas possuem *frames* e *slots* que são preenchidos à medida em que o processamento ocorre, gerando instâncias das *Polaroid Words*. As informações necessárias são coletadas durante a definição das dependências entre as palavras da sentença. A solução é construída em etapas e vai se revelando aos poucos, junto com o avanço do processamento, daí o nome *polaroid*.

Um SMA precisa definir também sua metodologia de comunicação. Aqui optou-se pelo reaproveitamento das primitivas definidas no *Parallel Expert Parser* (PEP) [29],[30]. O PEP é uma paralelização do WEP e prevê tanto a comunicação direta quanto a indireta entre os agentes.

Também foram definidos critérios de interação e protocolos de comunicação. Os princípios que regem a interação neste sistema refletem a granularidade (tratamento por categoria de palavras) e a decisão de reaproveitar as primitivas de comunicação do PEP. Basicamente existem dois fatores que regem o processamento:

- Conceito de forças de atração: as palavras podem ser classificadas em categorias e estas categorias atraem umas às outras. Mapeando as forças de atração pode-se realizar uma análise sintática.

- Definição de mecanismos de restrição para controlar o comportamento dos agentes, regido pelas forças de atração:
 - Subcategorização: Define o tipo de argumento que deve estar presente em posições específicas ao redor dos agentes;
 - Vizinhança: limita a influência das forças de atração de um agente a uma determinada distância. Sem a definição desta restrição, um verbo atrairia todos os substantivos;
 - Restrição semântica: descreve o que um agente espera dos complementos que são atraídos.

Processo de análise

Apesar dos agentes que representam os termos determinantes da sentença iniciarem o processo de análise, é a cooperação entre os agentes que define as conexões através dos protocolos de negociação de todos os agentes.

Estas conexões são estabelecidas porque os agentes são estimulados pelas forças de atração —essencialmente subcategorização— e dirigidos pelas restrições lingüísticas definidas. A análise termina quando todos os agentes fizeram suas conexões ou ficaram sem opções de negociação.

O processo é determinado pelo comportamento dos agentes, que é heterogêneo. Uma lista dos agentes definidos mais importantes para o processo e uma breve descrição de seus comportamentos é mostrada a seguir:

- *Verbos*: buscam seus argumentos e são responsáveis pelo início da negociação;
- *Substantivos*: são passivos e esperam a chegada de propostas de conexão;
- *Preposições*: buscam por um substantivo à sua direita e se ligam a ele. Alternativamente podem buscar no lado esquerdo por nomes ou verbos;
- *Pronomes relativos*: buscam um verbo à sua direita e se conectam de acordo com o *gap*. Este *gap* é definido pela presença ou não de um substantivo entre o agente e o verbo. Alternativamente podem buscar um substantivo à esquerda e se conectar.
- *Coordenações*: buscam dois agentes de mesma categoria, um de cada lado, e passam a agir de acordo com esta categoria;
- *Determinantes*: buscam um agente substantivo à sua direita;
- *Demais agentes*: buscam seus modificadores e aqueles a quem modificam.

O comportamento dos agentes não é limitado pela quantidade de conexões feitas, mas existe uma direção preferencial para estas conexões, ou seja, uma leitura preferencial da sentença de entrada.

A síntese da resposta

Os modelos de processamento possíveis para um SMA aplicado ao PLN ainda são uma questão aberta, o que acaba por impedir a determinação de quais os elementos mais importantes em uma resposta, conforme discutido anteriormente. O processo de síntese, no entanto, é necessário para que tenhamos respostas manipuláveis.

Aqui foram consideradas duas possibilidades de síntese para a resposta do sistema: a agregação e a conexão. No caso da agregação um agente é incorporado por outro, desativado, e tem suas conexões assimiladas e gerenciadas pelo agente incorporador. No caso da conexão os agentes se conectam uns aos outros e, caso a solução não seja satisfatória, simplesmente quebram as conexões; esta última foi a proposta adotada.

Este protocolo de interação permite que os agentes façam conexões erradas globalmente mesmo que estas estejam corretas localmente, demonstrando como a tática supera a estratégia. Quando isto ocorre os agentes que não estabeleceram suas conexões podem pedir aos demais que tentem uma nova configuração. Um exemplo de sentença onde isto pode ocorrer é “O menino que ama a menina caiu”¹².

Os agentes verbais determinam, além do início das negociações, o fim da análise: quando todos eles estão completamente conectados o processo acaba; se existir um agente com conexões incompletas, ele pode pedir auxílio para a sociedade (difusão) e aguardar as contribuições. Caso nenhuma contribuição surja a sentença é considerada agramatical e o processo pára.

Aspectos de implementação

Analisando-se o protótipo implementado podemos observar que o sistema simula a distribuição através de uma lista de agentes, que são ativados de acordo com uma sequência pré-estabelecida. Esta ordem de ativação pode ser quebrada por mensagens diretas, que determinam a mudança imediata de contexto para o agente chamado. Ao final do processamento a ordem de ativação é restabelecida; uma sequência de ativações diretas pode determinar uma busca em profundidade, enquanto a lista de ativação pura determinaria uma busca em largura.

¹²No caso de um sistema que estipula um agente para cada palavra, o agente encarregado da palavra “menina” pode ser enganado pela presença do verbo “caiu” à sua direita. Esta “armadilha” não pode ser detectada pelo analisador diretamente porque existe uma concordância verbal. Este fenômeno também é conhecido como “caminho do bosque”. Parece correto, parece fácil de se seguir, mas pode levar a consequências negativas.

As mensagens em *broadcast* são colocadas em uma memória comum que é lida cada vez que um agente é acordado, quer seja pela ativação direta, quer seja pela seqüência pré-estabelecida.

O término do processo de análise pode ser determinado pelo comportamento de qualquer uma de duas listas do sistema: VIU_TÉRMINO e NÃO_FEZ_NADA. O conteúdo da primeira é alterado sempre que um agente consegue estabelecer suas conexões e gera uma mensagem de término. Depois de emitir esta mensagem ele verifica se existe alguma mensagem de término de outro agente na lista. Caso exista uma mensagem deste tipo ele coloca seu identificador na lista VIU_TÉRMINO ou solicita o reagrupamento do sistema, de acordo com seu estado interno. Quando um agente não realiza nenhuma atividade durante sua ativação, ele insere seu identificador na lista NÃO_FEZ_NADA, que é esvaziada toda vez que qualquer agente realiza uma operação. Quando uma das duas listas estiver cheia o processo de análise é considerado completo.

Algumas considerações sobre o sistema

Os agentes definidos misturam características reativas e cognitivas, tendo características em comum com o sistema PACO[28], demonstrando que a classificação entre reativo e cognitivo pode ser gradual.

O conceito de *Polaroid Word* foi estendido e novos fenômenos lingüísticos puderam ser tratados, mas alguns problemas persistiram devido à dificuldade em tratar estruturas sintagmais e ambigüidade categorial. Um dos problemas não resolvidos é o tratamento do *pp-attachment* [25], que acaba sendo resolvido de acordo com a leitura preferencial do sistema, derivada da busca em profundidade das mensagens diretas. A segunda interpretação estrutural do fenômeno só é executada através de uma solução *ad hoc*.

O próprio autor define como trabalho futuro o estudo e definição de um novo protocolo de resolução de conflitos entre agentes. A persistência deste problema, no entanto, pode não ser resolvida por um protocolo melhor de negociação, uma vez que ele é decorrente direto da granularidade dos agentes. Isto nos leva a supor que a determinação da granularidade seja uma etapa anterior à da definição dos protocolos de comunicação e interação. Isto não significa determinar as capacidades dos agentes (definí-los), mas sim um superconjunto de capacidades do qual selecionaremos o conjunto adequado, depois de definirmos os protocolos de comunicação e interação.

3.5.2 Utilização do Paradigma Multi-Agentes no Processamento da Linguagem Natural: um Modelo Voltado à Resolução da Ambigüidade Léxica Categorical na Língua Portuguesa

Este trabalho[22] busca resolver a ambigüidade categorial, um problema relevante para o PLN, enquanto considera as limitações de desempenho dos sistemas tradicionais de processamento.

Para superar as dificuldades de comunicação de dados entre etapas de processamento não adjacentes, assim como os problemas trazidos pelas estruturas de controle centralizado de processamento, é feito um estudo sobre os sistemas multi-agentes voltados ao PLN e apresentada uma sociedade baseada em agentes lingüísticos autônomos e distribuídos.

O problema da ambigüidade foi abordado em relação aos diversos níveis lingüísticos e sob vários critérios diferentes [23] para tratar construções ambíguas utilizando a interação entre os diversos níveis lingüísticos.

Metodologia de desenvolvimento

O aspecto cognitivo da sociedade foi analisado para tentar modelar a interação dos níveis semântico, sintático e neuropsicológico e para resolver os problemas de ambigüidade e de preferência de leitura. Foram considerados um modelo baseado em contexto, um modelo exaustivo de processamento e um modelo de acesso ordenado do significado.

No modelo de dependência de contexto apenas um significado é considerado pelo leitor para as palavras do texto. O significado escolhido para a palavra é determinado pelo contexto onde a palavra é encontrada.

Já no modelo exaustivo, todos os significados possíveis de uma palavra estão ativos e mantidos em memória ao mesmo tempo, independentemente do contexto. Este modelo considera que o leitor é capaz de determinar quais os significados adequados para as palavras à medida em que as lê e foi o modelo adotado na construção do SMA.

O modelo de acesso ordenado indica que alguns significados são predominantes para as palavras, ou seja, elas preferencialmente possuem este significado. As demais possibilidades só são analisadas quando o significado preferencial não se encaixa no contexto.

Enquanto o modelo cognitivo determina o espaço de busca pelo significado, é necessário determinar quais informações lingüísticas devem ser consideradas para efetuar a análise. Desta forma são definidos os aspectos sintático e semântico do SMA. Note que a determinação destas características dependem do domínio do sistema e não da natureza dos agentes.

A abordagem sintaxista propõe que a ambigüidade categorial seja tratada de acordo

com restrições de co-ocorrência. Neste caso a rotulação das palavras¹³ pode ocorrer de uma maneira automática ou semi-automática através de métodos estatísticos ou heurísticos (simbólico), ou de acordo com regras gramaticais.

Já a abordagem semanticista, na maioria dos sistemas em que é utilizada [23], faz uso de primitivas de representação. O método mais simples é determinar uma restrição de seleção para o significado das palavras da sentença. Apesar destes significados preferenciais auxiliarem no processo, não são suficientes para solucionar todos os casos de ambigüidade, especialmente quando verbos polissêmicos estão envolvidos.

Outra possibilidade é a adoção de redes semânticas que possuam mecanismos de análise de contexto que possam confrontar as palavras ambíguas com as demais palavras ao seu redor[42]. Dada a complexidade destas redes, apenas um pequeno número de palavras foi codificada no SMA, restritas a um subconjunto de seu universo de significados possíveis, evitando uma explosão combinatorial [23].

Processo de análise

Enquanto o domínio do problema determinou quais as características necessárias para a resolução da ambigüidade, a pré-existência de um conjunto de sistemas de PLN indicou o reaproveitamento no projeto dos agentes. Estes sistemas, no entanto, sofreram modificações para refletir o modelo de agente desejado. Foram incorporados aos agentes uma identidade, uma coleção de capacidades, objetivos e a capacidade de descrever os demais agentes. Também foi codificado um comportamento, ou seja, um módulo que rege as reações do agente à medida em que as mensagens do SMA chegam.

Desta forma podemos considerar que o SMA resultante foi derivado dos requisitos do domínio, da granularidade dos sistemas pré-existentes e do modelo cognitivo dos agentes, sendo as camadas de comunicação e cooperação conseqüências destas escolhas.

Foi definido um agente morfológico que é o responsável pela consulta de cada palavra da sentença de entrada a um dicionário morfológico. Esta consulta pode ser total (toda a sentença) ou parcial (apenas uma palavra) e retorna um conjunto de significados para cada item consultado.

As capacidades e habilidades deste agente são representadas internamente por uma árvore *trie*, que funciona como índice, e um conjunto de descritores, que são as entradas do dicionário. Uma árvore deste tipo armazena as palavras letra a letra, construindo um caminho da primeira letra, que funciona como raiz da entrada, até a última letra, cujo nodo possui um apontador para o descritor da palavra. Palavras que possuam o mesmo prefixo compartilham o início do caminho, como no caso de "bola" e "bolo", que compartilham o caminho "b-o-l".

¹³Do inglês *part-of-speech tagging*.

Os itens léxicos podem ser inseridos como formas ou bases neste dicionário. No caso da inserção como forma, a palavra é codificada direta e completamente nos nodos da árvore *trie* e o fim do caminho aponta para um descritor. Já no caso das bases apenas o prefixo invariável da palavra é inserido na árvore. O descritor apontado no final do caminho de uma base contém, além dos dados contidos para as formas, um apontador para um modelo. Os modelos contém todas as terminações que podem ser associadas à base que os apontam e a junção da base com qualquer uma das terminações forma uma palavra válida. Cada uma das terminações também possui um descritor que contém dados sobre gênero, número, grau, etc. Este mecanismo nos permite inserir apenas a base de um verbo e associar esta base a um modelo adequado para a família deste verbo, codificando toda as flexões verbais imediatamente.

Também foi definido um agente sintático cujo funcionamento se baseia nas *Tree Adjoining Grammars* (TAG) [47], conforme previamente discutido. Este agente é responsável pelas descrições sintáticas do sistema e se baseia nos sistemas definidos em [56].

O terceiro agente que completa o sistema é semântico e se baseia na Teoria do Léxico Gerativo, como proposto por Pustejovsky [70]. Esta teoria determina a existência de pelo menos quatro níveis de representação: estrutura de argumentos, estrutura de eventos, estrutura qualia e estrutura de herança lexical. A estrutura léxica determina como um item se associa com uma expressão sintática. A estrutura de eventos determina quais eventos estão associados ao item ou expressão. A estrutura qualia possui os atributos essenciais de um objeto, eventos e relações que definem os itens léxicos [34], e a estrutura de herança estabelece as relações entre os itens léxicos e os demais conceitos do léxico.

A estrutura qualia é um sistema relacional que descreve a semântica dos itens léxicos através da codificação de informações sobre os constituintes, espaço, forma, superfície, processo de criação, etc. Estes aspectos são organizados em quatro níveis de significado do objeto: constituinte, que descreve as partes de um objeto; formal, que distingue um objeto dentro de um grupo maior de objetos; télico, que descreve qual a função ou papel esperado do objeto; e agentivo, que descreve o processo de surgimento do objeto.

Esta rede foi codificada com o auxílio de uma árvore *Trie* e descritores, de forma similar ao agente morfológico[4] [3]. As diferenças concentram-se nos descritores semânticos e na presença de listas de referências semânticas para codificar cada um dos aspectos da estrutura qualia no total de quatro listas, uma lista para a estrutura de argumentos e outra lista para a estrutura de eventos.

A síntese da resposta

A metodologia de resolução da ambigüidade categorial desta proposta é, na verdade, um protocolo de comunicação e cooperação entre agentes baseados em sistemas pré-existentes. A implementação deste protocolo exigiu a incorporação de novas primitivas de comuni-

cação, cooperação e interação com o ambiente do SMA para transformar estes sistemas previamente existentes em agentes.

Após ter sido determinado o domínio de cada agente, foram definidas as primitivas de comunicação de cada um deles. Estas primitivas também definem os conjuntos A_{cap}^i e A_{sub}^i de cada um destes agentes.

Cada um dos agentes desta sociedade possui uma espécie de caixa postal para comunicação de forma assíncrona. O protocolo de comunicação do sistema é derivado do KQML [36] e se utiliza apenas de um subconjunto de primitivas que, segundo o autor, melhor representam intenções e cooperação, tais como: *ask-if*, *ask-one*, *tell*, *reply*, *broadcast*, etc.

Segundo [23], o uso destas primitivas dispensa a definição de um protocolo específico para requisitar informações ou enviar resultados parciais de um agente para o outro. Existe, entretanto, um analisador de mensagens incorporado em cada agente que é encarregado de analisar e decompor as mensagens que chegam de acordo com a estrutura interna da caixa postal de cada agente. Este módulo funciona como um especialista sobre as possibilidades do agente do qual faz parte.

Cada agente possui uma estrutura de dados que codifica seus objetivos, habilidades e seu registro das capacidades e descrições de todos os demais agentes da sociedade [80].

Devido ao fato dos agentes serem heterogêneos faz-se necessária a definição de um protocolo de comunicação para regular as interações do SMA. Este protocolo foi implementado também com as primitivas de comunicação do KQML e, para resolver os conflitos entre as hipóteses lingüísticas levantadas pelo processamento, baseia-se no protocolo de aprendizado cooperativo [57].

Os resultados obtidos pelo sistema, no entanto, não levam em consideração a tentativa de prova de uma mesma hipótese contra múltiplos dados. O SMA deve atingir um consenso entre as hipóteses, por vezes heterogêneas, levantadas por seus agentes.

Aspectos de implementação

O SMA foi implementado utilizando um ambiente de objetos ativos denominado DPSK+P [14] e um ambiente de execução de multi-agentes denominado MASENV [81] rodando em ambiente Sun-Solaris.

Foi desenvolvido, além dos agentes lingüísticos, um gerenciador de mensagens externas ao sistema. Este gerenciador é um agente capaz de permitir a interação do sistema SMA com entidades externas, incluindo uma aplicação tradicional. Quando uma aplicação é executada, externa ao SMA, uma mensagem padrão é enviada através do gerenciador. Se a mensagem for reconhecida por algum dos agentes, um processo é disparado pela sociedade até que a solução seja encontrada, ou até que seja determinada a agramaticalidade da sentença analisada.

Este gerente de comunicações também é responsável pela comunicação dos agentes com a interface do sistema. Esta interface também é utilizada para verificar e depurar o comportamento da sociedade. Ela permite visualizar gráfica ou textualmente as mensagens trocadas entre os agentes, mostrar os fenômenos lingüísticos identificados, a requisição interna e externa de serviços, etc.

Algumas considerações sobre o sistema

Ao optar pelo reaproveitamento de sistemas anteriores e pela coincidência entre os domínios lingüísticos e dos agentes do sistema, o modelo final tem uma granularidade baixa, limitando a capacidade de paralelização da análise. Em sentenças simples o processamento obedece a seqüência de análises morfológica-sintática-semântica.

Já em situações onde existam fenômenos lingüísticos mais complexos, o agente sintático é capaz de determinar a existência de um problema e solicitar que os agentes morfológico e semântico executem tarefas específicas. Este tipo de decomposição de tarefas é fortemente cognitivo [22] [23] e depende do problema (fenômeno lingüístico) para sintetizar uma solução. É da negociação realizada para a solução do problema que surge a resposta final do sistema.

Capítulo 4

Uma proposta de arquitetura e sua implementação

“Todo programa tem pelo menos um *bug* e pode ser encurtado em uma instrução de máquina. Daí, por indução, podemos concluir que todo programa pode ser reduzido a uma instrução que não funciona.”

anônimo

“As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

Maurice Wilkes
designer of EDSAC,
on programming, 1949

Os capítulos anteriores abordam o processamento da língua natural através da descrição de três formalismos lingüísticos centrados na sintaxe e dos sistemas computacionais que os implementam. Buscou-se comparar as propostas de cada sistema com o paradigma de programação e teoria lingüística vigentes sem, no entanto, tentar construir uma explicação definitiva sobre os porquês de suas diferenças.

Este capítulo traz o resultado deste estudo anterior na forma de um modelo de processamento que, se não resolve todos os problemas conhecidos e levantados, abre novas possibilidades de pesquisa. Este modelo baseia-se no paradigma de multi-agentes e tenta se aproveitar de algumas características das TAGs e das BMGs dentro de um ambiente

distribuído de processamento.

O modelo que será apresentado foi prototipado e está plenamente funcional na forma de um agente sintático. Para sua implementação foi utilizada a linguagem de programação C^{++} . Para aumentar a portabilidade foi obedecido o padrão ANSI 2.0 e nenhuma interface gráfica é provida, o que nos permitiu compilar e utilizar com sucesso o sistema nos ambientes Linux, Windos9X e Solaris sem a necessidade de alterar o código fonte¹. O apêndice A mostra exemplos de gramáticas, sentenças de entrada e derivações geradas pelo protótipo, e fornece algumas informações úteis voltadas ao processo de implementação.

A apresentação que se segue não foge da organização geral da descrição de cada formalismo, abordando a descrição lingüística e o mecanismo de derivação correspondente. Também são feitas considerações sobre o modelo e sua incorporação em um SMA heterogêneo.

4.1 Motivação — Balanceando distribuição e simplicidade

O modelo foi motivado parcialmente pela complexidade do PLN visto como um problema computacional. Foram citados, em conjunto com a descrição do formalismo, trabalhos que demonstram que a complexidade computacional do processamento de uma XG é a mesma dos problemas NP-completos [40]. Em relação às BMGs, devido à ausência de uma definição mais precisa, não foi possível afirmar o mesmo, mas foram apontados indícios de que o problema de seu processamento também é NP-completo. Dos formalismos abordados o mais bem posicionado nesta questão são as TAGs. Mesmo assim, seu custo computacional atinge $O(n^6)$, o que não pode ser considerado ideal, ainda mais porque o algoritmo utilizado [75] [92] permite apenas árvores binárias na derivação, entre outras limitações [71].

Devemos observar, no entanto, que apesar do custo computacional existem sistemas computacionais em uso baseados nestes formalismos. Isto parece indicar que, em sentenças de comprimento razoável para um leitor humano, a análise pode ser feita em um tempo aceitável. Além do tamanho das sentenças de entrada, existem três alternativas principais que podem explicar esse fato: as provas matemáticas de complexidade estão erradas; os dados publicados sobre os sistemas baseados nos formalismos estão incorretos; e o conjunto de gramáticas utilizado na prática influencia o custo do processamento.

A primeira alternativa é improvável, porém aceitável. O primeiro cálculo de custo computacional apresentado para as TAGs fornecia um custo de $O(n^4)$ [47], posteriormente

¹Espera-se que o protótipo funcione em qualquer plataforma que possua um compilador compatível com o padrão C^{++} ANSI 2.0, mas apenas estes três ambientes foram testados.

corrigido para $O(n^6)$ [75]. A segunda alternativa não pode ser verificada sem a análise de cada sistema, o que teria um custo proibitivo e dificilmente traria resultados diferentes dos conhecidos uma vez que estes sistemas foram apresentados em diversos congressos e sofreram o crivo de pesquisadores ao redor do mundo.

A alternativa restante é a de que as gramáticas influenciam diretamente o sucesso dos sistemas. É preciso cuidado ao considerarmos esta hipótese como verdadeira. Suponhamos que exista um formalismo cuja capacidade de reconhecimento seja tanto forte quanto fracamente igual às descrições destas gramáticas, ou seja, que exista um formalismo que consiga descrever exatamente estas gramáticas e nada mais. Este formalismo imaginário teria, possivelmente, um custo menor que o dos formalismos originais, dada a premissa de que as gramáticas influenciam diretamente no custo do processamento.

Este custo inferior, no entanto, não pode ser extrapolado para o PLN como um todo. Não pode-se afirmar que uma língua natural pode ser reconhecida com um custo menor que o dos formalismos originais, ou seja, não se sabe ainda se o tratamento da língua natural é um problema intrinsecamente NP-completo ou não. Isto ocorre porque, até onde se pôde apurar, nenhum sistema computacional existente pode tratar todos os problemas lingüísticos, ou seja, não existem garantias de que nosso formalismo imaginário consiga tratar uma língua natural plenamente. Desta forma o custo deste formalismo imaginário e do tratamento da língua como um todo não são necessariamente iguais.

Por outro lado, podemos concluir que o desenvolvimento de um sistema simplificado, que não trata todos os problemas da língua, é uma proposta válida. Todo o embasamento lingüístico das TAGs, por exemplo, depende da hipótese das gramáticas para o PLN não necessitarem de toda a capacidade descritiva fornecida pelos formalismos NP-completos, ou seja, de que basta tratar um subconjunto da língua que os formalismos NP-completos tratam.

Esta simplificação, que é essencialmente lingüística, está relacionada diretamente com o grau de sensibilidade ao contexto de cada formalismo, fator essencialmente matemático e computacional. Um indício desta relação é o fato de uma gramática livre de contexto poder ser tratada em tempo polinomial [31] [6]. As TAGs, usando esta lógica, evitam a NP-completude porque possuem uma capacidade de reconhecimento pouco maior do que a das gramáticas livres de contexto².

A definição de um tipo de gramática, no entanto, não pode considerar apenas o custo de seu processamento. O objetivo principal de um sistema de PLN é ser capaz de analisar e/ou reconhecer os fenômenos lingüísticos. Isto quer dizer que não se pode simplificar a descrição lingüística até o ponto onde ela deixa de ser útil. Deve-se buscar um compromisso entre a simplicidade gramatical e o tipo de resposta que se deseja do sistema.

²Este formalismo, no entanto, ainda não foi utilizado com sucesso para o tratamento de coordenação, o que indica que pode não ser suficiente para reconhecer uma língua natural completa.

Além da utilização de gramáticas simplificadas, outro paliativo para o problema da complexidade é utilizar unidades de processamento menores, tais como a sentença. A capacidade do sistema analisar um texto completo ficaria, neste caso, dependente de módulos externos. O sistema deve, portanto, ser capaz de operar cooperativamente.

Mesmo na análise de sentenças, as grandezas de tempo podem ser significativas, o que leva a considerar uma paralelização do algoritmo de análise. Lidar com paralelismo, não importa qual técnica seja utilizada, envolve a definição de uma metodologia para escalonar o processamento. No caso dos SMAs, isto não é diferente, apenas mais central: é neste ponto que se define qual o tipo de comunicação — e conseqüentemente cooperação — desejada entre os agentes. A divisão do problema é uma decorrência desta decisão.

4.2 Uma visão geral do modelo

O que nos leva a propor este modelo são as vantagens que podem ser obtidas com a utilização de um formalismo gramatical simples, paralelizável, e que favoreça o trabalho cooperativo entre agentes heterogêneos. Vale lembrar, no entanto, que a proposta que estamos descrevendo não é uma tentativa de gerar um modelo definitivo ou de criar um arcabouço para o PLN. Esta descrição reflete tão somente as soluções encontradas durante a criação do protótipo implementado que, por sua vez, teve suas características definidas a partir dos trabalhos relacionados aos projetos Lexis [56] [43] e Nalamas [26] [27]. Estas características também buscam contornar os problemas encontrados no tratamento de elipses [60] e nos dois sistemas apresentados anteriormente como estudo de caso [25] [22].

A cooperação desejada, como parte essencial das características do modelo, deve ocorrer durante o processo de derivação para permitir que informações de outros níveis de processamento que não o sintático sejam utilizadas. Um mecanismo tradicional não nos permite quebrar o processamento para amostrar seu desenvolvimento sem que exista uma completa compreensão de todo o processo. Um analisador sintático baseado em tabelas³ [5], por exemplo, pode ter sua derivação amostrada desde que todos seus dados mais o mecanismo de derivação sejam conhecidos. Em outras palavras, é preciso um outro analisador sintático completo para analisar a derivação.

Existem ainda os problemas práticos, tais como o compartilhamento de estruturas de dados, a necessidade de uma política comum de pilha de argumentos e um protocolo de interrupção que pare o processamento em um momento adequado, entre outros. É especialmente complicado modificar o comportamento da derivação neste modelo e o processo envolveria a reconstrução completa das tabelas e a reinicialização do processo.

Estes problemas podem ser contornados em grande parte separando os dados da análise

³Do Inglês *table driven*.

do mecanismo de inferência e tornando estes dados auto-contidos. Seria como fazer com que cada tabela de um analisador pudesse ser modificada sem a necessidade de alterar as demais e, ainda, manter todo o processamento prévio válido. As técnicas mais populares de análise, tais como o *chart parsing*, os analisadores sintáticos descendentes recursivos (ASDR), os analisadores baseados em tabelas e os autômatos de pilha, são inadequados para este objetivo porque mantêm seus dados de controle da derivação fortemente amarrados ao mecanismo de derivação [5]. No caso dos analisadores sintáticos dirigidos por tabelas estes mecanismos de controle são o analisador.

O modelo proposto, refletido em uma organização de agente, atinge estes objetivos adotando uma agenda de processamento, de forma similar às dos sistemas especialistas [7]. Sua organização geral, mostrada na figura 4.1, permite que o processamento seja iniciado por um usuário humano através de uma ordem em linha de comando ou por outros agentes. Neste segundo caso os agentes sintáticos ficam ociosos até que algum outro agente informe qual a gramática de entrada e passe a fornecer sentenças para análise.

Os componentes do modelo de agente estão separados em dois grandes grupos. Cada caixa de texto representa um objeto independente que pode ser substituído caso necessário. Estes objetos não são subagentes e possuem uma interface de comunicação baseada em métodos e predicados, como qualquer objeto. O grupo de objetos número 1 é responsável por funções auxiliares de leitura do texto e da gramática a partir de arquivos do sistema e só são necessários quando o agente é iniciado diretamente por um usuário humano através de linha de comando.

O módulo de carga do texto é responsável pela formatação do texto de acordo com o esperado pelo sistema e pode, caso necessário, comunicar-se com o sistema para resolver a ambigüidade categorial e coletar informações associadas às palavras da sentença de entrada.

O módulo responsável pela carga da gramática tem como função isolar o formato interno do sistema do restante do SMA. Substituindo este objeto por um outro qualquer é possível mudar a sintaxe da gramática do ponto de vista do SMA, desde que se mantenha o formato esperado pela interface de carga da gramática. Ele também é responsável pela validação da gramática e das referências cruzadas que possam vir a existir, conforme veremos mais adiante.

O segundo grupo de objetos é o núcleo do analisador. Este núcleo possui uma base de conhecimentos indexada e um analisador segmentado em estágios⁴ responsável pelo processo de derivação, além de uma interface de comunicação.

A interface de carga da gramática é o responsável pela indexação da base de conhecimento (gramática) e deve estar preparado para receber descrições adicionais do SMA. Esta organização permite que a gramática seja incrementada ou alterada durante o pro-

⁴Um *pipeline*.

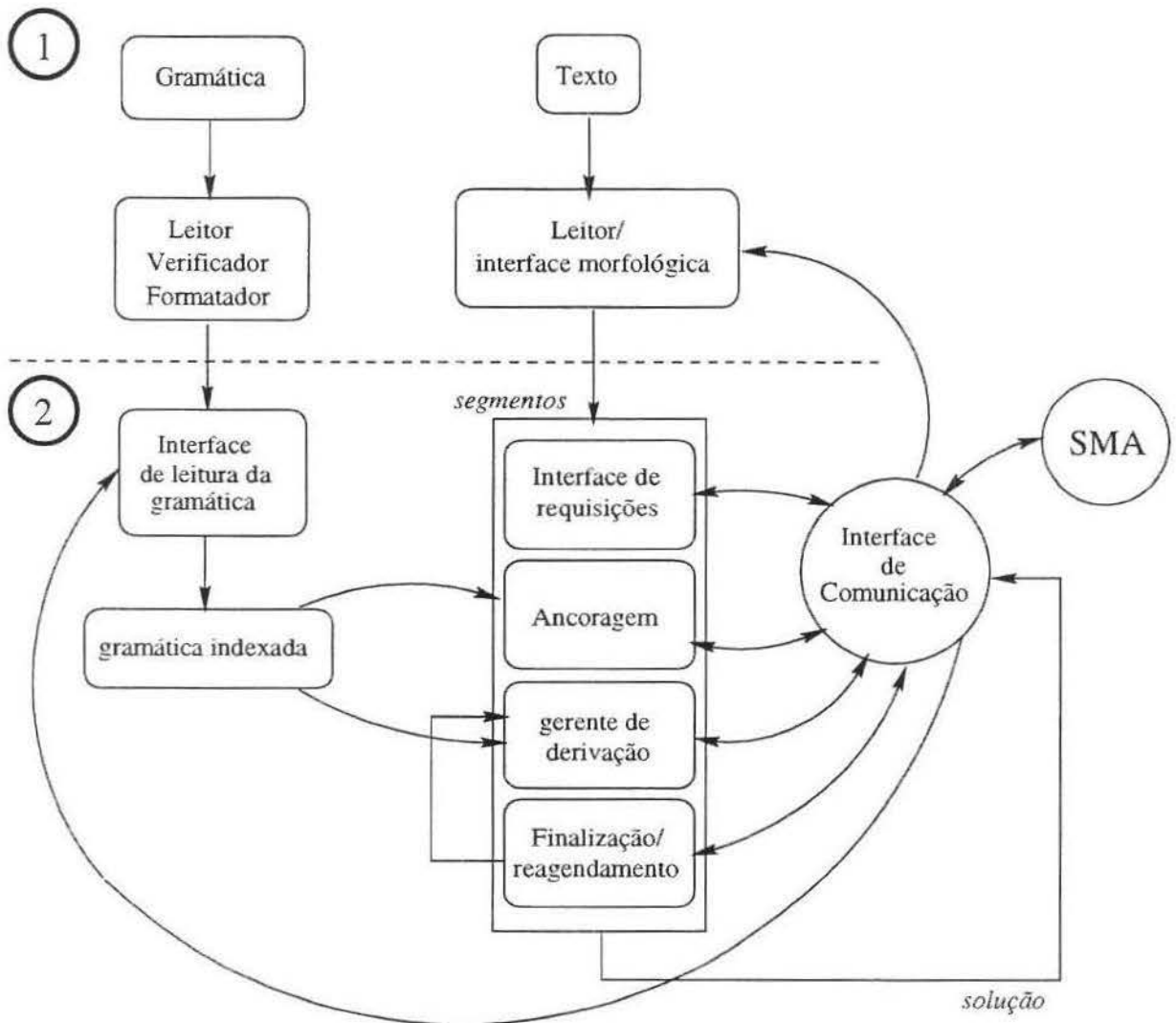


Figura 4.1: Organização geral de um agente.

cesso de derivação, processo este que é, obviamente, dependente do formalismo lingüístico adotado pelo sistema e que será descrito mais adiante.

O analisador de derivação possui quatro etapas de processamento e permite a reconfiguração do processamento através do reescalonamento de solicitações. Todas as etapas devem ser auto-contidas e cada módulo deste analisador deve ser capaz de distribuir sua carga de processamento para o SMA e receber solicitações deste mesmo SMA. Os dados de entrada de cada um dos módulos podem ser reconfigurados sem que isto afete os dados dos demais módulos, ou seja, cada solicitação é autocontida, tal qual um registro de ativação de uma função em um sistema sem variáveis globais.

Os objetos que implementam os estágios do analisador são responsáveis pela sincro-

nização e processamento das solicitações provenientes de etapas anteriores e da produção de solicitações para as etapas posteriores. Esta configuração os transforma em produtores e consumidores simultaneamente e, complementando este comportamento, também são responsáveis pelo controle de carga do sistema, retendo suas solicitações quando a etapa posterior estiver saturada e o SMA não puder ser acionado. Quando um destes módulos recebe uma solicitação e o sistema está saturado, ele deve ser capaz de armazenar a solicitação para um momento mais adequado.

O maior desafio na implementação deste modelo é definir um formato de representação auto-contido para as etapas intermediárias da análise sintática. Outro aspecto importante é a capacidade do SMA enviar e receber informações em cada etapa do processamento, o que também afeta a forma como os dados intermediários são armazenados.

Este modelo define, antes do formalismo gramatical, as capacidades de comunicação e cooperação desejadas para cada agente. É preciso, no entanto, compreender que tipo de informação as etapas intermediárias de processamento armazenam. Para tanto é necessário descrever o formalismo lingüístico adotado.

4.3 Em busca de um formalismo gramatical simples e efetivo

Satisfazer as primitivas de comunicação, por si só, não é o objetivo deste trabalho. O sistema deve ser capaz de reconhecer/analisar sentenças em português e prover maneiras simples de codificar o conhecimento lingüístico em gramáticas. A organização do modelo, no entanto, não permite que os formalismos estudados sejam implementados diretamente, uma vez que seus mecanismos originais de reconhecimento baseiam-se nos analisadores sintáticos tradicionais.

A alternativa mais direta é a construção de um analisador de acordo com o modelo que implemente um dos formalismos. Isto, no entanto, prejudicaria a utilização de características interessantes presentes em um mas não em outro formalismo. A alternativa adotada foi a criação de um formalismo novo, derivado do estudo anterior.

Uma das características desejadas para este novo formalismo é o domínio de localidade estendido das TAGs, que permite a manipulação direta da capacidade gerativa forte das gramáticas. Para tanto, optou-se por descrições gramaticais estruturadas, mais especificamente árvores, divididas em dois grupos: as árvores iniciais e as árvores auxiliares.

Assim como na definição original das L-TAGs, as árvores iniciais fornecem uma descrição mínima para a estrutura frasal que um determinado constituinte, geralmente um verbo, determina. Cada uma das árvores iniciais deve prover uma, e apenas uma, posição para a colocação do item léxico que a seleciona, ou seja, para a colocação da âncora da

árvore. Não pode existir mais de uma âncora por árvore. A figura 4.2 mostra um exemplo de árvore inicial para o verbo “comer”.

Cada nodo da árvore da figura 4.2 é representado por um rótulo. O sistema não especifica um conjunto rígido de rótulos, mas obtém seus nomes de um arquivo de entrada ou do SMA durante a leitura da gramática. Isto é importante porque permite que qualquer conjunto de rótulos seja utilizado sem a necessidade de alterar o mecanismo de derivação. No exemplo, “S” é um acrônimo de sentença, “SN” de sintagma nominal, “SV” de sintagma verbal, “Det” de determinante, “Subst” de substantivo e “V” de verbo. Do ponto de vista do analisador isto é indiferente, desde que o conjunto de rótulos da gramática seja coerente com o utilizado para classificar as palavras da sentença de entrada. Uma palavra só pode ser acomodada na estrutura (consumida) se sua categoria tiver um nome idêntico ao rótulo do nodo da árvore, conforme o conceito de pré-terminal [56].

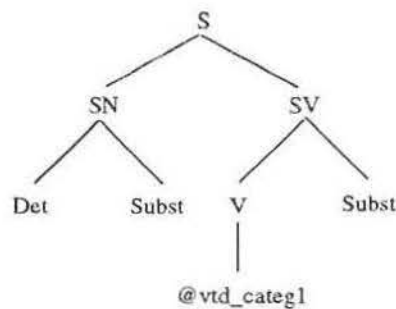


Figura 4.2: Árvore inicial do verbo comer.

Existe também um elemento denominado “@vtd_categ1”. Este é o ponto onde deve ser colocada a âncora léxica que seleciona a estrutura. Esta âncora é uma palavra da sentença de entrada do sistema e é usada para selecionar a árvore inicial. No caso a palavra foi o verbo “comer”, mas uma mesma estrutura pode ser ancorada por um conjunto de itens léxicos, assim como um mesmo item léxico pode aparecer em mais de uma estrutura. A forma geral da definição de uma âncora é:

$$@\text{nome_âncora} = \{\text{item}_1, \dots, \text{item}_{n-1}, \text{item}_n\} : \{\text{árvore}_1, \dots, \text{árvore}_{n-1}, \text{árvore}_n\}.$$

Ainda no exemplo da figura 4.2, qualquer item especificado na âncora “vtd_categ1” pode selecionar a estrutura e, por consequência, aparecer na posição de ancoragem. Isto permite que definamos estruturas capazes de representar categorias de itens. Um exemplo de âncora com estas características pode ser

$$@\text{vtd_categ1} = \{\text{comer}, \text{comeu}, \text{beber}, \text{pintou}\} : \{\text{vtd1}\},$$

onde “vtd1” é o nome identificador da árvore inicial selecionada pelas palavras “comer”, “comeu”, “beber” e “pintou”.

Associada à definição de uma árvore inicial ainda está a noção de estrutura mínima representando quais constituintes acompanham uma determinada âncora. Esta noção, no entanto, é ligeiramente diferente da adotada nas TAGs, como veremos mais adiante.

Assim como nas TAGs, as árvores auxiliares são utilizadas para expandir as descrições das árvores iniciais. No modelo aqui proposto, no entanto, elas não estão relacionadas com a recursão. Comparando a categoria das palavras com os rótulos da árvore inicial da figura 4.2 vemos que é possível acomodar uma sentença como “o menino comeu doce”. Por outro lado, não se pode acomodar mais nenhuma informação como, por exemplo, um local (“o menino comeu doce no quarto”) ou um instrumento (“o menino comeu doce no quarto com a colher da vovó”).

A figura 4.3 nos mostra duas árvores auxiliares mais a estrutura do exemplo anterior. Estas árvores, quando combinadas através da adjunção, permitem incrementar as informações que acompanham o verbo “comer”, que ancora a árvore inicial. Estas informações são opcionais e podem estar relacionadas, por exemplo, com o local ou instrumento. Codificar as informações opcionais em árvores auxiliares facilita o tratamento do caso geral sem abrir mão do controle sobre a estrutura de derivação gerada.

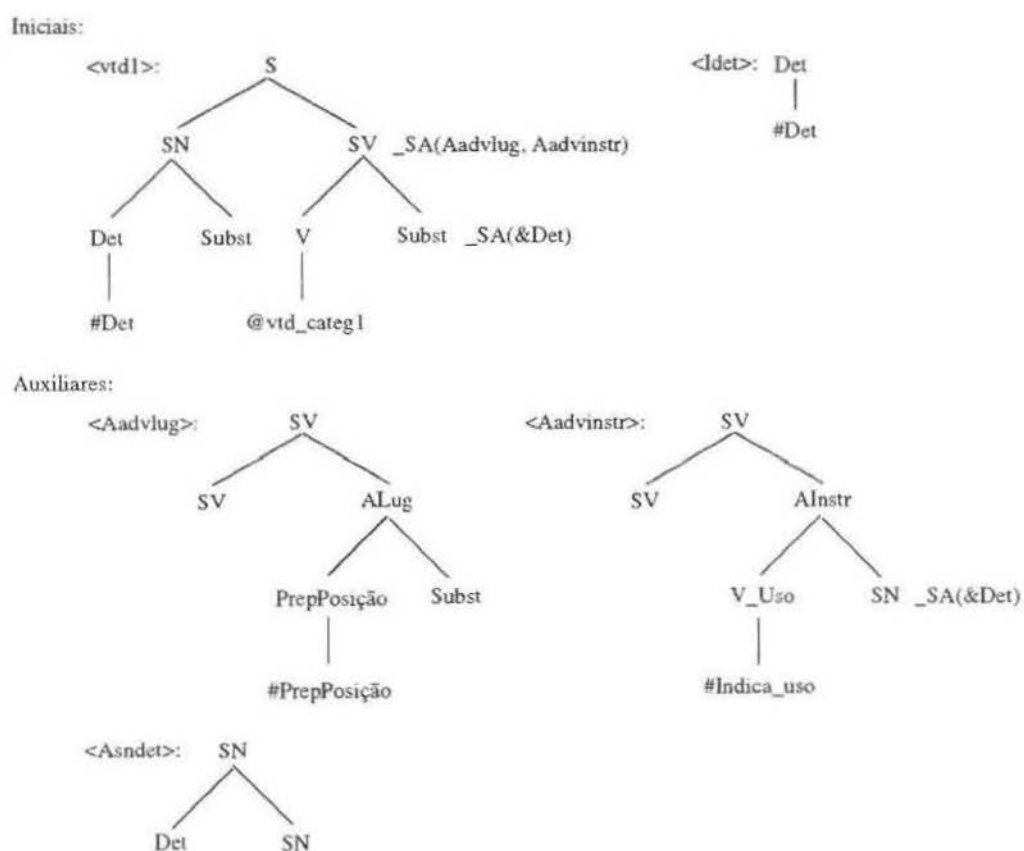


Figura 4.3: A expansão de uma árvore inicial.

Esta mesma figura traz ainda alguns elementos originais em relação às TAGs. Ela possui uma marca “_SA” seguida de uma lista de identificadores entre parênteses ao lado do nodo “SV”. Esta marca identifica um *seletor de adjunção*, que define um conjunto de árvores que podem ser adjuntadas naquele nodo. Ao contrário das TAGs originais, onde qualquer nodo interior ou exterior não-marcado para substituição pode sofrer adjunção, apenas os nodos com seletores de adjunção explícitos podem sofrer adjunção. Além disso, apenas as árvores auxiliares cujo identificador está na lista do seletor podem ser utilizadas.

Note que a implementação “oficial” das TAGs [92], define que uma adjunção é executada quando a unificação dos traços *top* e *bottom* de um nodo falha. Em outras palavras, ela ocorre quando os predicados codificados nestes traços não são satisfeitos. Dirigir a derivação de acordo com estes predicados é incompatível com o modelo proposto porque exige a utilização de uma estrutura global para armazenar os dados de controle. Esta estrutura dificultaria o envio das solicitações de análise para o SMA a partir de segmentos intermediários do analisador, tornando cada solicitação dependente do agente que a recebeu. Além disso, a descrição do que fazer quando se encontra um determinado conjunto de traços como, por exemplo, adjuntar ou desistir, depende de uma lógica codificada no analisador e não na gramática, o que é conflitante com a idéia de independência do mecanismo de inferência em relação à gramática. Ao trocarmos o processamento dos predicados pelas listas de adjunção também ganhamos em clareza e simplicidade.

Existem dois tipos de seletores de adjunção, um para a adjunção obrigatória e outro para a opcional. O operador “_SA” indica uma adjunção opcional, ou seja, não é obrigatório que ocorra uma adjunção no nodo assinalado. Já o operador “_OA” especifica uma adjunção obrigatória, ou seja, pelo menos uma das árvores auxiliares especificadas deve ser adjuntada. Como a adjunção é uma operação entre árvores, os seletores podem aparecer também nos nodos internos e não somente nas folhas das árvores da gramática.

Para tornar a notação mais concisa foram definidos os operadores “#” e “&”, também presentes na figura 4.3. O operador “&” precede o identificador de lista de árvores auxiliares. Ele substitui a lista de identificadores nos seletores da adjunção e evita que a descrição da árvore fique cheia de identificadores, o que dificultaria a leitura. A forma geral de uma lista de árvores é:

$$\&lista_árvores = \{ árvore_1, \dots, árvore_{n-1}, árvore_n \}.$$

Podemos definir, por exemplo, uma lista $\&Det = \{ Asndet \}$. É importante lembrar que não é permitido incluir um identificador de árvore inicial como, por exemplo, “Idet” nesta lista porque as árvores iniciais não possuem um nodo pé adequado, ou seja, não podem ser adjuntadas.

Já o símbolo “#” precede um identificador de lista de palavras. Neste caso o nodo pode acomodar uma palavra que esteja na lista definida por este identificador e não mais

as palavras com a categoria igual ao rótulo do nodo. Em outras palavras, quando o analisador encontra um identificador de lista de palavras ele compara a palavra, e não sua categoria, com os rótulos contidos na definição da lista. A forma geral de uma lista de palavras é

$$\#lista_palavras = \{palavra_1, \dots, palavra_{n-1}, palavra_n\},$$

e no exemplo do menino comendo doce, podemos definir uma lista

$$\#Det = \{o, os, a, as, um, uns, uma, umas, aquele, aqueles, aquela, aquelas\}$$

para guardar os determinantes esperados; uma lista

$$\#PrepPosição = \{na, no, nas, nos\}$$

para guardar as preposições; e, ainda, uma lista

$$\#Indica_uso = \{usando, com\}$$

para guardar as preposições indicando uso. O analisador considera estas listas como conjuntos de identificadores que não precisam ter nenhuma coerência gramatical.

A responsabilidade de manter os rótulos dos nodos e as palavras das listas coerentes com a grafia e com a classificação gramatical das palavras é de quem cria a gramática. O sistema simplesmente compara rótulos.

4.3.1 Conseqüências das modificações para o novo formalismo

Como dissemos anteriormente, os predicados não foram incorporados no nosso novo formalismo, deixando de influenciar o processo de derivação. Isto faz com que o sistema deixe de contar com as informações que regulam a adjunção. Estas informações permitem que os sistemas baseados na definição original das TAGs definam estruturas bastante elementares e controlem sua expansão através da verificação destes predicados. Por exemplo, uma produção livre de contexto " $S \rightarrow SNSV$ " pode estar acompanhada de predicados que obriguem a existência de um determinado contexto ao redor de cada nodo do lado direito da produção.

Para compensar a perda destas informações, definiu-se que as adjunções só ocorrem onde forem explicitamente indicadas pelos seletores de adjunção. Esta abordagem, no entanto, faz com que o sistema utilize apenas informações *top-down* durante a derivação e, como conseqüência, analise um conjunto de alternativas que seriam descartadas por falta de informações provenientes das palavras da sentença de entrada.

Por outro lado, é possível contribuir com o analisador evitando parte das adjunções. Para que isto ocorra é preciso que as árvores iniciais sejam mais detalhadas que as encontradas em [92]. A própria árvore definida na figura 4.2 obedece este preceito.

O detalhamento das estruturas aumenta o número de estruturas ancoradas porque nem todos os constituintes da estrutura frasal precisam estar realizados. Por exemplo, o determinante de um sintagma nominal pode ser omitido no português falado no Brasil. Pode-se dizer “João caiu” ao invés de “O João caiu”. Para amenizar este problema optou-se pela definição de um marcador de nodo opcional, como se pode ver na figura 4.4.

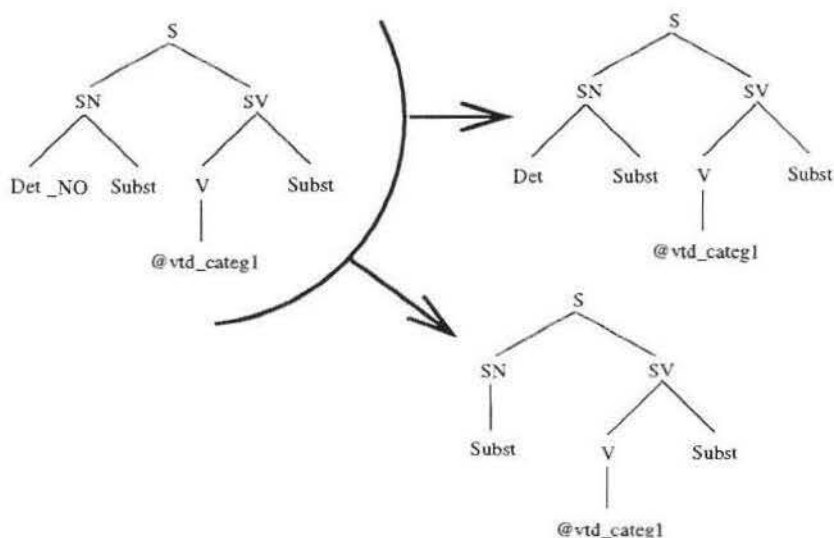


Figura 4.4: Nodos opcionais.

Qualquer nodo de uma árvore, interno ou externo, exceto a raiz, pode ser marcado como opcional. É possível marcar um nodo que domine a âncora como opcional, mas isso não modifica o comportamento do analisador. A presença da âncora continua sendo obrigatória na sentença de entrada.

O operador de adjunção foi modificado para refletir a perda dos predicados. Agora o seletor de adjunção de um nodo deslocado, caso exista na posição original, deixa de ser válido na sua nova posição, ao contrário do que ocorria com os predicados, que eram recombinações [89]. Além disso, qualquer seletor definido no nodo pé da árvore auxiliar é ignorado, como mostra a figura 4.5.

Esta nova definição impede a recursão na adjunção, o que é coerente com o tratamento dado às árvores auxiliares, que agora são linguisticamente definidas como uma construção frasal opcional relacionada com uma árvore inicial específica em um nodo determinado.

A operação de substituição, compensando as modificações sofridas pela adjunção, pode ocorrer em qualquer nodo externo da árvore, desde que não esteja marcado para adjunção, em uma inversão de prioridade das operações em relação à definição original.

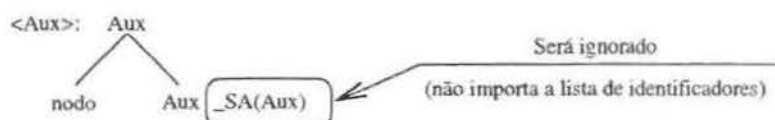


Figura 4.5: O seletor do nodo pé de uma árvore auxiliar é sempre descartado.

4.3.2 A linguagem de troca do analisador segmentado

O formalismo lingüístico, conforme definido até aqui, não possui regras implícitas que precisem de um controle externo ou de variáveis globais. A figura 4.6 nos mostra um exemplo de como podemos representar uma árvore qualquer de nosso formalismo em uma notação pseudo-funcional.

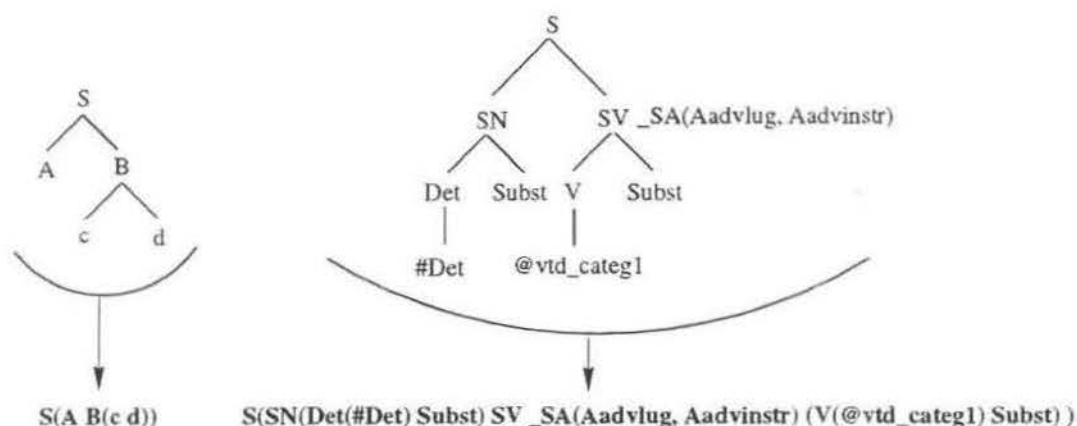


Figura 4.6: A linearização de uma árvore de derivação.

Esta representação pode ser transmitida mesmo pelo mais rudimentar mecanismo de comunicação, que é a transferência de linhas de texto. Ela também é independente dos demais segmentos do analisador e tem como único argumento implícito a gramática a partir da qual foi montada. Esta gramática deve ser comum entre o agente remetente da solicitação e o agente destinatário. Voltaremos a abordar outros aspectos do compartilhamento de gramáticas mais adiante.

O importante aqui é perceber que a representação intermediária adotada é independente, auto-contida e pode acomodar qualquer combinação de operadores, marcadores de opcional e descrição estrutural que possam aparecer em uma árvore.

4.3.3 O texto de entrada e os traços morfo-sintáticos

O sistema permite que um conjunto de informações individuais acompanhem as palavras da sentença de entrada. Estas informações, com exceção da categoria, não são manipuladas, apenas armazenadas para o caso de algum agente (o responsável pela unificação,

por exemplo) solicitá-las. Estas informações são idealmente traços morfo-sintáticos, mas não necessariamente. Podem ser qualquer outra informação que possa ser armazenada na forma geral

$$\text{palavra categoria } [\text{traço}_1, \dots, \text{traço}_{n-1}, \text{traço}_n],$$

com cada traço podendo assumir uma das formas:

- *Identificador*. Apenas um identificador qualquer. O sistema não entende o que os traços significam. Ele apenas os transporta e fornece para o SMA quando o texto é solicitado. Pode indicar uma característica constante como, por exemplo, em “garoto substantivo [masculino, singular]”, onde as constantes “masculino” e “singular” estão associadas à palavra “garoto”, de categoria “substantivo”.
- *Identificador = Identificador*. Pode dar valor a uma variável como, por exemplo, em “garotinho substantivo [masculino, singular, grau=diminutivo]”, onde a palavra “garotinho”, de categoria “substantivo”, tem seu “grau” determinado como “diminutivo”.

Com este tipo de anotação podemos fornecer informações localizadas para agentes que não façam parte da subsociedade que trata da análise sintática. Entre outros usos, é possível fornecer os dados para que a unificação seja executada externamente. Esperar que um agente externo complemente o trabalho do analisador sintático é coerente com a filosofia dos SMA. Seria um contrasenso propor uma organização deste tipo e esperar que apenas um agente resolva todos os problemas lingüísticos.

A capacidade de anotar os nodos folhas para acomodar os traços vindos das palavras da sentença de entrada foi generalizado para todos os nodos da árvore de derivação. Estas anotações são codificadas na gramática e acompanham a movimentação dos nodos durante as operações que ocorrem durante uma derivação. A única exceção é o nodo raiz de cada árvore auxiliar, que não pode ser anotado.

A figura 4.7 nos mostra uma árvore com nodos anotados e sua forma linearizada, de acordo com o formato utilizado nos segmentos do analisador de derivação, para a sentença “o vendedor era chato”. As anotações, fornecidas pelo SMA ou pelo arquivo de entrada de texto, sofrem os mesmos deslocamentos que seu nodo original. Quando adjuntamos uma árvore sobre um nodo qualquer, este nodo é deslocado para a posição do nodo pé da árvore auxiliar adjuntada e leva consigo os traços.

Como nenhuma alteração que venha a ocorrer com estas informações pode afetar o processo de derivação seu uso não é limitado. O processo de unificação, no entanto, é uma tarefa bastante provável e, por isso, foi desenvolvido um mecanismo para que o autor da gramática controle como as informações devem ser utilizadas através de uma simplificação do conceito de ilha das BMGs.

ou

$$\text{FREE}[\text{traço}_1, \text{traço}_{n-1}, \text{traço}_n].$$

Os identificadores de traços são sintetizados, sendo propagados para o nodo pai imediato de acordo com os especificadores. Quando o especificador for um “FREE”, os traços identificados na lista devem ser propagados enquanto todos os demais são bloqueados. Já no caso de “BLOCK”, os traços identificados pela lista são bloqueados e todos os demais são propagados. Esta lista de “propagação” não é obrigatória.

- *unificação* é um especificador de quais traços devem ser verificados pelo analisador externo de unificação neste ponto da árvore. Este item também é opcional e pode assumir a forma

$$\text{UNIFY}[\text{traço}_1, \text{traço}_{n-1}, \text{traço}_n].$$

Um exemplo de como os traços definidos na gramática e associados às palavras da sentença de entrada são codificados pode ser visto na figura 4.8, que mostra uma possível derivação da sentença “o garoto jogava bola”. Um traço está disponível para o processo de unificação em todo o caminho que leva do nodo onde foi definido até a raiz da árvore de derivação ou até que se encontre uma restrição que especifique o contrário.

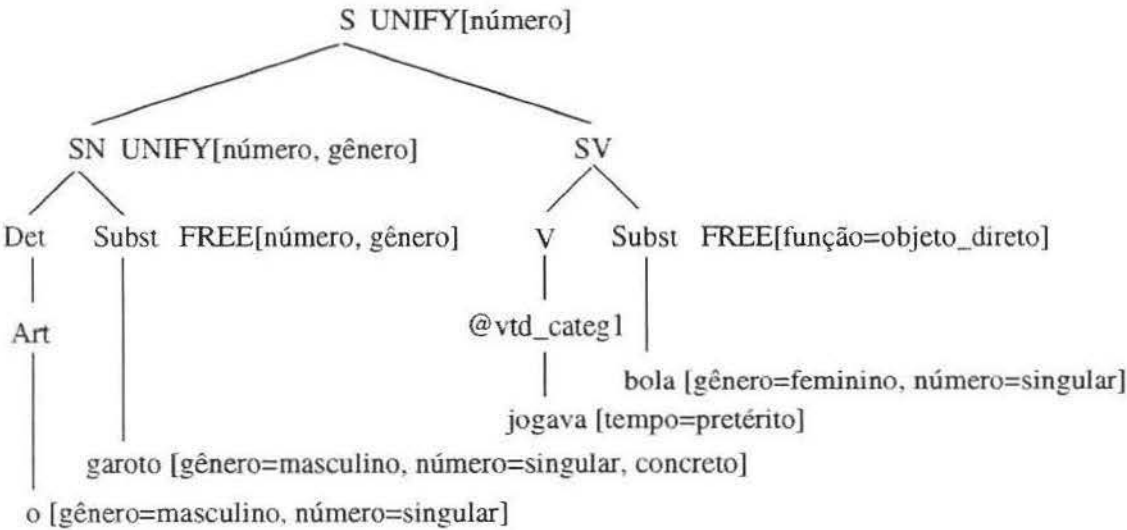


Figura 4.8: Um exemplo de codificação por traços.

Todas as informações associadas ao artigo “o”, por exemplo, estão disponíveis por propagação por todo o caminho até o nodo “S”, sendo efetivamente utilizadas no nodo “SN”. Neste nodo os valores dos traços “número” e “gênero” devem ser comparados com os valores dos mesmos traços associados à palavra “garoto” que propagam-se para o nodo

“subst”. A restrição “FREE” contida no nodo “SN” impede que qualquer outro traço que não seja “número” ou “gênero” seja referenciado pelos nodos que o dominam. Este é o caso do traço “concreto”, associado ao substantivo “garoto”. Outro exemplo de restrição pode ser visto no segundo substantivo da sentença, dominado pelo sintagma verbal “SV”, que bloqueia a propagação de todas as informações, com exceção do traço “função”, ao qual é associado o valor “objeto_direto”. Note que as restrições do tipo “FREE” e “BLOCK” são funcionalmente equivalentes. Foram definidos dois tipos de restrição pensando no conforto de quem define a gramática.

Ainda no exemplo da figura 4.8, a raiz da árvore indica a necessidade de uma última verificação entre o valor do traço “número” associado ao sintagma verbal “SV” e o valor associado ao sintagma nominal. Esta informação, no entanto, não está disponível para o sintagma verbal “SV” porque não pode ser propagada por nenhum nodo que domine. O nodo “subst” bloqueia este traço e o verbo acomodado na âncora não o possui.

A codificação dos traços em constantes e variáveis é apenas sugerida, na medida em que o modelo proposto não implementa a unificação. Ela, no entanto, é capaz de manipular não apenas traços morfo-sintáticos, mas também predicados mais complexos. Pode-se elaborar listas de identificadores que especifiquem um determinado predicado para o analisador externo ou mesmo, por exemplo, um identificador no léxico-semântico de um outro agente [34]. O mecanismo de bloqueio e propagação de traços também é apenas sugerido. Ele pode ser substituído por uma linguagem de automação (*macros*) caso necessário.

4.3.5 O operador “...” como um inversor de sentido

Ao propor o uso de ilhas, as BMGs generalizam o operador “...” definido pelas XGs. Os motivos que levaram a esta generalização já foram discutidos anteriormente, assim como os efeitos desse operador em relação à complexidade computacional do formalismo. Sua utilidade, no entanto, torna-o interessante para o processamento de elementos desconhecidos na sentença de entrada.

Para aproveitar sua flexibilidade sem comprometer o custo do modelo, ele foi adaptado e simplificado. Este operador pode ocupar o espaço de um nodo folha nas árvores da gramática, sejam elas iniciais ou auxiliares. Só é permitido um nodo rotulado “...” por árvore, seja ela da gramática ou de derivação. O analisador garante que nenhuma operação efetuada durante a derivação resultará em uma árvore derivada com mais de um nodo rotulado “...”, preservando a ocorrência mais antiga. A figura 4.9 mostra dois exemplos de como este operador é removido.

Ao garantirmos sua unicidade podemos modificar seu significado para indicar que na posição deste operador devemos acomodar um ou mais nodos desconhecidos. Caso se queira obter o efeito de “zero ou mais nodos desconhecidos” basta marcar o nodo do

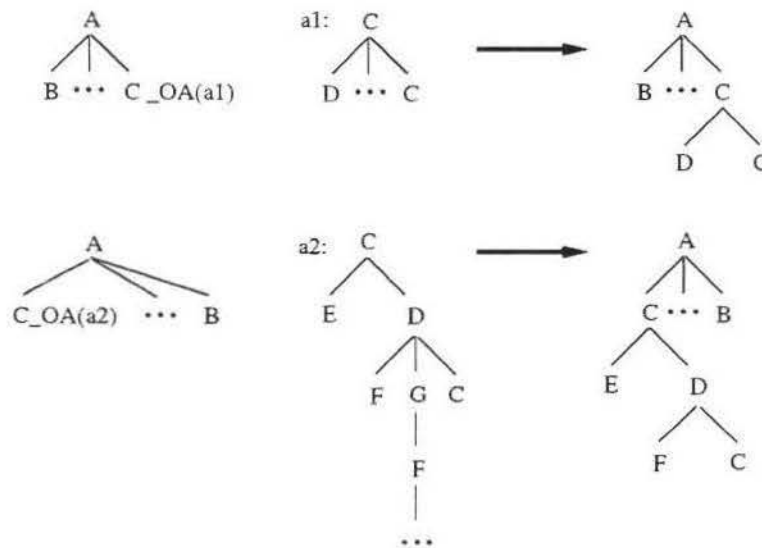


Figura 4.9: A remoção da ocorrência mais recente do operador "...".

operador "...", como não obrigatório ("NO"). Veremos mais adiante que esta modificação é suficiente para evitar a NP-completude.

4.3.6 As agendas dos segmentos do analisador e alguns exemplos simples de derivação

Tomando a gramática da figura 4.3 como exemplo, mostraremos como o sistema deriva algumas sentenças com grau de complexidade crescente. Antes disso, no entanto, é preciso entender como o agente organiza seu conhecimento internamente.

Cada segmento do analisador possui uma agenda de entrada utilizada para armazenar as solicitações até que existam recursos para seu processamento. Cada uma destas agendas é preenchida e analisada independentemente através de processos assíncronos (*threads*).

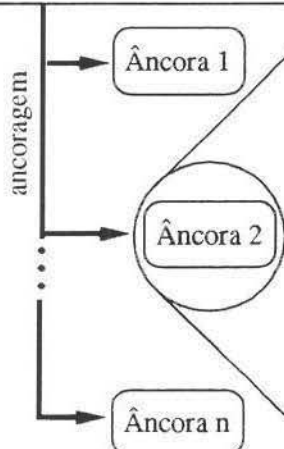
A primeira agenda, associada ao primeiro segmento do analisador, guarda as sentenças de entrada e as informações associadas às palavras desta sentença. Ela é preenchida quando o SMA solicita o processamento de uma sentença e esvaziada quando o analisador inicia o processo de ancoragem, ou seja, de seleção das árvores ancoradas pela sentença de entrada.

A segunda agenda, associada ao segundo estágio, é preenchida com uma ou mais solicitações geradas para cada sentença de entrada de acordo com as possíveis ancoragens. Ao ancorar a sentença na árvore inicial o processo segmenta a sentença de entrada em duas partes, uma que vai do início da sentença até uma palavra antes da âncora e outra que vai de uma palavra depois da âncora até o final da sentença. A palavra que selecionou a estrutura é imediatamente acomodada na âncora, o que transforma a árvore inicial em

uma árvore de derivação. Neste ponto ainda não foram analisados nem os seletores de adjunção nem os marcadores de nodo opcional e, por isso, estas solicitações são denominadas solicitações complexas. Esta agenda é esvaziada no terceiro estágio do processo. Um exemplo de transição do formato da agenda do primeiro estágio para o da agenda do segundo pode ser vista na figura 4.10.

Estágio 1

O	art	[masc, sing]
garoto	subst	[masc, sing]
comeu	verbo	[transitivo_direto]
doce	subst	[masc, sing]
no	prep	[masc, sing]
quarto	subst	[masc, sing]



Estágio 2

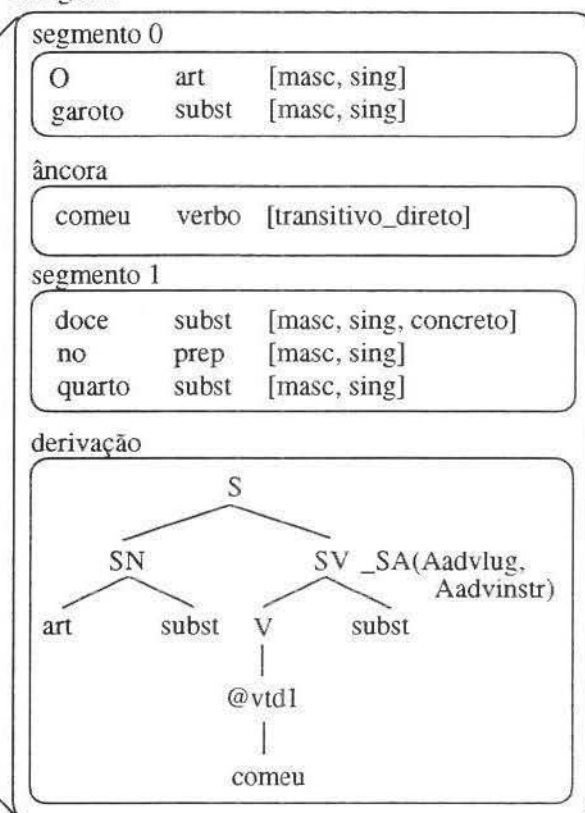


Figura 4.10: Conteúdo das agendas do primeiro e segundo segmentos do analisador.

A terceira agenda, associada ao terceiro estágio, contém uma ou mais solicitações geradas a partir de cada solicitação complexa. Estas solicitações são construídas com a aplicação de todas as combinações possíveis de adjunção, com e sem os nodos opcionais. Algumas restrições diminuem a quantidade de combinações possíveis como, por exemplo, a contagem de nodos. Operações que gerem uma árvore derivada com mais nodos folhas que o número de palavras da sentença de entrada são descartadas. Neste ponto as árvores de derivação não possuem mais nodos opcionais nem seletores de adjunção e as solicitações passam a ser denominadas solicitações simples. A figura 4.11 mostra como um seletor de adjunção pode ser removido ao efetuarmos todas as adjunções especificadas. Cada

entrada da agenda do terceiro estágio é composta por uma alternativa de derivação mais os segmentos de texto e a âncora. Esta agenda é esvaziada no quarto estágio, que tenta acomodar os segmentos da sentença de entrada nos nodos folha da árvore de derivação.

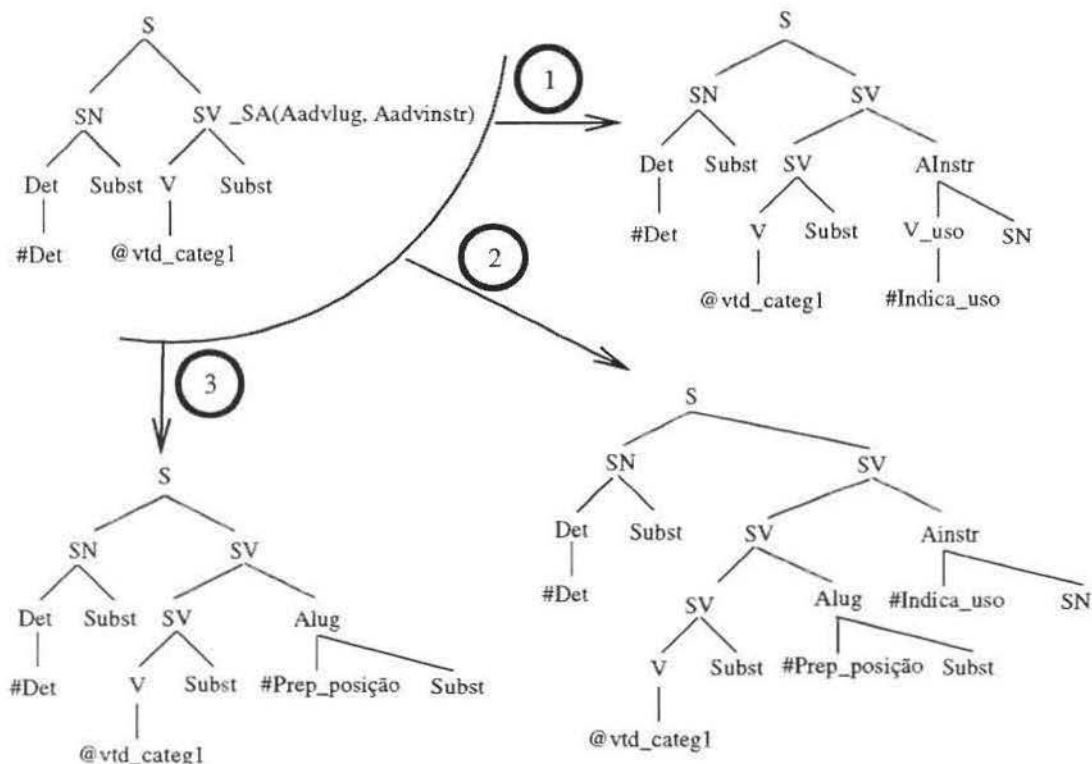


Figura 4.11: A expansão de um seletor de adjunção “_SA”.

Caso o processo de acomodação tenha sucesso, o sistema verifica se existe alguma ilha na árvore de derivação e disponibiliza uma solicitação de unificação ou a solução, conforme o caso. Caso o processo fracasse, são analisadas as possibilidades de substituição para cada nodo folha, são executadas as alternativas adequadas e reagendadas as novas alternativas como solicitações complexas na agenda do segundo estágio.

Apesar de termos representado as árvores de derivação explicitamente, as agendas utilizam o formato linearizado. Uma entrada de agenda, seja qual for, também contém uma cópia da sentença de entrada, o que evita a necessidade do texto ser tratado como uma variável global, o que não seria conveniente para a comunicação entre os agentes. Caso a gramática utilizada pelo sistema seja comum a todos os agentes, uma solicitação pode ser considerada atômica e propagada sem necessidade de se armazenar nenhuma informação adicional.

Ainda utilizando a gramática apresentada na figura 4.3, vamos analisar a sentença exemplo “o menino comeu doce”. O único item léxico que ancora uma estrutura aqui é

“comeu”. Ele seleciona a estrutura <vtd1> através da âncora @vtd_categ1. Com apenas uma estrutura selecionada, apenas uma solicitação complexa é gerada. Esta solicitação complexa gera apenas uma solicitação simples porque ambas as adjunções previstas no seletor “_SA” são descartadas por causa do excesso de folhas em relação ao número de palavras disponíveis.

O agente responsável pelo consumo da sentença tenta acomodar, da esquerda para a direita, as palavras na árvore derivada, o que ocorre com sucesso neste exemplo. Aqui se nota porque uma gramática eficiente neste formalismo deve ter a menor quantidade possível de operações definidas. Também se pode observar a importância das árvores iniciais serem realmente completas.

Um exemplo mais interessante de análise pode ser obtida com a sentença “o menino comeu doce no quarto”. O maior número de palavras na sentença de entrada permite que as variações 1 e 3 da figura 4.11 sejam agendadas, além da variação sem nenhuma adjunção. Novamente basta acomodar o texto na alternativa 3 para encontrar a solução. Neste exemplo a alternativa 2 é descartada por ter nodos-folha demais.

O interessante aqui é observar as alternativas geradas pelo sistema. Em um exemplo maior, como no caso da sentença “o menino comeu doce no quarto com as mãos”⁵, todas as alternativas da figura 4.11 seriam geradas. A alternativa 1 é gerada com a adjunção da árvore “Aadvinstr” da figura 4.3 no nodo adequado. A alternativa 2 é gerada com a adjunção da árvore “Aadvlug” e depois da árvore “Aadvinstr” em sequência. A alternativa 3 é gerada pela adjunção da árvore “Aadvlug” apenas. Estas combinações são definidas pela ordem em que os identificadores das árvores aparecem no seletor de adjunção. Note que a combinação em que primeiro se adjuntaria “Aadvinstr” e depois “Aadvlug” não ocorre.

A figura 4.12 mostra esquematicamente como as solicitações simples são montadas a partir de uma solicitação complexa com seletores de adjunção e nodos opcionais.

A presença do operador “...” não gera grandes complicações, uma vez que todas as derivações possíveis já estão agendadas. Aproveitando o fato deste operador ser único em cada árvore de derivação (por definição), o analisador simplesmente inverte a ordem de consumo das palavras da sentença de entrada. Ele inicia da esquerda para a direita até encontrar o nodo com o operador “...”, inverte a direção e passa a consumir da direita para a esquerda a partir da última palavra da sentença até encontrar o operador novamente. As palavras da sentença de entrada que sobrarem ao final deste processo são acomodadas sob o nodo do operador.

⁵Se a frase fosse “o menino comeu doce no quarto com o cachorro” o analisador entenderia que ele usou o cachorro como instrumento para comer, o que deixa claro que esta gramática é apenas um exemplo. A construção de uma gramática mais inteligente faz parte da lista de trabalhos futuros.

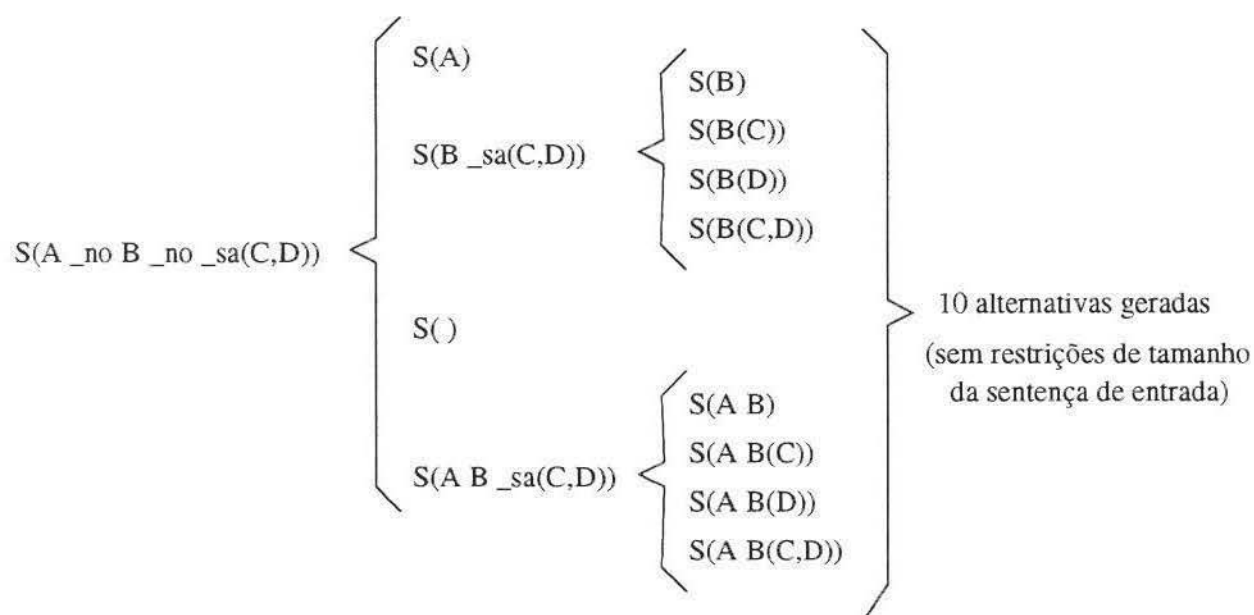


Figura 4.12: A expansão de uma solicitação complexa.

4.3.7 Sumarizando o formalismo lingüístico

O formalismo apresentado pode ser considerado como uma simplificação das TAGs com algumas características adicionais. As modificações, no entanto, abrangem desde a definição das árvores elementares, passando pelas operações permitidas e indo até a maneira como o processo de unificação deve ocorrer. Além disso são introduzidas características das BMGs e das XGs, o que nos leva a considerar até que ponto este modelo é uma simplificação ou um novo formalismo híbrido.

São utilizadas informações ascendentes⁶ para iniciar o processamento, na medida em que é a lexicalização que define as estruturas iniciais, mas praticamente toda a derivação tem lógica *top-down*. Esta lógica *top-down*, associada à retirada dos predicados de contexto, fez com que a operação de adjunção fosse modificada. Esta modificação também levou em consideração o novo papel lingüístico das árvores auxiliares e a diminuição das operações decididas implicitamente pelo sistema.

Algumas facilidades adicionais visam a manutenção das gramáticas e sua generalidade, tais como as listas de palavras, que permitem trabalhar diretamente com terminais, ao invés de pré-terminais; as listas de árvores, que servem como uma maneira concisa de citar uma lista de identificadores; o operador "...", que permite que estruturas esperadas, mas com sintaxe pouco previsível, como por exemplo os apostos, possam ser tratadas; as informações codificadas nos nodos das árvores elementares e nas palavras da sentença de

⁶Do inglês *bottom-up*.

entrada; e os pontos de verificação, na forma de ilhas simplificadas, que permitem que o autor da gramática defina a maneira como as informações coletadas durante a análise sintática são disponibilizadas.

O formalismo tem a capacidade de manter referências encaixadas e cruzadas sem as limitações das TAGs. Isto ocorre porque estas referências não são mais mantidas através de ligações, mas das anotações, como nos mostra a figura 4.13. Por outro lado, como não existe mais a recursão na adjunção, a quantidade de referências fica limitada pela gramática de entrada.

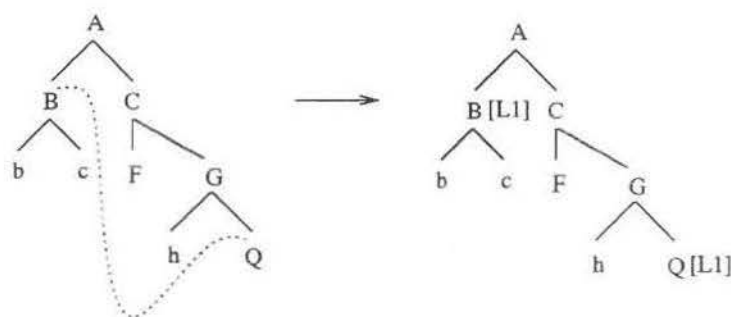


Figura 4.13: Uma ligação explícita e seu equivalente anotado.

O processo de anotação exige, no entanto, cuidado na formulação dos rótulos. Caso existam coincidências de nomes as referências podem ficar confusas, exigindo cuidado na sua interpretação. Em especial, quando duas instâncias de uma mesma árvore forem inseridas durante a derivação por uma substituição ou adjunção, as referências ficarão duplicadas. Não se espera que isto ocorra com frequência se as árvores auxiliares forem utilizadas como uma forma de representar componentes opcionais da estrutura frasal.

Apesar do formalismo ser derivado de três outros formalismos, a combinação obtida pode endereçar vários dos problemas lingüísticos abordados, completa ou parcialmente. Uma sentença complexa como “O livro que a uma amiga o João deu desapareceu”, usada anteriormente para demonstrar como as BMGs superam as XGs, não pode ser tratada diretamente porque as pilhas de referentes lingüísticos das ilhas foram retiradas.

Por outro lado, o analisador consegue identificar e selecionar as partes que compõem a sentença, facilitando a comunicação com qualquer módulo externo, que possa tratar estas referências. A figura 4.14 mostra como podemos separar as partes da sentença para construir solicitações mais simples.

A tarefa de decompor as chamadas e corrigir as referências necessárias para o tratamento dos fenômenos lingüísticos mais complexos depende da organização do restante do SMA. O analisador proposto não tem os recursos necessários nem pretende resolvê-los sozinho. Seu mérito é prover um conjunto de primitivas de análise sintática que podem ser utilizadas dentro de uma sociedade de agentes.

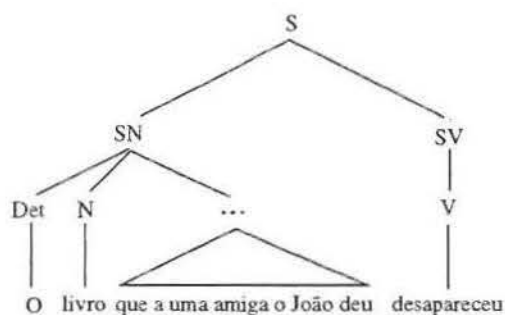


Figura 4.14: Separando constituintes de uma derivação complexa.

Para que possamos entender como isto é possível, é preciso descrever a parte da sociedade de agentes que trata a sintaxe, ou seja, a subsociedade sintática.

4.4 A definição da sociedade de agentes

O formalismo lingüístico que acabamos de descrever baseia-se não apenas em uma simplificação dos formalismos lingüísticos estudados. Muitas de suas características derivam das capacidades de comunicação e de cooperação de cada agente do sistema definidas de acordo com o modelo geral.

Uma das principais características dos agentes da subsociedade que deve executar a análise sintática é a capacidade de modificar o curso de sua derivação normal conforme novos dados são fornecidos pelo SMA. Podemos dizer que seu processo de derivação é sensível ao meio. Outra é permitir o incremento iterativo da gramática. Os demais agentes do SMA devem poder sugerir novas descrições gramaticais ou mesmo remover as existentes. A terceira característica principal é permitir que se audite cada etapa do processamento. Deve-se poder parar o processamento e verificar o estágio atual da análise, todas as descrições parciais, a gramática que está em uso, quais recursos desta gramática foram selecionados, quais foram descartados e o porquê desta escolha. Em outras palavras, o agente deve “lembrar” do que fez.

O agente também deve, obviamente, ser capaz de analisar sintaticamente uma sentença. Mais que isso, deve ser capaz de processar várias sentenças em paralelo usando a mesma base gramatical. Seu algoritmo de escalonamento deve ser capaz de aproveitar os recursos de uma máquina multi-processada assim como também a disponibilidade de várias máquinas em uma rede de computadores. Um esquema de como os agentes operam como uma subsociedade pode ser visto na figura 4.15. Nesta figura podemos perceber que os agentes podem se comunicar diretamente para troca de solicitações sem a colaboração do resto do SMA, o que caracteriza o comportamento de subsociedade. Esta comunicação pode ser feita entre agentes em uma mesma máquina ou sobre uma rede local de

computadores.

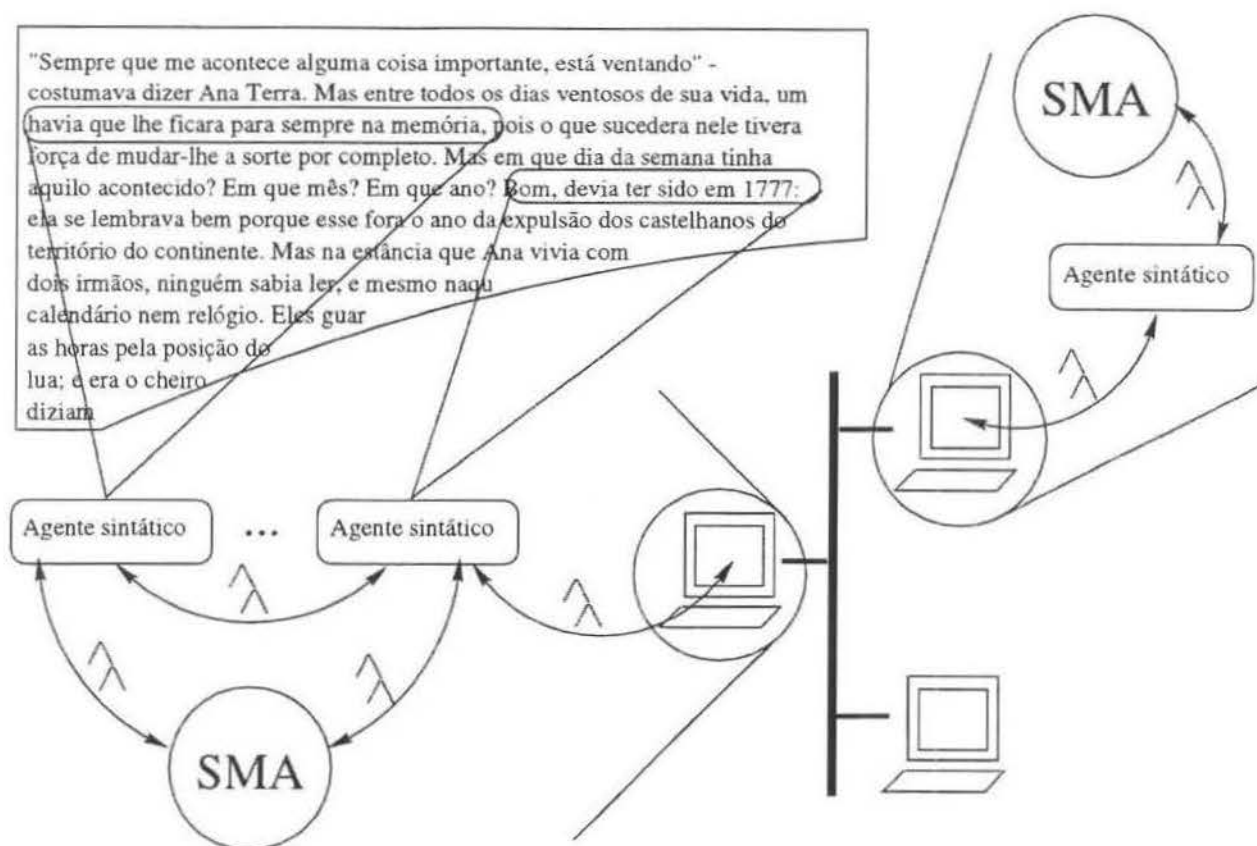


Figura 4.15: Ilustrando o funcionamento da sociedade de agentes.

Ao compararmos esta descrição informal com a técnica de descrição formal apresentada no final do capítulo 3, podemos considerar o conjunto G , da tripla

$$P(S, G, F) \rightarrow (A, E, I, O)$$

preenchido. Este mesmo método será usado para definir cada um dos conjuntos restantes e descrever a sociedade de agentes.

O conjunto G , que acabamos de descrever, define quais os objetivos da sociedade. O conjunto S também já é conhecido. Ele se refere aos recursos e restrições disponíveis para o processamento e equivale à nossa gramática e aos dados que acompanham a sentença de entrada, ou seja, à parte lingüística do modelo proposto. F é o conjunto de agentes que serão utilizados no processamento. Neste caso usaremos um agente sintático que pode ser replicado e distribuído em uma rede de computadores formando uma subsociedade.

A quádrupla seguinte, (A, E, I, O) , se refere às capacidades de cada agente do sistema. A subsociedade sintática é composta por um ou mais agentes homogêneos que "repartem"

o problema da análise sintática de forma predominantemente extrínseca, ou seja, repartem os dados de entrada e funcionam colaborativa porém independentemente. Cada agente pode funcionar como um elemento isolado caso necessário⁷.

Prosseguindo na definição temos o conjunto E , que se refere ao ambiente onde o agente está inserido. O agente foi desenvolvido em ANSI C++ prevendo seu encapsulamento em um objeto CORBA⁸. Objetos deste tipo podem ser replicados, distribuídos e se comunicar sobre uma interface de rede local, sendo também compatíveis com um grande conjunto de linguagens de programação modernas. Em outras palavras, quanto ao ambiente computacional, o agente pode ser incorporado em praticamente qualquer plataforma moderna⁹. Este conjunto, no entanto, não se refere apenas ao sistema operacional e à linguagem de programação. Ele se refere especialmente aos demais agentes do SMA que, no nosso caso, são réplicas do agente que está sendo descrito.

O conjunto O se refere à organização entre os agentes. O modelo proposto não determina uma hierarquia de agentes, apenas uma organização interna. As mensagens recebidas do SMA são consideradas sempre confiáveis, os objetivos dos agentes são homogêneos e os agentes colaboram sempre que existirem recursos disponíveis. A comunicação é direta, assíncrona e bidirecional.

Faltam ainda definir os conjuntos A e I . O conjunto A se refere às capacidades individuais dos agentes e o conjunto I ao modo como os agentes interagem durante o processamento. O conjunto I , na verdade, foi o primeiro a ser definido quando apresentamos o modelo de agente. Quando apresentamos os mecanismos de implementação dos formalismos lingüísticos, o fizemos em separado para frisar sua independência. Ao definirmos a interação desejada antes do nosso formalismo, no entanto, geramos novos requisitos que devem ser cumpridos. Esses requisitos, que veremos agora, levaram à organização do processo de unificação, à simplificação do conceito de ilha e às modificações nas operações de adjunção e substituição.

O conjunto A possui os subconjuntos $\{A_{cap}, A_{conf}, A_{disp}, A_{carga}\}$. O subconjunto A_{disp} varia no tempo e indica se o agente pode ou não responder às solicitações do SMA. No nosso caso ele se baseia na carga do sistema e na disponibilidade de memória. O conjunto A_{carga} é o responsável por indicar se o sistema comporta ou não novas solicitações de serviço. Ele foi definido para medir a quantidade de memória disponível para alocação

⁷O desempenho do sistema pode ser prejudicado neste caso porque não será possível balancear a carga do processamento nem distribuí-lo sobre uma rede local de computadores.

⁸Mais informações podem ser obtidas em <http://www.omg.org/corba/beginners.html>, <http://orbitcpp.sourceforge.net> ou <http://www.gnu.org>.

⁹Através de um encapsulamento CORBA pode ser inserido e/ou se comunicar, até o momento, nos ambientes VMS, AIX, Solaris, linux, WindowsXX, Mac, Irix e SmallTalk. Pode se comunicar com módulos desenvolvidos em C, C++, ADA, Pascal, TCL/TK, Perl, REXX, Lisp, Prolog, etc. Novas plataformas são adicionadas com frequência; uma lista atualizada pode ser obtida em <http://www.freshmeat.net>, assim como várias implementações gratuitas.

no sistema. Quando a memória disponível é zero, a carga chega ao máximo (100%). A carga mínima (0%) é definida pela memória disponível no início do programa.

Esta definição não é suficiente para evitar que os sistemas se sobrecarreguem, na medida em que a disponibilidade de memória nem sempre é o fator decisivo para o desempenho de um computador¹⁰. O conjunto A_{conf} indica a confiabilidade de um agente em resolver um problema. No modelo proposto, com decomposição extrínseca, este índice é sempre máximo (ou 100%). Caso o SMA descreva o tipo de problema lingüístico a ser tratado esta função deve passar a contar com uma tabela de fenômenos que podem ser analisados completa ou parcialmente.

Os conjuntos A_{cap} e G_{sub} se confundem na subsociedade sintática, uma vez que seus agentes são homogêneos. Cada agente possui um conjunto de capacidades que, quando analisadas separadamente, são simples. Ele pode receber uma solicitação de análise que contém uma sentença de entrada com as informações morfo-sintáticas associadas. Cada agente é capaz de analisar a sentença de entrada e selecionar as árvores iniciais que devem ser utilizadas, ou seja, é capaz de analisar as âncoras da gramática.

Para cada árvore inicial selecionada, o agente monta uma solicitação inicial de processamento. Esta solicitação contém todas as informações necessárias para que a análise prossiga, exceto a gramática, que é um argumento implícito. A solicitação é enfileirada em uma agenda de processamento e posteriormente enviada ao SMA ou processada pelo próprio agente, de acordo com sua carga, conforme descrito anteriormente. Quando uma solicitação é enviada ao SMA, sua transmissão é por difusão. Quando um agente recebe este tipo de mensagem, responde informando sua carga. O agente solicitante decide então quem será responsável pelo processamento numa comparação direta. Em caso de empate ocorre sorteio.

Em termos do conjunto de capacidades, isto quer dizer que cada agente é capaz de solicitar uma análise e de responder às solicitações do sistema. Também é capaz de analisar sua carga no momento da solicitação e de responder de acordo. Um agente também é capaz de “lembrar” das solicitações que fez e de escolher quem a executará entre os que responderam. Caso não existam “voluntários”, o agente é capaz de agendar a solicitação para que ele próprio resolva ou para solicitar novamente mais tarde.

O agente também pode ser configurado para não enviar solicitações ao sistema e funcionar isoladamente. Isto permite que ele trabalhe com uma gramática diferente da dos demais. Os agentes não são capazes de identificar qual agente usa que gramática, ou seja, não identificam conflitos. A resolução deste problema fica a cargo da configuração de cada agente fornecida pelo gerente do sistema, seja ele humano ou não. A figura 4.16 mostra

¹⁰Um computador com memória infinita e um processador típico sucumbiria diante da proliferação de programas do tipo “enquanto(verdadeiro) faça 1+1”. O processador estaria completamente ocupado mesmo que houvesse muita memória disponível. Um verificador de carga menos simplório faz parte da lista de trabalhos futuros.

como os agentes trocam ou não mensagens de acordo com a configuração inicial de suas gramáticas.

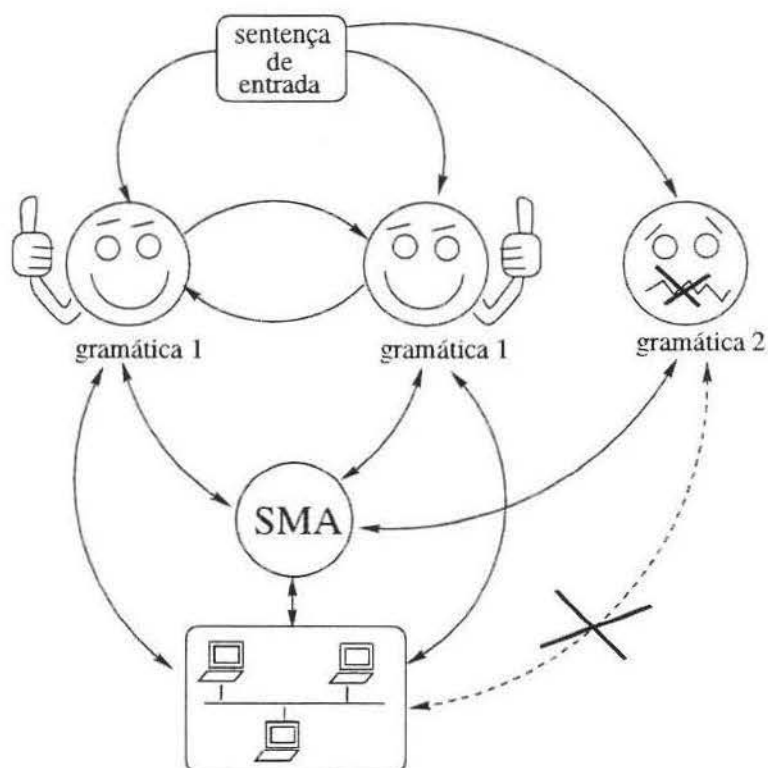


Figura 4.16: Gramáticas diferentes impedem a comunicação direta entre agentes.

Além da habilidade de se comunicar e de solicitar ajuda ao SMA, cada agente é capaz de efetuar as operações de substituição e de adjunção, de manipular os seletores de adjunção e o operador "...", ou seja, de analisar a sentença de acordo com o formalismo lingüístico proposto.

Cada agente também é capaz de identificar quando uma determinada derivação que chegou ao fim necessita de unificação. Basta verificar se existem ilhas na árvore derivada e, em caso afirmativo, solicitar a unificação por difusão para o SMA. Para montar esta solicitação o agente é capaz de selecionar a informação relacionada com a derivação e disponibilizá-la coerentemente. Caso não exista a necessidade de unificação, a derivação é considerada uma resposta e é disponibilizada também por difusão.

Cada agente também tem uma capacidade limitada de aprendizado. Ele é capaz de receber novas descrições gramaticais (árvores, ilhas, âncoras e listas de palavras e árvores), de verificar se estas descrições são coerentes com a gramática que carrega e de incorporá-las à gramática. Estas descrições, no entanto, sempre procedem de fora da subsociedade sintática, uma vez que seus agentes não são capazes de construí-las. Elas podem vir,

por exemplo, de um usuário humano através de um agente interface ou, também, de um agente que tenha um mecanismo de aprendizado mais complexo. As novas descrições, no entanto, não são armazenadas na gramática original, ou seja, não se tornam permanentes. Um agente sintático é capaz de descartar uma gramática completamente e de carregar uma diferente ou recarregar a atual, eliminando as alterações.

Através da comunicação de novas estruturas é que os agentes podem receber solicitações para efetuar uma adjunção específica. Caso queiramos forçar esta operação, basta criar uma nova árvore com um seletor “_OA(o_que_quero_adjuntar)” ao lado do nodo onde se quer que a adjunção ocorra.

Cada agente é capaz de identificar um outro agente a quem deve comunicar suas ações. Este outro agente pode ser, por exemplo, uma interface com o usuário. Apesar de notificado, este outro agente não é hierarquicamente superior ao agente da subsociedade sintática.

Também é possível fornecer ao agente uma árvore parcialmente derivada e solicitar que o agente termine de derivá-la. É exatamente isso que os agentes fazem quando montam as solicitações de processamento e as enviam ao SMA. É necessário, no entanto, que o agente que envia esta solicitação conheça a notação utilizada internamente pelos agentes da subsociedade sintática. Segundo o modelo proposto o SMA tem pleno acesso às agendas de todos os segmentos do analisador, o que obriga a implementação de mecanismos de controle de acesso¹¹. Os agentes consideram todas as solicitações de análise e todas as descrições lingüísticas recebidas como válidas e de igual importância. Em outras palavras, são incapazes de especificar uma prioridade na análise ou de identificar um agente “mal-intencionado”.

Em relação ao conjunto I , que codifica as comunicações, podemos dizer que para cada capacidade do agente existe uma primitiva de comunicação. Como as solicitações são atômicas, ou seja, contêm todas as informações necessárias para sua realização, não é necessário implementar um protocolo de troca de dados complexo, exceto para decidir qual agente deve processar uma solicitação quando mais de um responde a um pedido.

Toda capacidade de um agente é implementada por um serviço confirmado, ou seja, para cada solicitação existe uma resposta de recebimento. Existe ainda um limite de tempo para o recebimento da resposta que, caso não seja respeitado, equivale a uma falha.

¹¹Em C++ isto significa a implementação de estruturas de dados seguras para ambiente distribuído, ou seja, estruturas *thread safe*.

4.5 Problemas conhecidos

Um dos problemas encontrados com o modelo de multi-agentes está relacionado com o cálculo da complexidade computacional do sistema. Apesar das alterações, o modelo proposto ainda possui grande semelhança com as TAGs, o que indicaria um caminho para a construção da análise.

Os elementos adicionais, como o operador "...", podem ser reduzidos a operações similares. Quando dissociado da pilha de referentes lingüísticos, dentro do modelo de processamento baseado em agenda, e sendo único em cada árvore de derivação, este operador não altera a complexidade do sistema. Os seletores de adjunção também não parecem aumentar o custo do processamento em relação às FB-TAGs.

O problema, no entanto, surge quando consideramos que um agente só existe dentro de uma sociedade e que o comportamento deste agente é afetado por esta sociedade. Como calcular o custo computacional de um elemento desconhecido? Um agente definido de acordo com o modelo proposto pode ser dirigido indefinidamente por um usuário. Também pode ser dirigido por outros agentes através da edição das agendas, o que gera a possibilidade de que a interação de protocolos de comunicação não permita uma derivação adequada.

Mesmo se trocarmos a análise do custo pela análise de convergência do protocolo de comunicação, não podemos descartar a possibilidade do sistema se comportar de maneira errática, uma vez que não se conhece o comportamento dos agentes do SMA que não pertencem à subsociedade sintática.

Também existem problemas relacionados com a convergência do modelo gramatical. A definição lingüística dada permite que o número de nodos de uma árvore de derivação permaneça constante durante uma derivação. Isto quer dizer que, como só se descarta uma derivação quando não se pode mais adjuntar nem substituir por excesso de nodos ou por falta de alternativas, o processamento pode não convergir. A gramática

$$\begin{aligned} <1> A \rightarrow B(@\text{ancora}) \\ <2> B \rightarrow C \\ <3> C \rightarrow B \end{aligned}$$

onde "@ancora={pal2}:{1}", levaria à recursão eterna quando a sentença de entrada for "pal1 D [], pal2 E []". A palavra "pal2" da sentença de entrada seria ancorada em "@ancora", mas a palavra "pal1", de categoria "D", não poderia ser consumida porque se espera uma palavra da categoria "B". Como não existem adjunções possíveis, *B* seria substituído por *C*. Como não existe um aumento no número de nodos, este por sua vez seria substituído por *B* e assim por diante.

Este problema é fácil de resolver na teoria, mas capcioso na prática. Bastaria definir que toda árvore elementar (inicial ou auxiliar) deve consumir ao menos um nodo da sentença de entrada através de uma lista de palavras, ou seja, definir que todas as operações passam a ser lexicalizadas. Esta foi a solução encontrada pelas L-TAGs, mas não pode ser considerada ideal porque torna as gramáticas complicadas, difíceis de construir, usar e pouco intuitivas.

Uma alternativa mais simples é não permitir a definição de árvores elementares com apenas um nodo folha, o que faria com que qualquer operação de adjunção ou substituição aumentasse o número de nodos folha da árvore de derivação. Neste caso, em algum momento da derivação o número de nodos da árvore derivada superaria o de nodos folhas e o processamento se encerraria.

Além dessas alternativas, ainda é possível fatorar automaticamente a gramática de entrada para que ela obedeça a qualquer uma destas restrições. Este tipo de fatoração é necessária, por exemplo, para analisadores ascendentes recursivos (ASDRs) que tratam gramáticas livres de contexto. Este tipo de analisador não suporta recursão à esquerda nas produções.

Optou-se, no entanto, por deixar esta questão em aberto, sob responsabilidade de quem escreve a gramática. Na lista de tarefas futuras está a construção de um verificador de dependências circulares.

Capítulo 5

Conclusões e a continuidade do trabalho

"Beware of bugs in the above code;
I have only proved it correct, not tried it."

Donald Knuth

"A successful [software] tool is one that was used
to do something undreamed of by its author."

S. C. Johnson

O processamento da língua natural é intrinsecamente multi-disciplinar. Para compreendermos como juntar a eficiência computacional com a adequação lingüística foi necessário percorrer um longo caminho que passou por diversos campos do conhecimento. O processo de aprendizado, do qual fez parte a elaboração desta dissertação, nos levou a reformular grande parte do que considerávamos "conhecimento sólido". Para tentar dar alguma ordem às conclusões a que chegamos neste trabalho, vamos seguir a ordem dos temas como foram apresentados.

O desenvolvimento de sistemas para o PLN é uma tarefa multi-disciplinar que não pode ser levada a cabo de maneira solitária. Apesar de ser possível que um indivíduo desenvolva experiências que contribuam para a área, é preciso coordenar e planejar os trabalhos individuais visando um objetivo maior. Coordenar um grupo de pesquisa que envolve Ciência da Computação, Lingüística e, em alguns casos, Biologia e Medicina, não é tarefa simples. Essa dificuldade, somada à pluralidade de objetivos dos pesquisadores, talvez seja o maior empecilho para o avanço do PLN.

Uma das diferentes visões sobre o problema da análise automática de textos nos leva à abordagem sintaxista. Esta abordagem considera a sintaxe como central para a com-

preensão da língua e serve como base para os formalismos estudados. Das XGs, passando pelas BMGs, TAGs, pela Teoria das Barreiras, *Government and Binding*, até as soluções mais centradas na lexicalização radical, como o *Word Expert Parser* e o *Parallel Expert Parser*, tentar descobrir qual a estrutura sintática que sustenta a integridade da sentença é uma constante.

A dificuldade de se obter uma explicação definitiva sobre o problema da sintaxe levou à multiplicação dos formalismos. Os formalismos que surgiram, no entanto, também não são capazes de resolver o problema por completo e, para cada solução encontrada ou fenômeno lingüístico explicado, novos problemas surgem. Estes problemas por vezes se manifestam na incompatibilidade das soluções, o que dificulta a integração dos formalismos.

Uma técnica recorrente no conjunto de formalismos analisado é a lexicalização. Ela permite diminuir o espaço de busca da gramática e organizar o processamento porque permite prever quais constituintes devem ser encontrados, dado o conhecimento prévio sobre um determinado item léxico que faz parte do texto analisado. Alguns formalismos baseiam-se inteiramente na lexicalização enquanto outros a utilizam para minimizar o problema da complexidade computacional.

A complexidade computacional pode ser vista como a medida do tempo e/ou espaço necessários para se resolver um problema em ambiente computacional. O tempo necessário para analisar um determinado texto (ou uma simples sentença) pode ser substancial, o que nos revela que a preocupação com as definições formais é necessária. Os formalismos analisados exibem diferentes graus de preocupação com a complexidade em sua definição. Os formalismos variam dos que se baseiam em propriedades matemáticas até os que consideram estas propriedades secundárias.

A complexidade, no PLN, se relaciona com que tipo de descrição lingüística se deseja obter. Uma descrição mais elaborada permite que o sistema identifique com maior precisão quais sentenças pertencem a uma determinada língua. Aumentar a precisão significa diminuir a quantidade de sentenças válidas descartadas e diminuir a quantidade de sentenças incorretas consideradas corretas pelo sistema.

A capacidade de fornecer descrições mais elaboradas está relacionada com a sensibilidade ao contexto do formalismo utilizado para analisar a sintaxe. Os formalismos que se baseiam em gramáticas mostram isso diretamente. Os que se baseiam na lexicalização radical mostram isso indiretamente e é necessário estudar seus protocolos de análise, ou seja, estudar como constróem uma resposta ou reconhecem uma sentença.

O artigo que primeiro apresentou as TAGs[47] já perguntava em seu título “Quanta sensibilidade ao contexto é necessária para se obter descrições razoáveis?”. Não parece haver uma resposta consensual para esta pergunta. A definição das XG, por exemplo, não toca neste problema diretamente, mas este formalismo é comprovadamente sensível ao contexto e tem a mesma complexidade dos problemas NP-completos [40]. Muitos

outros formalismos seguem o mesmo caminho[53], o que nos levou a imaginar se qualquer sistema que trate de uma língua natural seria necessariamente NP-completo.

As TAGs possuem um custo computacional polinomial e são apenas parcialmente sensíveis ao contexto. Apesar deste formalismo não resolver todos os problemas lingüísticos, sua pesquisa ainda é um campo aberto. Sendo assim, não é possível afirmar que a língua natural é um problema NP-completo. As conclusões sobre complexidade devem ser tomadas caso a caso, formalismo a formalismo.

Tomar conhecimento dos problemas que um custo computacional elevado acarreta leva naturalmente à busca por alternativas. Uma análise dos sistemas de PLN existentes mostra que a simples mímica de parcelas da teoria lingüística, em especial o tratamento isolado da sintaxe, sem levar em consideração informações de outra ordem, gera ineficiência e ineficácia. Para muitos fenômenos, como as elipses e anáforas, é preciso coletar informações de vários níveis lingüísticos para sintetizar uma solução.

Quanto e quais são os níveis de análise necessários varia de acordo com os resultados esperados e o problema lingüístico tratado por cada sistema. A integração destes níveis, no entanto, sempre depende de uma política de comunicação e de cooperação. A maneira mais simples de se integrar os níveis de processamento é fazer com que os resultados de um deles sejam a entrada do seguinte, sem colaboração interativa. Esta falta de cooperação leva a ineficiência pela necessidade de múltiplas análises sobre um mesmo conjunto de informações.

Os sistemas de PLN são complicados pela própria natureza multidisciplinar e pelas características da língua natural. Esta complicação gera alguns problemas, entre eles a dificuldade de se obter a colaboração entre os níveis de análise, a dificuldade de se reaproveitar os trabalhos prontos em novos sistemas e a de se coordenar o desenvolvimento de sistemas ao longo do tempo e frente à mudança das pessoas envolvidas.

Ao longo do tempo as técnicas de desenvolvimento evoluíram da programação imperativa para os sistemas multi-agentes, passando pelos sistemas orientados a objetos e arcabouços. Essa evolução foi marcada pela retirada do foco dos sistemas dos processos de resolução dos problemas (programação imperativa), passando pelas entidades envolvidas no problema (orientação a objetos), pelos domínios de competência que envolvem a solução do problema (arcabouços) e chegando finalmente nos protocolos de comunicação (agentes).

Um agente isola não apenas seus dados e processos, mas também a interface de seus métodos internos. Ele possui a capacidade de interpretar um protocolo de comunicação, o que permite que se especifique uma interface simplificada e única para todas as solicitações. Desde que se adapte o protocolo de comunicação, um agente pode ser inserido em qualquer sistema novo, mesmo que exista sobreposição dos níveis de competência ou que o agente não preencha um destes níveis completamente, resolvendo os problemas de forma parcial.

O agente parece ser a unidade básica adequada de desenvolvimento nos sistemas em que a arquitetura global do problema não é conhecida ou em que as técnicas de paralelização tradicionais não sejam adequadas. A mesma interface que esconde os métodos internos de um agente protege a sociedade onde ele se insere de suas modificações internas. Ao alterar um agente não existem efeitos colaterais fora dele desde que a coerência do protocolo de comunicação seja mantida.

Apesar de ser natural definir um agente a partir de suas capacidades esta técnica nem sempre gera as melhores soluções. Ao definir o agente a partir das interações que ele deve ser capaz de gerenciar, ou seja, a partir da comunicação desejada na sociedade, podemos gerar soluções mais escalonáveis, no sentido da capacidade de balancear a distribuição de tarefas entre os agentes.

O modelo proposto de subsociedade sintática, que se utiliza de decomposição extrínseca, nos levou a reformular o conceito de “solução” para uma análise sintática. Com a interação vindo antes das capacidades individuais, uma solução passou a ser entendida como qualquer parcela de informação coerente que possa auxiliar na resolução de um problema maior. Mesmo a derivação por completo de uma sentença é parte da solução de um problema maior, o da análise do texto.

Este modelo busca preservar o domínio de localidade estendido das TAGs, assim como incorporar facilidades de outros formalismos (como as BMGs e XGs) para facilitar a construção, expansão e manutenção de gramáticas complexas.

As motivações lingüísticas do modelo, no entanto, foram “dirigidas” pelos requisitos computacionais, em especial a capacidade de balancear a carga de processamento sobre múltiplos processadores e/ou computadores em uma rede local. Esta restrição afeta o tipo de dados que o analisador pode manipular.

Para resolver o problema do escalonamento optou-se por definir uma estrutura auto-contida de comunicação. Ao transpor esta unidade de um agente para qualquer outro não é mais necessária nenhuma comunicação adicional. Este objetivo foi apenas parcialmente obtido na medida em que a base gramatical pode ser grande demais para que valha a pena encapsulá-la dentro deste objeto.

Para facilitar a organização do algoritmo de escalonamento propõe-se um modelo baseado em agendas (ou filas) de solicitações. Estas agendas podem ser acessadas concorrentemente e permitem que a parcela do agente que cuida do escalonamento e da distribuição de carga seja independente do núcleo do analisador sintático.

Por outro lado, o uso de agendas pode aumentar a quantidade de informação armazenada durante o processamento, na medida em que para cada solicitação inicial de carga podem ser geradas muitas alternativas de processamento. O crescimento do espaço, no entanto, pode ser controlado da mesma maneira que o processo é escalonado. Uma das agendas de entrada funciona como uma caixa postal de solicitações que pode ser fechada

ou não analisada quando a carga do sistema superar o recomendável.

O sistema baseado no modelo proposto não é um trabalho completo. Entre os trabalhos futuros estão o refinamento do controle da gramática para eliminar dependências circulares, a implementação de um agente responsável pela unificação, a integração dos agentes com outros sistemas e, principalmente, a construção de uma gramática abrangente para o português. Outro ponto interessante é o aprimoramento do mecanismo de unificação, que deve passar do simples bloqueio de traços para uma linguagem de *scripts* que permitam operações mais elaboradas.

O processo de integração envolve não apenas a utilização do modelo como um analisador sintático completo, mas também a análise de sua viabilidade como auxiliar no estudo de fenômenos lingüísticos específicos, tais como anáforas e elipses. Este estudo deve permitir que se identifique quais as alterações necessárias frente a novas necessidades. Ao analisarmos este tipo de integração poderemos validar a generalidade da proposta.

Outra possibilidade interessante é identificar como e se este modelo pode ser utilizado como uma ferramenta de análise do discurso, em especial para o preenchimento dos constituintes não realizados mapeados nos nodos não obrigatórios.

Uma das possibilidades de integração já iniciadas envolve a utilização do modelo em um indexador semântico de textos baseado no léxico-gerativo de Pustejovsky [70], [34].

Uma carência neste trabalho é a ausência de uma análise da complexidade computacional do sistema, que fica como sugestão para trabalhos futuros.

Capítulo 6

Apêndice A — Exemplos de derivação

Neste apêndice estão contidos alguns exemplos de derivação que pretendem ilustrar o funcionamento do protótipo construído. Apesar de não existir um estudo formal sobre a capacidade gerativa do formalismo híbrido desenvolvido, esperamos que os elementos aqui apresentados sirvam como um indicador de sua capacidade de reconhecimento.

Para representar as derivações parciais e os resultados encontrados utilizaremos a notação linearizada, conforme descrita no capítulo 4.

6.1 Exemplos triviais

Alguns exemplos de derivação apresentam um grau de complexidade baixo demais para demonstrar todos os recursos do protótipo, mas servem para ilustrar seu mecanismo de derivação.

Tomemos como exemplo a seguinte gramática¹:

iniciais

```
<icomer> s( sn( det _NO ( #det) subst)
            sv _SA(a_advlug, a_instr)( v( @vtd1)
                                     sn( det _NO ( #det) subst))) )
```

auxiliares

```
<a_advlug> sv( sv alug( prep( #prep) subst) )
<a_instr> sv( sv ainstr( v_uso( #indica_uso) sn( det subst)) )
```

âncoras

¹O formato apresentado é igual ao esperado pelo protótipo. Todas as gramáticas e sentenças de entrada utilizadas neste apêndice foram testadas e validadas.

```
@vtd1 = {comer, comeu}:{icomer}
palavras
#det = {o, os, a, as, um, uma, uns, umas, aquela, aquele}
#prep = {na, no, nas, nos, em}
#indica_uso = {usando, com}
```

Ela foi construída para lidar com algumas construções do verbo “comer”, como podemos ver pela âncora “@vtd1”. A lista de palavras “#det” agrega alguns termos que podem ser utilizados com a função de determinante. Obviamente esta lista, assim como as outras duas, pode ser trocada por regras mais elaboradas. Esta simplificação busca simplificar o exemplo. Da mesma forma, a âncora “@vtd1” contém apenas as formas “comer” e “comeu”, podendo ser reaproveitada para um conjunto maior de conjugações deste e de outros verbos transitivos diretos.

Conforme descrito no capítulo 4, tanto a gramática quanto a sentença de entrada são fornecidos, idealmente, pelo SMA, mas podem também ser lidos de arquivos de entrada. Neste último caso algumas abreviaturas podem ser utilizadas para diminuir o espaço gasto. Os rótulos utilizados para as palavras não seguem um conjunto fixo e permitem que o usuário crie seu próprio conjunto de acordo com suas necessidades. A única exigência é que a gramática use rótulos idênticos (inclusive maiúsculas e minúsculas). Algumas abreviaturas mais comuns são o uso de “art” no lugar de “artigo”, “subst” no de “substantivo”, “vtd” no de “verbo transitivo direto”, “sing” no de “singular” e “masc” no de “masculino”.

6.1.1 Uma derivação direta

O primeiro exemplo de derivação é trivial. A sentença de entrada “o menino comeu doce” pode ser derivada diretamente sobre a árvore “icomer”. Quando o agente sintático lê as informações de um arquivo texto ele tem o seguinte formato (a indentação é opcional):

o	art	[masc, sing]
menino	subst	[masc, sing]
comeu	vtd	[tempo=pret]
doce	subst	[masc, sing]

A agenda do primeiro estágio do analisador contém exatamente as informações do texto acima. O segundo estágio percorre o texto e identifica que a palavra “comeu” está presente na ilha “@vtd1” e que esta ilha ancora apenas a estrutura inicial “icomer”. O segundo estágio do analisador passa a conter, depois da ancoragem, uma entrada contendo o texto dividido em segmentos e uma árvore de derivação construída a partir de “icomer”.

Neste ponto o primeiro estágio do analisador fica vazio porque a única sentença de entrada foi ancorada e transferida para o segundo estágio, agora dentro do seguinte formato:

segmento 0

o	art	[masc, sing]
menino	subst	[masc, sing]

segmento 1

doce subst [masc, sing]

âncora

comeu vtd [tempo=pret]

derivação

s(sn(det _NO (#det) subst)
sv _SA(a_advlug, a_instr)(v (vtd1(comeu))
sn(det _NO (#det) subst)))

A troca de informações entre estágios pode ocorrer de forma síncrona ou assíncrona dependendo da configuração do agente. Na forma síncrona os estágios são ativados em uma ordem circular constantemente e verificam a disponibilidade de informações nas agendas dos estágios anteriores. Na forma assíncrona cada estágio verifica seu estado (ocupado ou desocupado) e lê a agenda do estágio anterior quando seu critério de decisão² for apropriado e verifica se existem dados disponíveis. Caso existam ele os processa; caso não existam ele hiberna por um tempo pré-determinado antes de fazer nova verificação. Um estágio, operando em forma assíncrona pode ser ativado por um agente do SMA sempre que necessário através da interface de comunicação. Na forma síncrona as solicitações externas somente serão atendidas na próxima ativação circular.

A partir das informações contidas na entrada do segundo estágio o terceiro estágio identifica a existência de um seletor de adjunção e dois marcadores de nodo opcional (“_NO”). Para retirar os marcadores de opcional, a árvore derivada do segundo estágio é transformada em quatro alternativas: com o segundo nodo opcional e com o primeiro (1), com o segundo e sem o primeiro (2), com o primeiro e sem o segundo (3) e, finalmente, sem nenhum dos dois nodos (4). As quatro combinações geradas para a árvore derivada são:

```

1) s( sn( det( #det) subst)
    sv _SA(a_advlug, a_instr)( v( @vtd1( comeu))
                                sn( det( #det) subst)) )

```

²Uma taxa de ocupação de recursos da máquina anfitriã definida previamente.

- 2) `s(sn(subst)
 sv _SA(a_advlug, a_instr)(v(@vtd1(comeu))
 sn(det(#det) subst)))`
- 3) `s(sn(det(#det) subst)
 sv _SA(a_advlug, a_instr)(v(@vtd1(comeu))
 sn(subst)))`
- 4) `s(sn(subst)
 sv _SA(a_advlug, a_instr)(v(@vtd1(comeu))
 sn(subst)))`

Cada uma das quatro alternativas ainda possui o seletor de adjunção original que permite, mas não obriga (é um “_SA”), que duas árvores auxiliares sejam adjuntadas, “a_advlug” e “a_instr”. Este seletor pode gerar quatro alternativas de derivação para cada uma das quatro alternativas geradas pelos marcadores de nodos opcionais: sem nenhuma adjunção, adjuntando somente “a_advlug”, adjuntando “a_instr” e, adjuntando primeiro “a_advlug” e depois “a_instr” nesta ordem. O processo de análise sintática computa, desta forma, dezesseis alternativas de derivação, nem todas promissoras, a partir da combinação de três árvores.

O objeto responsável pelo terceiro estágio sabe, olhando a requisição do segundo estágio, que existem quatro palavras na sentença de entrada e que uma delas funciona como âncora, ou seja, já foi afixada na árvore de derivação. Ele também sabe, olhando as alternativas geradas pelo marcador de opcional, que a alternativa (1) possui cinco nodos folhas, sendo uma delas ocupada pela âncora. Desta forma a alternativa (1) não é interessante como opção de derivação porque tem nodos-folha demais. É importante lembrar que não são permitidas derivações vazias no formalismo usado pelo agente sintático, o que conseqüentemente, impede que o número de nodos-folha decresça durante uma derivação. As alternativas (2) e (3) possuem quatro folhas cada uma e a alternativa (4) possui três, o que torna todas elas viáveis.

Ao descartarmos uma das alternativas gerada pela remoção dos marcadores de opcionais, a de número (1), o número de opções de derivação cai de dezesseis para doze. O analisador ainda verifica quantos nodos-folha novos são adicionados em cada adjunção. Isto é feito contando quantos nodos-folha cada árvore auxiliar possui e diminuindo um deste valor³. Esta subtração ocorre porque um dos nodos desta árvore auxiliar é o nodo pé, que será descartado na adjunção. Desta forma sabemos que a árvore auxiliar “a_advlug” possui três nodos-folha e vai adicionar dois nodos-folha (3 folhas - 1 pé = 2 nodos novos) na árvore derivada. Isto impossibilita sua adjunção nas árvores (2), (3) e (4) porque as árvores derivadas resultantes teriam seis nodos-folha nos dois primeiros casos e cinco no

³Nodo-folha não é sinônimo de terminal.

terceiro, sempre contando com a âncora, enquanto a sentença de entrada possui apenas quatro palavras. Isto reduz as alternativas de doze para seis.

Usando a mesma lógica de contagem de nodos para a árvore auxiliar "a_instr", sua adjunção fica impossibilitada de ocorrer em todas as três alternativas. Isto porque ela gera três novos nodos (4 folhas - 1 pé = 3 nodos novos) o que elevaria o total de folhas das alternativas (2) e (3) para sete e da alternativa (4) para seis. Descartando esta alternativa de adjunção nossas alternativas de derivação caem de seis para três. Esta seleção de alternativas não ocorre depois da adjunção, ou seja, o analisador não adjunta para depois contar o número nodos folhas da árvore resultante. Ele calcula este resultado antes de realizar a operação.

Cada uma das três alternativas não descartadas gera uma entrada livre de seletores de adjunção e de marcadores de nodo opcional na agenda do quarto estágio do analisador. Cada uma destas entradas contém a sentença de entrada, segmentada da mesma forma que no estágio dois, e mais uma das árvores derivadas:

- A) s(sn(subst)
 sv(v(@vtd1(comeu))
 sn(det(#det) subst)))
- B) s(sn(det(#det) subst)
 sv(v(@vtd1(comeu))
 sn(subst)))
- C) s(sn(subst)
 sv(v (@vtd1(comeu))
 sn(subst)))

O quarto estágio do analisador do agente seleciona uma alternativa de cada vez e tenta acomodar a sentença de entrada. Neste caso ele obtém sucesso na estrutura (B), que é fornecida como solução na seguinte forma:

segmento 0

o	art	[masc, sing]
menino	subst	[masc, sing]

segmento 1

doce	subst	[masc, sing]
------	-------	--------------

âncora

comeu	vtd	[tempo=pret]
-------	-----	--------------

derivação

```
s( sn( det( #det( o [masc, sing])) subst( menino [masc, sing]))
  sv( v( @vtd1( comeu [tempo=pret])
      sn( subst( doce [masc, sing])) ) )
```


As duas alternativas restantes não são descartadas de imediato. O quarto estágio busca alternativas de substituição nas árvores iniciais da gramática. Na gramática exemplo não existem substituições possíveis. Caso existissem, este estágio geraria todas as combinações possíveis e as reagendaria no segundo estágio. Como vimos, o segundo estágio expande os seletores e marcadores de opcional, o que garante que cada alternativa reagendada seja tratada corretamente mesmo quando possuem novos seletores e marcadores.

6.1.2 Usando a adjunção

Para ilustrar um exemplo mais complexo vamos incrementar a sentença do exemplo anterior e manter a gramática idêntica. Tomemos a seguinte sentença (sem os traços morfo-sintáticos para facilitar a leitura):

o	art	[]
menino	subst	[]
comeu	vtd	[]
doce	subst	[]
no	prep	[]
quarto	subst	[]

O primeiro e segundo estágios da análise seguem idênticos, mas o terceiro estágio não pode mais descartar a alternativa (1) porque agora a sentença de entrada tem palavras suficientes (cinco folhas contra cinco palavras, contando a âncora).

Das dezesseis alternativas de derivação, no entanto, oito são descartadas. Quatro se perdem porque a adjunção das duas árvores auxiliares em sequência geraria cinco novos nodos-folha (dois de “a_advlug” e três de “a_instr”). Se estas adjunções ocorressem, a alternativa (1) ficaria com dez nodos, as alternativas (2) e (3) com nove nodos e a alternativa (4) com oito, contra seis palavras da sentença de entrada.

Das doze alternativas restantes são ainda descartadas mais três porque a árvore auxiliar “a_instr” (três novos nodos) não pode ser adjuntada nas alternativas (1), que ficaria com oito nodos, (2) e (3), que ficariam com sete nodos cada. Das nove alternativas o analisador descarta mais uma, a adjunção da árvore auxiliar “a_advlug” na alternativa (1), que ficaria com sete nodos.

As oito alternativas restantes, portanto, são as quatro alternativas originais — (1), (2), (3) e (4) — sem adjunção alguma (os seletores são do tipo “_SA”) mais as alternativas geradas pela adjunção de “a_advlug” (dois nodos novos) em (2), (3) e (4) e, ainda, pela adjunção de “a_instr” em (1), que ficaria com seis nodos. As alternativas analisadas seriam, desta forma, as seguintes:

- A) `s(sn(subst)
sv(v(@vtd1(comeu))
sn(subst)))`
- B) `s(sn(subst)
sv(sv(v(@vtd1(comeu))
sn(subst))
alug(prep(#prep) subst)))`
- C) `s(sn(subst)
sv(sv(v(@vtd1(comeu))
sn(subst))
ainstr(v_uso(#indica_uso) sn(det subst))))`
- D) `s(sn(subst)
sv(v(@vtd1(comeu))
sn(det(#det) subst)))`
- E) `s(sn(subst)
sv(sv(v(@vtd1(comeu))
sn(det(#det) subst))
alug(prep(#prep) subst)))`
- F) `s(sn(det(#det) subst)
sv(v(@vtd1(comeu))
sn(subst)))`
- G) `s(sn(det(#det) subst)
sv(sv(v(@vtd1(comeu))
sn(subst))
alug(prep(#prep) subst)))`
- H) `s(sn(det(#det) subst)
sv(v(@vtd1(comeu))
sn(det(#det) subst)))`

A solução encontrada no quarto estágio é a fornecida pela alternativa (G). Como não é possível executar substituições pela ausência de árvores iniciais adequadas o processamento se encerra sem reagendamentos.

Uma sentença com mais palavras, tal como:

o	art	[]
menino	subst	[]
comeu	vtd	[]
doce	subst	[]

no	prep	[]
quarto	subst	[]
usando	vtd	[]
as	art	[]
mãos	subst	[]

permite que mais das dezesseis combinações possíveis da gramática sejam geradas e analisadas. Neste caso apenas uma alternativa seria descartada: a adjunção em sequência de “a_advlug” e “a_instr” em (1). As quinze alternativas geradas são as seguintes:

- A) s(sn(subst)
 - sv(v(@vtd1(comeu))
 - sn(subst)))
- B) s(sn(subst)
 - sv(sv(v(@vtd1(comeu))
 - sn(subst))
 - alug(prep(#prep) subst)))
- C) s(sn(subst)
 - sv(sv(sv(v(@vtd1(comeu))
 - sn(subst))
 - alug(prep(#prep) subst))
 - ainstr(v_uso(#indica_uso) sn(det subst)))))
- D) s(sn(subst)
 - sv(sv(v(@vtd1(comeu))
 - sn(subst))
 - ainstr(v_uso(#indica_uso) sn(det subst)))))
- E) s(sn(subst)
 - sv(v(@vtd1(comeu))
 - sn(det(#det) subst)))
- F) s(sn(subst)
 - sv(sv(v(@vtd1(comeu))
 - sn(det(#det) subst))
 - alug(prep(#prep) subst)))
- G) s(sn(subst)
 - sv(sv(sv(v(@vtd1(comeu))
 - sn(det(#det) subst))
 - alug(prep(#prep) subst))
 - ainstr(v_uso(#indica_uso) sn(det subst)))))

- H) s(sn(subst)
 sv(sv(v(@vtd1(comeu))
 sn(det(#det) subst))
 ainstr(v_uso(#indica_uso) sn(det subst))))
- I) s(sn(det(#det) subst)
 sv(v(@vtd1(comeu))
 sn(subst)))
- J) s(sn(det(#det) subst)
 sv(sv(v(@vtd1(comeu))
 sn(subst))
 alug(prep(#prep) subst)))
- K) s(sn(det(#det) subst)
 sv(sv(sv(v(@vtd1(comeu))
 sn(subst))
 alug(prep(#prep) subst))
 ainstr(v_uso(#indica_uso) sn(det subst))))
- L) s(sn(det(#det) subst)
 sv(sv(v(@vtd1(comeu))
 sn(subst))
 ainstr(v_uso(#indica_uso) sn(det subst))))
- M) s(sn(det(#det) subst)
 sv(v(@vtd1(comeu))
 sn(det(#det) subst)))
- N) s(sn(det(#det) subst)
 sv(sv(v(@vtd1(comeu))
 sn(det(#det) subst))
 alug(prep(#prep) subst)))
- O) s(sn(det(#det) subst)
 sv(sv(v(@vtd1(comeu))
 sn(det(#det) subst))
 ainstr(v_uso(#indica_uso) sn(det subst))))

A solução encontrada neste caso é a de letra (K).

6.2 Reconhecendo referências cruzadas

No capítulo 3 apresentamos as BMGs — *Bound-Movement Grammars* — e mostramos como este formalismo é capaz de reconhecer referências cruzadas que as XGs — *Extrapo-*

sition Grammars — não podem.

O formalismo simplificado proposto também é capaz de reconhecer a mesma sentença e de gerar a mesma árvore de derivação. Dada a sentença de entrada⁴:

o	art	[]
livro	subst	[]
que	prel	[]
a	prep	[]
uma	art	[]
amiga	subst	[]
o	art	[]
João	subst	[]
deu	vti	[]
desapareceu	vi	[]

e utilizando a gramática:

iniciais

```
<vi_ini> s( sn _SA(rel1)( det( #det) subst)
           sv(vi( @vi)) )
<rel_topicalizada> rel_topicalizada( sp( prep sn( det( #det) subst))
                                       s( sn( det( #det) subst)
                                       sv(vti)) ) auxiliares
<rel1> sn( sn relativo( cabeça( #que) rel_topicalizada) )
```

âncoras

```
@vi = {desapareceu}:{vi_ini}
```

palavras

```
#det = {o, os, a, as, um, uma, uns, umas, aquela, aquele}
#que = {que}
```

são analisadas duas alternativas de derivação e um reagendamento, que é gerado quando uma tentativa de resolução falha e existem substituições possíveis no quarto estágio do analisador. A árvore derivada obtida como resposta é:

⁴Foram utilizadas quatro novas abreviaturas neste exemplo: “prel” no lugar de “pronome relativo”, “prep” no lugar de “preposição”, “vti” no lugar de “verbo transitivo indireto” e “vi” no lugar de “verbo intransitivo”.

```

s( sn( sn( det( #det( o)) subst( livro))
    relativo( cabeça( #que( que))
        rel_topicalizada( rel_topicalizada( sp( prep( a)
            sn( det( #det( uma)) subst( amiga)))
        s( sn( det( #det( o)) subst( João))
            sv( vti( deu))))))
    sv( vi( @vi( desapareceu))))

```

6.3 Provendo recursão

Apesar do formalismo simplificado não permitir a recursão direta nos nodos pés das árvores auxiliares, devido a algumas restrições derivadas do tipo de comunicação de que o agente sintático é capaz, um artifício pode ser utilizado para contornar este problema.

A seguinte gramática utiliza uma substituição para fornecer a recursão na adjunção por meios indiretos:

iniciais

```

<poesiai> s( sn( subst)
            sv( v( @poesia)
                sn _SA(rec)( subst)) )
<sni>     sni(sn _SA(rec)( subst) )

```

auxiliares

```

<rec>     sn( sn prel(#que)
            relat(vtd sni) )

```

ancoras

```

@poesia = {amava}:{poesiai}

```

palavras

```

#det = {o, os, a, as, um, uns, uma, umas, aquela, aquele, ele}
#que = {que}

```

Esta gramática é capaz de reconhecer sentenças como, por exemplo:

João	subst	[]
amava	vtd	[]
Maria	subst	[]
que	prel	[]
amava	vtd	[]
Pedro	subst	[]
que	prel	[]

amava	vtd	[]
Juliana	subst	[]
que	prel	[]
comia	vtd	[]
bolinho	subst	[]

A árvore de derivação resultante desta sentença é:

```

s( sn( subst( João))
  sv(v( @poesia( amava))
    sn(sn ( subst( Maria)) prel(#que( que))
      relat(vtd ( amava)
        sni( sni( sn( sn( subst( Pedro)) prel( #que( que))
          relat(vtd( amava)
            sni( sni( sn( sn( subst( Juliana)) prel( #que( que))
              relat( vtd( comia)
                sni( sni( sn( subst( bolinho))))))))))))))

```

6.4 Exemplos lingüísticos

Para demonstrar como o analisador pode manipular sentenças mais complexas, foram definidos exemplos lingüísticos envolvendo coordenações e termos elípticos. Para manipular alguns tipos de elipses podemos definir a seguinte gramática:

iniciais

```

<vti> s(s _SA(coord1)(sn( det _NO( #det) subst)
          sv( vti( @vti) prep sn( det _NO( #det) subst)) ) )
<vti-part> s( s _SA(coord2, coord5)(
          sn( det _NO( #det) subst)
          sv( vaux vtdi( @vtdi-part)
            sn( det _NO( #det) subst)
            prep sn( det _NO( #det) subst)) ) )
<vtdi> s( s _SA(coord6)(sn( det _NO( #det) subst)
          sv( vtdi( @vtdi) sn( det _NO( #det) subst)
            prep sn( det _NO( #det) subst)) ) )
<vtd> s(s _SA(coord3)(sn( det _NO( #det) subst)
          sv( vtd( @vtd) sn( det _NO( #det) subst _SA( posse))) ) )
<el> gapping_t1( sn( det _NO ( #det) subst) prep subst )
<e0> gapping_t2(sn( det _NO ( #det) subst)

```

```

        sn( det _NO( #det) subst))
<e2> ellipse_t2( sn( det _NO( #det) subst)
        repete _SA( part_repete)( #repete))
<e3> ellipse_t3( s _SA( coord4)(
        sn( det _NO( #det) subst)
        sv _SA( nega)( vtd sn( det( #det) subst)) ) )
<e4> reduction_vtdi( sv( vtdi sn( det _NO( #det) subst)
        prep sn( subst) ) )
<simpsn> simpsn( subst)

```

auxiliares

```

<nega> sv( negação( #nega) sv)
<part_repete> repete( repete vaux)
<posse> subst( subst prep pposs simpsn)
<coord1> s( s conj( #conj) gapping_t1)
<coord2> s( s conj( #conj) ellipse_t2)
<coord3> s( s conj( #conj) ellipse_t3)
<coord4> s( s prep prel ellipse_t2)
<coord5> s( s conj( #conj) reduction_vtdi)
<coord6> s( s conj( #conj) gapping_t2)

```

âncoras

```

@vti = {gosta}:{vti}
@vtdi_part = {dado, comprado}:{vtdi_part}
@vtdi = {deu}:{vtdi}
@vtd = {conquistou}:{vtd}

```

palavras

```

#det = {o, os, a, as, um, uns, uma, umas, muitos, esses, sua}
#conj = {e}
#repete = {tambem}
#nega = {não}

```

Esta gramática é capaz de reconhecer sentenças complexas caracterizando o tipo de termo elíptico na cláusula coordenada. Esta caracterização permite que se saiba quantos e quais termos são elípticos e o que buscar na estrutura sintática para preenchê-los. Apesar desta gramática ter essa capacidade, ela não pretende resolver o problema das elipses em cláusulas coordenadas, apenas demonstrar a capacidade do analisador. Conforme mencionado no capítulo 5, a construção de uma gramática robusta para o português faz parte dos trabalhos futuros.

Tomemos como exemplo a seguinte sentença:

João	subst	[]
gosta	vti	[]
de	prep	[]
cinema	subst	[]
e	conj	[]
Pedro	subst	[]
de	prep	[]
teatro	subst	[]

Após analisar treze alternativas, incluindo os reagendamentos, o sistema identifica a seguinte solução:

```
s( s( s(sn( subst( João))
      sv(vti( @vti( gosta))
            prep( de) sn( subst( cinema))))
  conj( #conj( e))
  gapping_t1( gapping_t1( sn( subst( Pedro))
                          prep( de) subst( teatro))))
```

Uma sentença um pouco mais complexa, também contendo termos elípticos, é a seguinte:

João	subst	[]
tinha	vaux	[]
dado	vtdi	[]
esses	pdem	[]
livros	subst	[]
ao	prep	[]
filho	subst	[]
e	conj	[]
Maria	subst	[]
tambem	adv	[]
tinha	vaux	[]

Usando a mesma gramática, o sistema analisa cinquenta e oito alternativas e fornece a seguinte solução:

```

s( s( s(sn( subst( João))
      sv( vaux( tinha ) vtdi( @vtdi_part( dado))
      sn( det( #det( esses)) subst( livros))
      prep( ao) sn( subst( filho))))
  conj( #conj( e))
  ellipse_t2( ellipse_t2( sn( subst( Maria))
                        repete( repete( #repete( tambem)) vaux( tinha))))))

```

Também é possível analisar duas ou mais coordenações agrupadas. O mecanismo de traços morfo-sintáticos pode ser utilizado, neste caso, para identificar quais constituintes devem ser referenciados. Vejamos a seguinte sentença:

João	subst	[]
conquistou	vtd	[]
a	art	[]
confiança	subst	[]
de	prep	[]
seu	pposs	[]
chefe	subst	[]
e	conj	[]
Maria	subst	[]
não	neg	[]
admite	vtd	[]
a	art	[]
hipotese	subst	[]
de	prep	[]
que	prel	[]
Pedro	subst	[]
tambem	adv	[]
tenha	vaux	[]

Após avaliar duzentas e noventa e quatro alternativas o sistema determina que a solução da sentença é:

```

s( s( s( sn( subst( João))
      sv( vtd( @vtd( conquistou))
      sn( det( #det( a)) subst( subst( confiança)
        prep( de) pposs( seu)
        simpsn( simpsn( subst( chefe)))))))))
conj( #conj( e))
elipse_t3( elipse_t3( s( s( sn( subst( Maria))
      sv( negação( #nega( não)) sv( vtd( admite)
      sn( det( #det( a)) subst( hipotese))))))
      prep( de) prel( que)
      elipse_t2( elipse_t2( sn( subst( Pedro ))
        repete( repete( #repete( tambem))
        vaux( tenha)))))))))

```

Nesse exemplo podemos observar claramente que o número de combinações analisadas cresce de acordo com o número de palavras da sentença de entrada e de acordo com a ambigüidade da gramática. Ao explicitarmos os seletores de adjunção tornamos possível que o autor da gramática controle o grau de ambigüidade. Este exemplo também nos dá indícios de que a paralelização pode ser interessante, na medida em que o analisador proposto disponibiliza as alternativas uma a uma e não em lote.

Um outro tipo de elipse pode ser observado na seguinte sentença de entrada:

João	subst	[]
tem	vaux	[]
comprado	vtd	[]
muitos	adjadn	[]
livros	subst	[]
aos	prep	[]
filhos	subst	[]
e	conj	[]
oferecido	vtdi	[]
muitos	adjadn	[]
discos	subst	[]
a	prep	[]
mulher	subst	[]

O sistema, sempre usando a mesma gramática, analisou noventa e quatro alternativas de derivação e determinou que a solução para a sentença é:

```

s( s( s(sn( subst( João))
      sv( vaux( tem) vtdi( @vtdi_part( comprado))
      sn( det( #det( muitos)) subst( livros ))
      prep( aos) sn( subst( filhos))))
  conj( #conj( e))
  reduction_vtdi( reduction_vtdi( sv( vtdi( oferecido)
                                     sn( det( #det( muitos)) subst( discos))
                                     prep(a) sn( subst(mulher)))))))

```

Para o agente sintático não importa quantos elementos elípticos existem em uma sentença, desde que a gramática preveja a possibilidade de sua existência usando marcadores de nodo opcional ou multiplicando o número de árvores. Um exemplo de dois termos elípticos em uma coordenação pode ser visto na seguinte sentença:

João	subst	[]
deu	vtd	[]
flores	subst	[]
a	prep	[]
sua	pposs	[]
mãe	subst	[]
e	conj	[]
Pedro	subst	[]
chocolates	subst	[]

Após analisar vinte e oito alternativas, o agente sintático determinou que a solução para este exemplo é:

```

s( s( s(sn( subst( João))
      sv( vtdi( @vtdi(deu))
      sn( subst( flores))
      prep( a) sn( det( #det( sua)) subst( mãe))))
  conj( #conj( e))
  gapping_t2( gapping_t2( sn( subst( Pedro))
                          sn( subst( chocolates )))))

```

6.5 Comentários sobre a paralelização

Conhecendo o funcionamento dos estágios do analisador e observando os exemplos deste capítulo podemos inferir que alguns estágios do analisador podem ficar ociosos enquanto

outros ficariam sobrecarregados. O terceiro estágio, por exemplo, manipula os marcadores de nodo opcional e os seletores de adjunção, consumindo praticamente todo o tempo de processamento do analisador para os exemplos fornecidos.

Isto nem sempre é verdade, na medida em que estamos trabalhando com exemplos fabricados. Em uma gramática robusta, capaz de analisar textos reais, o primeiro estágio do sistema seria bastante exigido porque seria obrigado a pesquisar o SMA em busca da classificação categorial das palavras e de seus traços morfo-sintáticos.

Da mesma forma, com uma gramática mais robusta, a ancoragem das sentenças pode envolver a pesquisa de uma base de dados considerável antes de ser possível segmentar o texto e escolher um conjunto de ancoragens. O quarto estágio, por sua vez, pode ter sua carga de trabalho aumentada ao comunicar os resultados para o SMA.

O formato do analisador, em estágios independentes, permite que se especifiquem políticas especiais de processamento. Caso se identifique que uma aplicação específica exija mais de um estágio de análise que dos demais, pode-se especializar um ou mais agentes para realizar apenas uma etapa da análise. Pode-se, por exemplo, dedicar três ou quatro agentes, residentes em máquinas anfitriãs ou processadores diferentes, para realizar apenas as tarefas do terceiro estágio. Estes agentes aceitariam somente solicitações neste formato e requisitariam o processamento das alternativas a outros agentes.

Esta configuração especial de processamento torna o analisador superescalar, no sentido de multiplicar unidades de processamento para operações específicas. Esta é uma maneira alternativa de utilizar a capacidade deste modelo de balancear o custo da análise entre máquinas ou processadores diferentes.

Referências Bibliográficas

- [1] A. Abeillé. A lexicalized tree adjoining grammar for French: the general framework. Technical Report MS-CIS-88-64, University of Pennsylvania, Philadelphia, PA 19104-6389, 1988.
- [2] A. Abeillé, K. Bishop, S. Cote, and Yves Schabes. A lexicalized tree adjoining grammar for English. Technical Report MS-CIS-90-24, University of Pennsylvania, Philadelphia, PA 19104-6389, 1990.
- [3] Paulo Ricardo Carneiro Abrahão. Modelagem e implementação de um léxico semântico para o português. Master's thesis, Instituto de Informática, PUCRS, Porto Alegre, Dezembro 1997.
- [4] Paulo Ricardo Carneiro Abrahão and Vera Lúcia Strube de Lima. Modelagem e implementação de um léxico semântico para linguagem natural. In *Anais do 1o. Encontro Nacional de Inteligência Artificial, ENIA '97*, Brasília, DF, Agosto 1997.
- [5] A. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, MA, 1986.
- [6] A. Aho and J. D. Ullman. *Theory of Parsing, Translation and Compiling: parsing*. Prentice-Hall, Englewood Cliffs, NJ, 1973.
- [7] James Allen. *Natural Language Understanding*. Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.
- [8] Luis Otavio Alvares, Yves Demazeau, and Uma Garimella. On problem decomposition in distributed artificial intelligence. In *First Ibero American Workshop on Distributed Artificial Intelligence and Multi-Agent Systems*, October 1996.
- [9] Andrew W. Appel. *Modern compiler implementation in Java*. Cambridge University Press, New York, NY, 1997.
- [10] C. Beardon, D. Lumdsem, and G. Holmes. *Natural Language and Computational Linguistics*. Ellis Horwood Ltda., UK, 1991.

- [11] O. Boissier and Yves Demazeau. A distributed artificial intelligence view on general purpose vision systems. In E. Werner and Yves Demazeau, editors, *Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 311–330, Kaiserslautern, August 1991. Elsevier Science Publishers B. V.
- [12] Ronald V. Book. *Topics in formal language theory*. Automatic Computation. Prentice-Hall, New Jersey, NY, 1973.
- [13] Jean W. Bresnan, Ronald Kaplan, Stanley Peters, and Annie Zaenen. Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 1982.
- [14] E. Cardozo, Jaime Simão Sichman, and Yves Demazeau. Using the active object model to implement multi-agent systems. In *Proceedings of Fifth IEEE International Conference on Tools with Artificial Intelligence*, Boston, USA, 1993.
- [15] Eugene Charniak. Natural language learning. *Revista da ACM*, 1995.
- [16] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.
- [17] Noam Chomsky. *Lectures on Government and Binding*. Foris Pub, Holland, 1982.
- [18] Noam Chomsky. *Barriers*. MIT Press, Cambridge, MA, 1986.
- [19] E. Csuhaj-Varjú and Reda A. Alez. Multi-agent systems in technology. In K. Sikkel and A. Nijholt, editors, *Twenth Workshop on Language Technology*. 1993.
- [20] Hamish Cunningham, Mike Freeman, and William J. Black. Software reuse, object-oriented frameworks and natual language processing. In *International Conference on New Methods in Language Processing*, Manchester, UK, 1994.
- [21] João Luís Tavares da Silva. Algumas considerações sobre a resolução da ambigüidade léxica no processamento da língua natural. In *Anais do II Encontro para Processamento Computacional de Português Escrito e Falado*, Curitiba, PR, 1996.
- [22] João Luís Tavares da Silva. Utilização do paradigma multi-agentes no processamento da linguagem natural: um modelo da ambigüidade léxica categorial na língua portuguesa. Master's thesis, Instituto de Informática, PUCRS, Abril 1997.
- [23] João Luís Tavares da Silva and Vera Lúcia Strube de Lima. An alternative approach to lexical categorial disambiguation using a multi-agent systems architecture. In *RANLP'97*, pages 330–335, Tzigov Chark, Bulgária, 1997.

- [24] Daniel da Silva de Paiva, Ariadne M. B. R. Carvalho, and Jacques Wainer. Um sistema para processamento distribuído de linguagem natural. In *Anais do II Encontro para Processamento COmputacional de Português Escrito e Falado*, Curitiba, PR, 1997.
- [25] Daniel Pereira da Silva de Paiva. Um sistema multi-agentes para processamento distribuído de linguagem natural. Master's thesis, Instituto de Computação—Unicamp, Campinas, SP, 1996.
- [26] Vera Lúcia Strube de Lima, João Luis Tavares da Silva, and Flávio Oliveira. O enfoque paralelo-distribuído e o projeto nalamas. In *II Encontro para o Processamento Computacional de Portugus Escrito e Falado*, pages 129–138, Curitiba, PR., Outubro 1996. CEFET.
- [27] Vera Lúcia Strube de Lima and Ivan Santa Maria Filho. O processo de desenvolvimento de software para o processamento da linguagem natural: da engenharia à arquitetura. Technical report, PUCRS, Porto Alegre, RS, 1997.
- [28] Yves Demazeau. La plateforme paco et ses applications. In *2ème Journée Nationale du PRC-IA sur les Système Multi-Agents*, Montpellier, 1993. PRC-IA.
- [29] M. Devos and G. Adriaens. The parallel expert parser (pep): A thoroughly revised descendant of the word expert parser (wep). In *Proceedings of the 12th International Conference on Computational Linguistics —COLING'88*, pages 142–147, 1988.
- [30] M. Devos and G. Adriaens. The parallel expert parser (pep). In G. Adriaens and Udo Hahn, editors, *Parallel Natural Language Processing*, pages 350–375. Ablex, 1994.
- [31] J. Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1968.
- [32] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2), 1970.
- [33] R. Wessen et al. Network structures for distributed situation assessment. *IEEE Trans on Systems, Man and Cybernetics*, 11(1):5–23, Jan/Fev 1981.
- [34] Ivan Santa Maria Filho, Vera Lúcia Strube de Lima, and Rove Luiza de Oliveira Chishman. Indexação semântica de documentos: uma aplicação da teoria do léxico gerativo. In *PROPOR'99*, Portugal, 1999. Springer-Verlag.
- [35] M. Corrêa Filho. *A Arquitetura de Diálogos entre Agentes Cognitivos Distribuídos*. PhD thesis, COPPE/UFRJ, Rio de Janeiro, RJ, 1994.

- [36] T. Finin and R. Fritzson. Kqml as an agent communication language. In *Third International Conference on Information and Knowledge Management*. ACM Press, 1994.
- [37] Robert Frank. Computation and linguistic theory: A government and binding theory parser using tree adjoining grammar. Master's thesis, University of Pennsylvania, Philadelphia, PA, 1990.
- [38] Catherine Fuchs. *Linguistique at Traitment Automatique des Langues*. Hachete, Paris, 1993.
- [39] Danilo Fum, Giovanni Guida, and Carlos Tasso. A distributed multi-agent architecture for natural language processing. In *COLING'88. 12th International Conference on Computational Linguistics*, Budapest, Hungria, August 1988. COLING.
- [40] A. V. Groenink. Tractability issues in extraposition grammar. Technical report, National Research Institute for Mathematics and Computer Science in the Netherlands, 1996.
- [41] Udo Hahn and Geert Adriaens. *Parallel Natural Language Processing: Background and Overview*, chapter 1. Ablex Publishing Corporation, Norwood, New Jersey, 1994.
- [42] G. Hirst. *Semantic interpretation and the resolutions of ambiguity*. Cambridge University Press, 1988.
- [43] II Workshop NALAMAS. *Lexicalização de um Analisador sintático baseado em Tree Adjoining Grammars*, Porto Alegre, RS, 1997.
- [44] IV Escola Regional de Informática. *Processamento da Linguagem Natural: premissas e desafios*, Canoas, RS, Abril 1996.
- [45] Ivan Izquierdo. A palavra escrita. *Jornal Correio do Povo*, Porto Alegre, RS, 10 de Setembro 1996.
- [46] R. Jackendoff. X-bar syntax: a study of phrase structure. *Linguistic Inquiry monograph two*. MIT Press., 1977.
- [47] Aravind K. Joshi. Tree adjoining grammars: How much context sensitivity is required to provide reasonable descriptions? In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*. Cambridge University Press, Cambridge, MA, 1985.

- [48] Aravind K. Joshi. An introduction to tree adjoining grammars. In Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1987.
- [49] Aravind K. Joshi, Leon S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of the Computer and System Sciences*, 1975.
- [50] Aravind K. Joshi, Leon S. Levy, and M. Takahashi. Local constraints. *SIAM Journal of Computing*, 1978.
- [51] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars and lexicalized grammars. In *Definability and Recognizability of Sets of Trees*. Elsevier, 1991.
- [52] Aravind K. Joshi, K. Vijay-Shanker, and D. Weir. The convergence of mildly context-sensitive grammatical formalisms. In P. Sells, S. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*. MIT Press, Cambridge, MA, 1991.
- [53] G. Edward Barton Jr., Robert C. Berwick, and Eric Sven Ristad. *Computational Complexity and Natural Language*. Bradford Book, London, UK, 1988.
- [54] Lauri Karttunen and Arnold M. Zwicky. *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*. Cambridge University Press, Cambridge, MA, 1985.
- [55] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Technical report, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.
- [56] Karin Christine Kipper. Uma experiência de utilização do formalismo das gramáticas de adjunção de árvores para a língua portuguesa. Master's thesis, CPGCC, UFRGS, Porto Alegre, RS, 1994.
- [57] J. Koning and Yves Demazeau. Collaborative learning using dai interaction protocols in a telecommunication setting. In *ITP'95: Interacting Agents*, Plodice, Junho 1995.
- [58] Gabriel Pereira Lopes, Vitor Rocio, Rosa M. Viccari, and Emiliano Padilha. Bound-movement grammar for natural language parsing. In *Anais do II Encontro para o Processamento Computacional de Português Escrito e Falado*. SBIA'96, Curitiba, PR, 1996. XIII Simpósio Brasileiro de Inteligência Artificial, SBIA'96.
- [59] M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In V. Lesser, editor, *Proceedings of First International Conference on Multi-Agent Systems*, pages 254–260, San Francisco, CA, junho 1995. AAAI Press/ MIT Press.

- [60] Ralph Moreira Maduro. Um sistema multi-agentes para tratamento de elipse em cláusulas coordenadas. Technical report, Unicamp, 1997.
- [61] Bill Z. Manaris and Brian M. Slator. Interactive natural language processing: Building on success. *IEEE. Computer-innovative technology for computer professionals*, 1996.
- [62] Marvin Minsky and D. Riecken. A conversation with marvin minsky about agents. *Communications of the ACM*, 37(7):22–29, julho 1994.
- [63] Anton Nijholt. *Computers and Languages - Theory and Practice*. North Holland, 1988.
- [64] Michael A. Palis and Sunil Shende. Fast parallel recognition of tree-adjoining languages. *Theoretical Computer Science*, 1990.
- [65] Michael A. Palis, Sunil Shende, and David S. L. Wei. An optimal linear-time parallel parser for tree-adjoining languages. *SIAM Journal of Computing*, 1990.
- [66] Ivandré Paraboni. Uma arquitetura para resolução de referências pronominais possessivas no processamento de textos em língua portuguesa. Master's thesis, Instituto de Informática, PUCRS, Porto Alegre, RS, 1997.
- [67] Fernando C. N. Pereira. Extraposition grammars. *American Journal of Computational Linguistics*, 7(4), December 1981.
- [68] Fernando C. N. Pereira and David H. D. Warren. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 2(13), 1980.
- [69] Steven Pinker. *The Language Instinct — How the Mind Creates Language*. Harper-Perennial, New York, NY, 1995.
- [70] James Pustejovsky. *The Generative Lexicon*. MIT Press, Cambridge, MA, 1995.
- [71] Owen Rambow. *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, university of Pennsylvania, Philadelphia, PA, 1994.
- [72] J. R. Ross. Excerpts from constraints on variables in syntax. In *On Noam Chomsky, critical essays*. Anchor Press, New York, NY, 1974.
- [73] P. Russel. Prolog: Manuel de référence at utilisation. Technical report, Groupe d'Intelligence Artificielle, U. E. R. de luminy, Université d'Aix-Marseille II, 1975.

- [74] A. Salomaa. Formal languages and power series. In Jan Van Leeuwen, editor, *Formal Models and Semantics*, volume B. MIT Press, Cambridge, MA, 1992.
- [75] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.
- [76] Yves Schabes, A. Abeillé, and Aravind K. Joshi. New parsing strategies for tree adjoining grammars. In *Proceedings of 12th International Conference on Computational Linguistics, COLING'88*, 1988.
- [77] Yves Schabes and Aravind K. Joshi. Parsing with lexicalized tree adjoining grammar. Technical Report MS-CIS-90-11 LINC LAB 164, University of Pennsylvania, Philadelphia, PA, 1990.
- [78] Pascale Sebillot. *Modelisation de Mecanismes de Filtrage par Arbres Traits dans les Grammaires logiques pour L'Analyse Automatique du Language Naturel*. PhD thesis, L'Institut National des Sciences Appliquees de Rennes, Rennes, 1989.
- [79] Sunil Shende. *Computational Feasibility of some Constraint Formalisms for Natural Language*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.
- [80] Jaime Simão Sichman. *Du Raisonnement Social Chez les Agents: une Approche Fondée sur la Théorie de la Dependance*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, França, 1995.
- [81] Jaime Simão Sichman, Yves Demazeau, and O. Boissier. When can knowledge-based systems be called agents? In *Proceedings of IX Simpósio Brasileiro de Inteligência Artificial*, Rio de Janeiro, RJ, 1992.
- [82] S. L. Small. *Word Expert Parsing: a Theory of Distributed Word-based Natural Language Understanding*. PhD thesis, University of Maryland, DCS, 1980.
- [83] Marie-Hélène Stefanini. *TALISMAN: une architecture multi-agents pour l'analyse du français écrit*. PhD thesis, Université Pierre Mendès-France, Grenoble, França, 1993.
- [84] Marie-Hélène Stefanini and Yves Demazeau. Talisman: a multi-agent system for natural language processing. In *SBIA '95, 12th. Brazilian Symposium on Artificial Intelligence*, Campinas, 1995.
- [85] G. Uma, Luis Otavio Alvares, and Yves Demazeau. On decomposition methodology. *Journal of the IETE*, 42(3):111–116, May-June 1996.

- [86] G. Uma, O. Nalini Kumari, and B. E. Presad. Knowledge base partitioning in distributed intelligence systems. In *Canadian Workshop on DAI*, May 1994.
- [87] K. Vijay-Shanker. *A study of tree adjoining grammars*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1987.
- [88] K. Vijay-Shanker and Aravind K. Joshi. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Meeting of the Association for Computational Linguistics*, Chicago, IL, 1985.
- [89] K. Vijay-Shanker and Aravind K. Joshi. Feature structure based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics, COLING'88*, 1988.
- [90] K. Vijay-Shanker and Aravind K. Joshi. Unification based tree adjoining grammars. In J. Wedekind, editor, *Unification-based Grammars*. MIT Press, Cambridge, MA, 1991.
- [91] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, university of Pennsylvania, Philadelphia, PA, 1988.
- [92] The Research Group XTAG. A lexicalized tree adjoining grammars for english. Technical report, University of Pennsylvania, Walnut Street 3401, Suite 400C. Philadelphia, PA. 19104-6228., 1995. requests online: xtag-request@linc.cis.upenn.edu.
- [93] D. H. Younger. Recognition and parsing of context-free languages in n^3 . *Inf. Control*, 10(2), 1967.
- [94] Rémi Zajac. Towards computer-aided linguistic engineering. In *Proceedings of the International Conference on Computational Linguistics, COLING'92*, Nantes, France, 1992.