Um Sistema Multi-Agentes para Processamento Distribuído de Linguagem Natural

Daniel da Silva de Paiva

Um Sistema Multi-Agentes para Processamento Distribuído de Linguagem Natural[*]

Daniel da Silva de Paiva[**]

INSTITUTO DE COMPUTAÇÃO

UNICAMP

Banca Examinadora:

Profª Drª Ariadne Maria Brito Rizzoni Carvalho (orientadora)

Profª Drª Vera Lima Strube de Lima — PUC-RS

Prof. Dr. Jaime Simão Sichman — POLI-USP

---

[*] Dissertação apresentada ao Instituto de Computação como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

[**] Bacharel em Informática pela Universidade Federal de Viçosa.

Um Sistema Multi-Agentes para Processamento Distribuído de Linguagem Natural

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida por Daniel da Silva de Paiva e aprovada pela comissão julgadora.

Campinas, 20 de dezembro de 1996.

Ariadne Maria Brito Rizzoni Carvalho
(orientadora)

Jacques Wainer
(co-orientador)

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

UNICAMP

Tese de Mestrado defendida e aprovada em 20 de dezembro de 1996 pela Banca Examinadora composta pelos Professores Doutores

Vera Lúcia Strube de Lima

Profª. Drª. Vera Lúcia Strube de Lima

Prof°. Dr°. Jaime Simão Sichman

Ariadne M. B. R. Carvalho

Profª. Drª. Ariadne Maria B.R. Carvalho

Epigrama nº 5 (Cecília Meireles)

Gosto da gota d'água que se equilibra
na folha rasa, tremendo ao vento.

Todo o universo, no oceano do ar, secreto vibra:
e ela resiste, no isolamento.

Seu cristal simples reprime a forma, no instante incerto:
pronto a cair, pronto a ficar — límpido e exato.

E a folha é um pequeno deserto
para a imensidade do ato.

Dedico este trabalho a meus pais e a meus irmãos.

Resumo

Nesta dissertação será apresentado um Sistema Multi-Agentes para Processamento Distribuído de Linguagem Natural que utiliza a idéia de que a interpretação de sentenças deve vir do inter-relacionamento entre os diferentes *agentes lingüísticos* que compõem o sistema. Os agentes são associados às palavras de uma sentença, sendo *um agente por palavra*. Cada agente terá seu comportamento ditado pela classe morfossintática a que a palavra associada a ele pertence (assume-se que cada palavra pode pertencer a apenas uma classe). De acordo com seu comportamento, cada agente tenta *encontrar* outros agentes com o objetivo de formar estruturas mais complexas e de atingir uma representação que englobe todos os agentes da sentença. Todo esse processo de busca por outro agente é realizado utilizando troca de mensagens entre os agentes. Quando um agente encontra um outro que ele considera ideal para formar uma estrutura mais complexa, então esses agentes *estabelecem ligações*. A representação atinge o nível de sentença quando todos os agentes estão conectados.

Abstract

This thesis will present a Multi-Agent System for Distributed Natural Language Processing that uses the idea that the sentence intepretation will come from the interaction among the *linguistic agents* that comprise the system. Agents are associated with the words of the sentence, one *agent by word*. Each agent behaves in accordance with its word category (it's assumed that each word can belong only to one category). The agents will try to *find* the other agents to *establish associations* aiming at forming structures that can represent the sentence as a whole. When all the agents are connected the sentence level was reached and a solution is said to be found.

Agradecimentos

Não sei andar sozinho por essas ruas
Sei do perigo que nos rodeia pelos caminhos
Não há sinal de sol
Mas tudo me acalma no teu olhar[1].

Antes de mais nada, gostaria de agradecer o incentivo para iniciar o mestrado e o apoio constante de todos da família (pai, mãe e irmãos — Pepê, Marcão e Valéria) e a meus tios e primos.

À Luciana que esteve sempre presente: não há maneiras de agradecer a confiança e o carinho depositados por você em mim.

À Galera da Toca (Celiota, Elbson, Moreba e Drigão) e à Stella, por tudo que há de bom que a convivência pode trazer: vocês são especiais!!!

À Mônica, pela força e carinho nestes últimos dias, pelos girassóis e pelo "Eterno".

Aos meus orientadores, Ariadne e Jacques, pela maneira como conduziram esse trabalho!!!

À Rosa, ao Brancão (Luciano) e irmãos, pelo constante apoio e pelo período em Piracicaba.

Ao Luís, Roseli e Roland, juntamente com Solange, Alda, Vera e Niero, por terem quebrado todos os "galhos" que pintaram pelo caminho.

---

[1] Tavinho Moura e Márcio Borges na música "Cruzada".

Ao Zé Luís, pelo constante incentivo para a carreira acadêmica, e ao Departamento de Informática da UFV, pelo apoio de todos os professores e pelo uso que pude fazer dos equipamentos.

Ao pessoal do prédio onde morei em Campinas (Araribá) especialmente Maranhão (Rodrigo), Linu (André) e Bruce (Ivan), pela música, pelo xadrez, pela bagunça!

À Biana e à Si, pela companhia, música e alegria.

À Ana ("—Afinal, é queijo ou queixo?"), Horácio, Alexandre, Jé, Cecília (companheiros de SBIA) e todo o pessoal das turmas de 94, 95 e 96.

À Mara, pela música: fitas, coral & mineiros! Ao Toninho, por "Gershwin", um dos melhores presentes que recebi!

Aos membros da banca, pela gentileza prestada ao ler essa dissertação em tão exíguo tempo.

Ao Graeme Hirst e ao Geert Adriaens que, mesmo sem me conhecerem, deram-me a maior atenção.

Ao CNPq, pelo apoio financeiro.

Índice Analítico

Resumo iii

Abstract iv

Agradecimentos v

Índice Analítico vii

Índice de Figuras x

Introdução 1

1. Sistemas Multi-Agentes 1-4

1.1 Resolução Distribuída de Problemas versus Sistemas Multi-Agentes 1-5

1.2 Tipos de Agentes 1-6

1.3 Tópicos de Interesse 1-8

1.3.1 Ação 1-9

1.3.2 Comunicação 1-9

1.3.3 Cooperação e Resolução de Conflitos 1-10

1.3.4 Sociedade 1-11

1.4 Considerações Finais sobre o Capítulo 1-12

2. Considerações sobre Processamento Lingüístico I Apresentação de alguns sistemas 2-13

2.1 "*Word Expert Parser — WEP*" 2-14

2.1.1 Interações entre Agentes 2-14

2.1.2 Ações dos Agentes 2-16

2.1.3 Considerações Finais Sobre WEP 2-17

2.2 "*Polaroid Words*" e a Resolução de Ambigüidade Semântica 2-18

2.2.1 O Problema da Ambigüidade 2-19

2.2.2 Utilizando *Polaroid Words* 2-20

2.3 "*Parallel Expert Parser — PEP*" 2-22

2.4 Considerações Finais Sobre o Capítulo 2-23

3. Considerações sobre Processamento Lingüístico II Apresentação de alguns fenômenos lingüísticos 3-24

3.1 Um Esboço de Processamento para Orações Simples 3-25

3.2 Exemplos de Fenômenos Lingüísticos 3-27

3.2.1 Topicalização 3-27

3.2.2 Posposição de Sujeito 3-28
3.2.3 "*PP-Attachment*" 3-29
3.2.4 Construção Relativa 3-30
3.2.5 Coordenação 3-31
3.2.6 Redução 3-32
3.2.7 "*Gapping*" 3-32
3.3 Considerações Finais sobre o Capítulo 3-33
4. Um Sistema Multi-Agentes para Processamento de Linguagem Natural 4-34
4.1 Motivações 4-35
4.2 As Possibilidades de Segmentação 4-35
4.3 Modelo de Forças de Atração e Mecanismos de Restrição 4-37
4.3.1 Forças de Atração 4-37
4.3.2 Mecanismos de Restrição 4-39
4.4 Processo de Interpretação para Agentes 4-40
4.4.1 A Fase de Busca 4-41
4.4.1.1 Mapeando o Comportamento dos Agentes 4-41
4.4.1.2 A Questão da Extensão da Vizinhança e de Como Fazer o Endereçamento 4-43
4.4.1.3 Considerações Finais Sobre a Fase de Busca 4-44
4.4.2 Formação de Estruturas Mais Complexas 4-45
4.4.2.1 Construção da Hierarquia 4-45
4.4.2.2 A Importância das Ligações como Comportamento nos Agentes 4-47
4.4.2.3 Considerações Finais sobre a Formação de Estruturas 4-48
4.5 Exemplos 4-49
4.5.1 Um Exemplo Simples Completo 4-49
4.5.1.1 Dicionário 4-50
4.5.1.2 A Fase de Busca 4-51
4.5.1.3 Criando uma Hierarquia 4-53
4.5.2 Topicalização e Posposição do Sujeito 4-54
4.5.3 "*PP-Attachment*" 4-55
4.5.4 Construção Relativa 4-58
4.5.5 Coordenação 4-60
4.5.6 Redução e "*Gapping*" 4-62
4.5.6.1 Reconstrução 4-62
4.5.6.2 Detecção 4-64

4.5.7 Possibilidade de Quebra de Ligações 4-66
4.6 Detecção da Gramaticalidade e da Agramaticalidade de uma Sentença 4-69
4.7 Considerações Finais sobre o Capítulo 4-71
5. Aspectos da Implementação 5-72
5.1 Fase de Inicialização 5-73
5.1.1 Informações dos Agentes 5-73
5.1.2 Informações Extra-Agentes 5-74
5.2 Interação dos Agentes 5-75
5.2.1 Características Gerais do Procedimento "Acorda" 5-75
5.2.2 Implementação da Troca de Mensagens 5-76
5.2.3 Monitoramento do Término 5-78
5.3 Os Procedimentos "Acorda" 5-79
5.3.1 Acorda para Agente Nominal 5-79
5.3.2 Acorda para Agente Verbal 5-81
5.3.3 Acorda para Agente Determinante 5-82
5.3.4 Acorda para Agente Pronome Relativo 5-83
5.3.5 Acorda para Agente Coordenação 5-84
5.4 Considerações Finais sobre o Capítulo 5-85
6. Resultados Obtidos e Problemas Não Resolvidos 6-86
Conclusão 92
Apêndice 94
Referências Bibliográficas 137

Índice de Figuras

Figura 1 — Exemplo de uma rede de discriminação de sentido para a palavra "deep" 2-17
Figura 2 — Representação das ligações estabelecidas entre os agentes da sentença (1). 3-26
Figura 3 — Árvores representando ambigüidade estrutural relativa ao sintagma preposicional (SP). 3-29
Figura 4 — Tipos de associações que os agentes podem ter com as segmentações possíveis para a sentença. 4-37
Figura 5 — Formação de uma estrutura do tipo sintagma nominal usando agregação. 4-46
Figura 6 — Formação de uma estrutura do tipo sintagma nominal usando ligações. 4-46
Figura 7 — Representação da hierarquia, com ligações, para a sentença "o menino caiu ...". 4-47
Figura 8 — Primeiro passo na interpretação de uma sentença. 4-52
Figura 9 — Segundo passo na interpretação de uma sentença. 4-52
Figura 10 — Estrutura hierárquica formada com as ligações dos agentes. 4-53
Figura 11 — Agente preposicional verificando dupla possibilidade de interpretação. 4-57
Figura 12 — Ligação do agente preposicional ao agente verbal como primeira interpretação. 4-57
Figura 13 — Ligação do agente preposicional ao agente nominal como segunda interpretação. 4-58
Figura 14 — Representação da ligação entre um agente pronome relativo e um nominal. 4-60
Figura 15 — Possibilidades de coordenação em sentenças com presença de relativos. 4-61
Figura 16 — Reconstrução com duplicação de ligação. 4-63
Figura 17 — Reconstrução com duplicação de agente. 4-63
Figura 18 — Duplicação de ligações para substituir um agente verbal. 4-64
Figura 19 — Detecção da situação de "gapping". 4-65
Figura 20 — Representação final para a sentença com "gapping". 4-65
Figura 21 — Detecção da situação de redução. 4-66
Figura 22 — Representação final para a sentença com redução. 4-66
Figura 23 — Exemplo de situação que resultará em conflito. 4-67
Figura 24 — Aspecto parcial resultante da quebra de ligações. 4-68
Figura 25 — Situação final após a resolução de alguns conflitos. 4-68
Figura 26 — Esquema geral do processamento. 5-72
Figura 27 — Esquema de encadeamento dos agentes. 5-74

Figura 28 — Esquema de ativação dos agentes..... 5-75
Figura 29 — Simulação de mensagem direta do Agente A para o Agente B..... 5-77
Figura 30 — Simulação de mensagem global..... 5-78

Introdução

É Minas, é Minas. É hora de partir, eu vou.
Vou-me embora pra bem longe.[2]

Este projeto foi iniciado com o intuito de delinear algumas partes da fronteira entre duas sub-áreas da Inteligência Artificial (IA): Processamento de Linguagem Natural (PLN) e Sistemas Multi-Agentes (SMA).

Um dos objetivos da área de PLN é obter uma representação e um processamento em que as ambigüidades inerentes à linguagem possam ser expressas e, se possível, resolvidas pelo uso de diversos tipos de conhecimento. É possível fazer uma divisão em termos do fluxo de processamento da informação no sistema e da segmentação da sentença que será processada[3].

Com relação ao fluxo, é possível diferenciar basicamente dois modos:

- um *modelo de processamento linear*: onde o processamento é dividido em fases e a interação entre elas ocorre apenas pela passagem, em um único sentido, do resultado obtido em uma fase para outra, isto é, o resultado de uma fase não é utilizado por uma fase anterior (não há retorno de informação); e
- um *modelo de processamento interativo*: onde o processamento também pode ser dividido em fases, mas que possuem uma interação mais forte, isto é, o resultado de uma fase pode ser utilizado por uma fase anterior (pode haver retorno de informação).

---

[2] Dory Caymmi e Paulo César Pinheiro na música "Desenredo".

[3] Esses enfoques, além de outros,podem ser encontrados em [Hahn94]

Com relação à segmentação da sentença com que se trabalha, pode-se pensar em três divisões iniciais: divisão por palavra, sintagma e sentença. Quando se inicia o processamento segmentado apenas por *palavras*, o objetivo é conseguir formar estruturas no sentido de obter a interpretação no nível da sentença (como em uma abordagem de baixo para cima — "*bottom-up*"). Quando se inicia o processamento sem segmentação, isto é, a *sentença* por completo, o objetivo é verificar como ela pode ser dividida em estruturas menores cujas partes devem trazer a interpretação do todo (como em uma abordagem de cima para baixo — "*top-down*"). Quando se utiliza segmentação por *sintagmas*, o processamento deve utilizar aspectos tanto da primeira quanto da segunda abordagem.

Diferentes modos de associar os diversos níveis lingüísticos (morfológico, sintático, semântico etc.) podem ser vislumbrados com relação a essas maneiras de fazer a segmentação — refletindo também um pouco como pode ser tratado o fluxo do conhecimento para o PLN.

Um dos objetivos da área de SMA é verificar como as diferentes maneiras de dividir um problema podem ser implementadas utilizando agentes — que podem aqui ser pensados como unidades de processamento autônomas que tentam resolver um problema utilizando as interações entre os pares que compõem o sistema.

Quando se pensa então em unir essas duas áreas para fazer um Sistema de Processamento Distribuído de Linguagem Natural, o que se quer é poder testar as diversas configurações possíveis para o sistema (relativas principalmente ao modo como se fez a segmentação da sentença e ao nível de conhecimento lingüístico utilizado por cada agente, visto que, pela característica de SMA, o fluxo de processamento deverá ser interativo) e observar qual o comportamento dessas diferentes implementações frente a aspectos como desempenho, níveis de conhecimento utilizados em cada configuração, facilidade de implementação etc. Entretanto, como o inter-relacionamento dessas duas áreas (SMA e PLN) é extremamente incipiente, conseguir verificar o comportamento de um sistema em apenas uma dessas configurações possíveis já é um trabalho considerável.

Esta dissertação irá apresentar o desenvolvimento de um sistema cuja configuração utiliza agentes que são associados às palavras de uma sentença, *um agente por palavra.* Cada agente terá seu comportamento ditado pela classe morfossintática da palavra (assume-se que cada palavra pertence a apenas uma classe). De acordo com seu comportamento, cada agente tenta *encontrar* outros agentes com o objetivo de formar estruturas mais complexas e de atingir uma representação que englobe todos os agentes da sentença. Todo esse processo de busca por outro agente é realizado utilizando troca de mensagens entre os agentes. Quando um agente encontra um outro agente que ele considera ideal para formar uma estrutura mais complexa, então esses agentes *estabelecem ligações.* A representação atinge o nível de sentença quando todos os agentes estão conectados.

O objetivo desta dissertação é apresentar o desenvolvimento de um sistema que utiliza características de Sistemas Multi-Agentes (Capítulo 1) em uma abordagem onde o processamento é iniciado a partir da palavra, seguindo algumas características verificadas em outros sistemas que já a utilizaram (Capítulo 2), e tentando resolver alguns problemas lingüísticos (Capítulo 3) que são interessantes por tentar estender algumas características dos modelos vistos no Capítulo 2 e por exercitar algumas características do modelo de processamento do sistema proposto. Este modelo será apresentado por completo no Capítulo 4 e a descrição de sua implementação, no Capítulo 5. Alguns dos problemas encontrados e esboços para suas soluções serão apontados no Capítulo 6, assim como os resultados obtidos com a realização deste projeto. O código completo do programa, que feito em Prolog, está listado no final da dissertação como um apêndice.

1. *Sistemas Multi-Agentes*

Sei que um depende do outro
só pra ser diferente, pra se completar.[4]

A Inteligência Artificial Distribuída (IAD) tem por objetivo estudar maneiras de fazer com que problemas complexos — seja devido à sua distribuição inerentemente geográfica, seja devido à necessidade de utilizar conhecimento de vários domínios diferentes — possam ser resolvidos por meio de sua divisão. Tal divisão implica que, da idéia de um *único agente* anteriormente utilizada em Inteligência Artificial (IA) como a do *ser* responsável pela resolução de problemas, possa-se passar então à outra: a de uma *sociedade de agentes* onde cada agente contribui com uma parcela na solução de um problema de acordo com o seu domínio de conhecimento.

Duas abordagens surgiram com a IAD: *resolução distribuída de problemas*[5]— onde o objetivo é verificar como um problema bem definido pode ser dividido em subproblemas e quais as melhores estratégias de comunicação, coordenação, resolução de conflitos etc. a serem incutidas nos agentes para que estes encontrem a solução do problema; e *sistemas multi-agentes* — onde não se impõe um problema global aos agentes para que eles o resolvam; mesmo que ainda se estudem os temas comunicação, coordenação, resolução de conflitos etc., estes são tratados com um novo enfoque: o que leva um agente a cooperar com outro, como utilizar protocolos de comunicação que permitam que novos agentes entrem no sistema a qualquer hora e, mesmo assim, saibam se comunicar para resolver seus problemas etc.

---

4 Guinga e Aldir Blanc na música "Catavento e Girassol".

O presente capítulo tem por objetivo apresentar um panorama sobre o paradigma Sistemas Multi-Agentes (SMA) da Inteligência Artificial Distribuída para que o trabalho realizado possa ser situado dentro desse contexto. Primeiramente, serão apresentadas duas abordagens para IAD e algumas características utilizadas para diferenciá-las. Uma definição do termo *agente* será apresentada, assim como dois extremos entre os tipos possíveis de agente encontrados na literatura. Algumas áreas de importância (tanto para esse trabalho quanto para a o entendimento de Sistemas Multi-Agentes) serão abordadas tendo em vista esses dois tipos de agentes.

1.1 Resolução Distribuída de Problemas *versus* Sistemas Multi-Agentes

Basicamente existem dois tipos de sistemas em Inteligência Artificial Distribuída: Resolução Distribuída de Problemas (RDP ou "*Distributed Problem Solving*") e Sistemas Multi-Agentes (SMA ou "*Multi-Agent Systems*").

Existem diversas maneiras de caracterizar, dependendo do ângulo pelo qual se analisa, as diferenças entre esses dois tipos de sistemas. Em [Sichman92], parte-se do princípio da autonomia do agente para a caracterização de um sistema como SMA. Assim, um agente em SMA é apresentado como sendo uma entidade que tem existência própria, independente de haver *a priori* algum problema a ser resolvido, enquanto um agente em RDP é criado em função de um determinado problema a ser resolvido.

Em [Durfee94], três visões são apresentadas para tentar delinear as duas abordagens:

1. *RDP como um subconjunto de SMA*: tem-se um RDP quando os seguintes pontos — benevolência, objetivo global comum, projeto centralizado — são assumidos;
2. *SMA provendo um substrato para RDP*: onde o estudo do comportamento dos agentes em termos de SMA pode resultar em propriedades que podem ser utilizadas em uma abordagem RDP na resolução de um determinado problema;

---

[5] Que historicamente foi a primeira.

3. *SMA e RDP como áreas complementares de pesquisa*: devido à impossibilidade de, na maioria das vezes, um observador externo conseguir distinguir entre as duas abordagens, pode-se partir do pressuposto que as duas áreas são complementares e que, na realidade, diferenciam-se pelas motivações científicas e técnicas empregadas.

Com relação ao sistema que irá ser apresentado nesta dissertação, se se pensar em termos do ponto de partida da pesquisa — fazer um sistema de Processamento Distribuído de Linguagem Natural do seguinte modo: (1) pensando no problema, (2) criando os agentes e (3) obtendo a solução, este pode parecer um RDP (cf. [Sichman92]). Entretanto, pensando-se que Processamento de Linguagem Natural (PLN) é uma área em que vários de seus problemas estão em aberto, então uma tentativa de utilizar agentes para buscar soluções neste domínio é, para a IAD, uma maneira de experimentar com os comportamentos possíveis e/ou necessários dos agentes no sentido de que essas propostas de soluções sejam factíveis.

Pensando-se em termos de uma experimentação, que possui ligação com o item 3 de [Durfee94] citado acima, o sistema poderá ser considerado como um Sistema Multi-Agentes. Outras considerações como a não existência de um objetivo global embutido nos agentes e o não compartilhamento de informações também corroboram esta classificação.

1.2 Tipos de Agentes

Uma *agente* é uma entidade que possui como características gerais (ver, entre outros, [Sichman92], [Sichman95], [Demazeau90b], [Demazeau91], [Bussmann94]):

- estar inserido em um *ambiente* — do qual pode manter uma representação;
- ter a capacidade de *percepção* do que ocorre no ambiente;
- ter a capacidade de *comunicação* com outros agentes da sociedade;
- ter a capacidade de *realização* de determinadas tarefas;
- ter a capacidade de *raciocínio* para realizar suas tarefas e para inferir coisas sobre os outros agentes;
- possuir um comportamento *autônomo*, no sentido de o agente, em última instância, deci-

dir por si próprio o que deverá realizar em um determinado instante.

Estas características são variáveis de acordo com o tipo de agente com que se está trabalhando. Há dois extremos para a classificação de tipos de agentes: reativos e cognitivos.

A idéia do *modelo reativo* é especificar um agente pelo seu comportamento, ou seja, pelo modo como ele reage a determinados estímulos do ambiente [Drogoul94]. Tal agente não armazena fatos do passado para utilizá-los no presente/futuro, não possui noção de si mesmo e dos outros agentes da sociedade e, por isso mesmo, não tem necessidade de usar mecanismos sofisticados de comunicação. Há duas motivações para utilizar tal tipo de agente: produzir comportamento emergente a partir de agentes muito simples (ou seja, a tarefa local do agente é simples, mas o resultado global é complexo, como em [Steels90]), ou permitir comportamentos rápidos frente às mudanças no ambiente (como em [Wavish95]).

A idéia do *modelo cognitivo* é utilizar a representação simbólica de conhecimento do agente para operações em um nível mais complexo. Assim, técnicas tradicionais de Inteligência Artificial são adaptadas para o raciocínio não só sobre o próprio agente, mas também para o dos outros agentes que formam a sociedade. Tal agente pode utilizar comunicação estruturada e formalismos lógicos sofisticados que lidam com *intenções*, *crenças* e *objetivos*[6], como os de [Cohen87] ou de [Rao91]. Um exemplo de estrutura para agentes cognitivos pode ser encontrada em [Demazeau90b]. A Tabela 1 resume e compara as principais características dos dois tipos de agentes.

---

[6] Um exemplo de sistema que utiliza tais agentes, normalmente chamados de "*BDI-Agents*" ("*Belief-Desire-Intention*"), pode ser visto em [Rao95].

Agentes Cognitivos	*Agentes Reativos*
• conhecimento representado de forma explícita	• conhecimento representado de forma implícita
• utilizam memória dos fatos acontecidos	• não utilizam memória
• sociedade com poucos agentes	• sociedade com muitos agentes
• controle deliberativo	• controle instintivo
• programação mais complexa	• programação mais simples

Tabela 1 — Comparação entre agentes cognitivos e reativos.

O *modelo misto* proposto em [Bussmann94] tenta obter um equilíbrio entre os dois modelos, com os seguintes requisitos:

- um agente tem que ser capaz de *reagir* a eventos imprevistos;
- um agente tem que ser capaz de *adaptar* a velocidade do seu comportamento de acordo com a evolução do sistema/ambiente;
- um agente tem que ser capaz de *manipular* de forma explícita o conhecimento sobre o universo/ambiente.

Naturalmente, esses requisitos são válidos para agentes que se encontram em ambientes dinâmicos (como robôs), não implicando que todo tipo de agente misto necessite seguir as características acima apontadas.

1.3 Tópicos de Interesse

Após as primeiras considerações sobre SMA, é interessante tentar mapear o tipo de agente de acordo com algumas outras características, quais sejam: modelo de ação, de comunicação, de

resolução de conflitos, e de sociedade. Assim, para cada um desses eixos de análise citados, alguns trabalhos/autores serão apresentados como fonte de exemplos.

1.3.1 Ação

Por *modelo de ação*, entende-se o modo como uma ação pode ser concebida tendo em vista as características do agente. Para agentes reativos, tem-se dois esquemas: *estímulo-reação* (cf. [Drogoul94]), onde não há nenhum "raciocínio" entre o estímulo (que pode ser considerado como um estado particular do ambiente) e a reação (que pode ser vista como uma seqüência de ações básicas); *ação situada*, onde os planos são vistos como uma seqüência de ações que sofre improvisos de tempos em tempos de acordo com a situação do ambiente [Wavish95].

Um modelo de ação (estímulo-reação) para agentes reativos que é de interesse é o do sistema PACO [Demazeau93a]. Nesse sistema as interações são modeladas sob a forma de *forças* que restringem o deslocamento dos agentes em um ambiente. Essas restrições são baseadas nas noções de *campo de comunicação* (que guia as interações entre os agentes), *campo de percepção* (que restringe as interações entre os agentes e o ambiente) e *campo de ação* (que restringe as possibilidades de deslocamento dos agentes no ambiente).

Para agentes cognitivos, o esquema característico também envolve *planos*, só que, devido ao processo de ação ser mais elaborado, esse planos também devem sê-lo. O enfoque de estudo também pode mudar. Por exemplo, [Martial90] propõe que um agente deve analisar os planos de outros agentes e assim verificar a existência de conflitos benéficos (que ele chama de "positivos") ou maléficos ("negativos"). Isso pode levar o agente a reconsiderar suas ações ou, pelo menos, a ordem em que elas devem ser executadas, no sentido de evitar os conflitos negativos.

1.3.2 Comunicação

Por *modelo de comunicação* entende-se o modo como o agente interage com outro agente no ambiente. A comunicação entre agentes cognitivos é mais estruturada do que a utilizada por agen-

tes reativos. Em [Demazeau93b], o objetivo é mapear, em termos de redes de transição de estados, uma estrutura que permita ao agente saber como reagir a uma mensagem e também o que esperar de uma mensagem. Em [Campbell90], o objetivo é fazer uma taxonomia de "tons" de comunicação que os agentes poderão utilizar.

Um protocolo que será de interesse é o da *rede de contrato* [Davis83] que utiliza a metáfora "comercial" para fazer alocação de tarefas (que foi a aplicação utilizada pelos autores). Ele funciona da seguinte maneira: um agente (*contratante*) anuncia para todos os outros agentes que possui uma tarefa com determinados requisitos; cada agente verifica suas possibilidades (em termos de suas habilidades, sua carga, seu desempenho etc.) e envia uma oferta; o agente contratante, a partir das ofertas que recebeu, escolhe o melhor agente para executar a tarefa.

Para agentes puramente reativos, normalmente a comunicação é feita de modo indireto por mecanismos de sinalização (espalhar marcas, por exemplo) e de percepção, como as *agressões* utilizadas pelos agentes em "Eco-Problem-Solving" [Drogoul92 e Ferber91] — onde um agente "agride" um outro no sentido de tentar tomar-lhe a posição, ou como as *assinaturas* do "EthoModeling Framework" [Drogoul94] — onde um agente espalha marcas pelo ambiente que podem ser interpretadas de diferentes formas, dependendo de quem as percebe[7].

1.3.3 Cooperação e Resolução de Conflitos

Por cooperação e resolução de conflitos, entende-se o modo como um determinado tipo de agente interage com os outros agentes e, caso haja interesses conflitantes, como é que eles os superam.

---

[7] "*Agents can communicate in a very indirect manner by propagating their signature(s) in the environment. (...) For instance, the signature of an ant will activate a friendly behavior from another ant and an aggressive one from a predator*" ([Drogoul94], página 9).

Em arquiteturas reativas, devido ao fato de os agentes normalmente não possuírem conhecimento explícito sobre outros agentes, a cooperação e a resolução de conflitos são geralmente embutidas como um comportamento. Assim, não existe propriamente uma etapa de resolução de conflitos. Com relação a conflitos internos ao agente (isto é, conflitos entre comportamentos), normalmente se utilizam níveis de prioridade, em que um determinado comportamento possui precedência sobre outro [Steels90].

Em arquiteturas cognitivas, normalmente se utiliza negociação entre agentes no sentido de tentar evitar situações de conflito. Werner, em [Werner90], salienta que, para uma melhor cooperação, é preciso que o agente tenha em sua representação o modelo de cooperação utilizado pelos outros agentes da sociedade.

Mapear as dependências que um agente possui com relação a outros agentes da sociedade referentes a determinados planos ou ações [Sichman94] *pode ser* uma maneira de um agente instanciar um modelo de cooperação ou de negociação com outros agentes da sociedade.

[Galliers90, p. 34] advoga que os sistemas que expressam conflitos e os resolvem sem impor benevolência aos agentes poderão ser suficientemente flexíveis para lidar com eventos imprevistos, que é uma característica importante se os agentes forem interagir em um ambiente aberto, onde agentes podem entrar e sair a qualquer momento.

1.3.4 Sociedade

Por *modelo de sociedade*, entende-se o tipo de estruturação utilizado por um determinado grupo de agentes em uma determinada situação. Os modelos reativos geralmente tendem a seguir modelos biológicos (como colônias de formigas, [Drogoul94] e [Steels90]). O enfoque então é observar como a emergência de comportamentos se dá pelo comportamento global da sociedade, e não pelo comportamento isolado dos indivíduos da sociedade. Os modelos cognitivos tendem a seguir modelos humanos e o enfoque é pesquisar o porquê de um agente agir ou não de um determinado modo, ou qual o resultado de ele agir de uma forma ou de outra.

1.4 Considerações Finais sobre o Capítulo

Esta pesquisa se iniciou sem a pressuposição do que seria um agente para o Processamento Distribuído de Linguagem Natural: as características obtidas são *o resultado* da pesquisa e poderão ser vistas no Capítulo 6.

Na construção dos agentes lingüísticos, levou-se em consideração as possibilidades de segmentação de uma sentença (cf. seção 4.2, à página 4-35) e o nível de conhecimento lingüístico que deveria ser associado aos agentes. Escolheu-se então uma configuração dessas duas características (tipo de segmentação e nível lingüístico) no intuito de *experimentar* as capacidades de um modelo que levasse em consideração algumas das características de Sistemas Multi-Agentes, entre as principais: a capacidade de resolução de problemas pelo inter-relacionamento (cooperação usando comunicação) dos agentes e a idéia de independência de cada agente.

Com a idéia de divisão estabelecida, alguns sistemas que possuíam certa semelhança, mas que *não foram criados* com a idéia de SMA, foram analisados. Esses sistemas serão apresentados no próximo capítulo e serão apontadas algumas direções para a extensão dos mesmos.

2. *Considerações sobre Processamento Lingüístico I*

Apresentação de alguns sistemas

Chega mais perto e contempla as palavras
Cada uma tem mil faces secretas sob a face neutra.[8]

No decorrer dos anos 80, uma grande soma de esforços em Processamento de Linguagem Natural (PLN) foi despendida na busca de soluções para o problema da ambigüidade léxica — isto é, a determinação do significado de uma palavra em um contexto quando ela, extra-contextualmente, pode ter vários. Alguns desses empreendimentos se situam no que se denomina *sistemas baseados na palavra* (do inglês "*word-based systems*") cuja idéia principal é que pequenos fragmentos de informação (no caso, palavras e/ou morfemas) podem, em um processo de interação onde são trocadas informações, não apenas verificar qual o sentido de uma palavra em um determinado contexto, mas também obter o significado da sentença como um todo.

Essa idéia foi originalmente apresentada por Steven Small [Small80] e teve influência em outros trabalhos como [Hirst87], [Hirst88], [Adriaens86], [Devos88] e [Devos94], assim como no presente trabalho. Este capítulo irá apresentar as idéias utilizadas pelos autores citados e apontar como elas poderão ser situadas em um contexto de Sistemas Multi-Agentes.

---

[8] Carlos Drummond de Andrade em "Procura da Poesia" à página 95 do livro POESIA E PROSA, editora Nova Aguilar, 8ª edição, 1992.

2.1 "*Word Expert Parser — WEP*"

A motivação apresentada por [Small80] para considerar a idéia da palavra como elemento central na interpretação de uma sentença vem da observação da grande quantidade de sentidos que uma palavra pode expressar e também da facilidade com que as pessoas os escolhem quando utilizando a linguagem [Small80, p. 27], assim como da dificuldade dos sistemas da época em lidar com isso utilizando regras gerais.

A idéia proposta é um tanto quanto radical:

> *"Individual words do not constitute the input to some mechanism which analyzes sentences from above according to some general external principles about language. Rather, the words themselves form the mechanism*[9]*."* [Small80, p. 1]

Assim, cada palavra da língua, segundo Small, deve ser vista como um agente[10] léxico, chamado "*word-expert*", que participa do controle global do processo de análise ("*parsing process*") com suas *ações internas* e suas *interações* com os outros agentes da sentença (isto é, os outros "*word experts*"), assim como com os mecanismos que contêm conhecimento do discurso e do mundo.

2.1.1 Interações entre Agentes

Os agentes são postos em funcionamento de acordo com a ordem em que aparecem na sentença. Assim, um a um, os agentes são executados e trabalham o máximo que conseguem até que um dos três seguintes estados é alcançado:

---

[9] Grifos do autor da presente dissertação.

[10] Deve ser observado que "agente", no contexto deste capítulo, não necessariamente possui o mesmo sentido de um "agente" em Sistemas Multi-Agentes.

1. o agente *completou sua tarefa* a ponto de conseguir determinar um sentido único que ele pode ter, dado seu contexto atual;
2. o agente já determinou um sentido único, mas pode esperar que uma informação adicional apareça para *refinar ainda mais seu significado*;
3. o agente *necessita de mais informação* para prosseguir.

No primeiro caso, o agente *termina* sua execução e *não* será ativado novamente. No segundo, o agente termina sua execução, mas poderá ser acordado no futuro. No terceiro, o agente *suspende* sua execução para ser acordado no futuro.

Antes de suspender sua execução, o agente envia uma mensagem com um *pacote de informações* para a memória global[11] em que ele especifica:

- o *tipo de dado* que está esperando — isto é, qual tipo de informação que o poderá levar a acordar novamente;
- uma *expectativa* — ou seja, uma preferência com relação ao melhor tipo de dado esperado (item anterior);
- um *tempo máximo* — indicando o quanto ele deve esperar por uma informação;
- um *ponto de continuação* — indicando qual a próxima tarefa a ser executada tanto no caso de a informação esperada chegar, quanto no caso de o tempo chegar ao limite especificado pelo tempo máximo (item anterior).

Associado a cada pacote de pedido de informações é colocado um monitor ("*demon*") que fica observando as mensagens que passam pela memória para poder reativar seu agente — no caso de alguma informação atender a uma expectativa.

Além desse tipo de mensagem (*pacote de informações*), os agentes também trocam *sinais de controle* que são utilizados para caracterizar determinados aspectos do processamento (como, por exemplo, identificar a possibilidade de a sentença estar na voz passiva) a partir do nível local do agente. Nas palavras de Small:

> *"The role of syntax as providing structural cues for understanding language corresponds to the sending of control signals between experts in the Word Expert Parser."*[Small80, p. 61]

Além das interações entre si, os agentes interagem com um mecanismo que contém conhecimento do discurso e de mundo (pragmático), e que provê medidas quantitativas para verificar a associação entre palavras[12].

2.1.2 Ações dos Agentes

Quanto ao funcionamento interno dos agentes, este pode ser visto como o processo de percorrer uma *rede de discriminação de sentidos* (ou "*sense discrimination network*") — que consiste em módulos com questões para o refinamento de um conceito. A Figura 1 representa uma rede para a palavra "*deep*" [Small80, p. 48].

Neste pequeno fragmento, "*entity-construction*" é um sinal de controle que é enviado por um determinante (por exemplo, o agente "*the*"). Assim, o agente "*deep*" tentará verificar se ele é um adjetivo ou um substantivo, analisando a próxima palavra. Se for um substantivo, então ela participará na formação do grupo (que representa um conceito) e, assim, "*deep*" será um adjetivo. Caso contrário, "*deep*" será um substantivo (por exemplo, se a palavra seguinte for um verbo ou se não houver palavra seguinte). O lado esquerdo da figura (a parte do "*yes*") representa exatamente a continuação para o caso de o agente ser um adjetivo (por exemplo, na sentença "*the deep philosopher likes existentialism*") e a parte direita, para o caso de ele ser um substantivo (por exemplo, na sentença "*the giant squid lives in the deep*").

---

[11] É utilizado um quadro-negro (*blackboard*), cf. [Engelmore88].

[12] Apesar de esse mecanismo não ter sido implementado, sua interface com os agentes o foi, permitindo que certas delimitações do sistema fossem testadas.

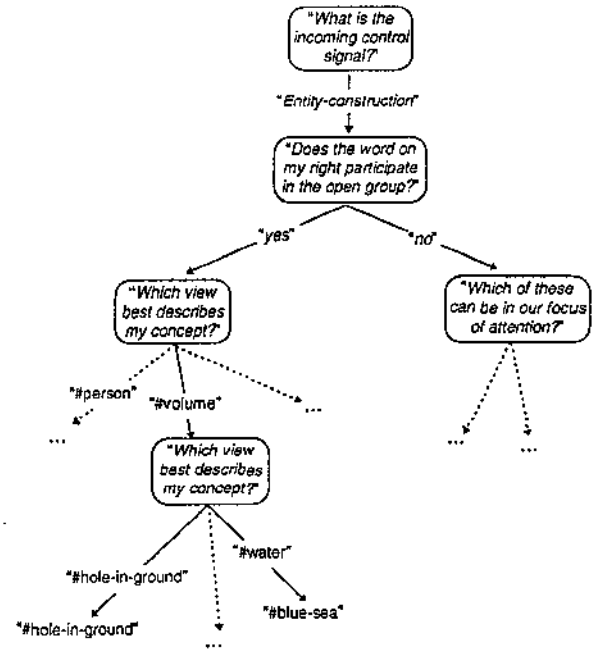

Figura 1 — Exemplo de uma rede de discriminação de sentido para a palavra "*deep*".

Assim, cada agente é um procedimento que implementa uma rede de discriminação de sentidos e o processo de caminhamento por essa rede, entremeado com partes para envio de mensagens, sinais de controle e inquisições ao mecanismo de discurso e de mundo — representando, então, as *ações* e as *interações* (passagem do controle para outros agentes) que o agente pode realizar.

2.1.3 Considerações Finais Sobre WEP

O sistema WEP de Small faz o tratamento de sentenças relativamente simples[13] em termos de estrutura, principalmente porque a ênfase está no processo da resolução da ambigüidade das palavras. Em [Adriaens86], o sistema foi estendido com vistas à interpretação do holandês e os seguintes tipos de sentenças foram tratados: interrogativas, imperativas e declarativas. Além

[13] O caso mais complicado é o tratamento de voz passiva.

disso, sentenças que contêm constituintes descontínuos[14] puderam testar a tática de *expectativa* e *espera* utilizada pelo modelo.

A idéia mais problemática do WEP (também apontada por [Hirst87, p. 83]) é a de que não é possível, segundo Small, construir processos genéricos para o tratamento das palavras — o que faz com que *toda palavra* tenha que ser *literalmente programada* como um procedimento específico que prevê todas suas possíveis interações com outras palavras![15]

O processo de comunicação utilizado pelos agentes é sempre via memória centralizada[16], o que, juntamente com o sistema de pilha ("*stack*") utilizado no armazenamento das mensagens, impede que o sistema (da maneira que foi concebido) possa ser executado em paralelo [Devos88, p. 143].

2.2 "*Polaroid Words*" e a Resolução de Ambigüidade Semântica

Graeme Hirst em sua tese de doutorado (que posteriormente foi publicada em [Hirst87] e [Hirst88]) aborda tanto o problema da ambigüidade (estrutural e semântica) quanto o de uma interpretação semântica (ou seja, de conseguir fazer um mapeamento formal entre objetos sintáticos e seus correspondentes semânticos). O interesse dessa dissertação reside no modo como ele consegue resolver ambigüidade semântica com suas "*polaroid words*", como será visto mais adiante.

---

[14] Um exemplo é o verbo "*opbellen*" (ou, em inglês, "*to call up*") que pode aparecer em uma sentença da seguinte maneira: "*Geert belt Hilde op maandag op*" / "*Geert calls up Hilde on Monday*" [Adriaens86, p. 228].

[15] Para se ter uma idéia da dificuldade que isso implica, o procedimento utilizado por Small para a palavra "*throw*" ocupa seis páginas!

[16] Como se todas as mensagens passassem por um único duto que é escutado por todos os agentes.

Hirst separa a análise sintática da semântica, mas não impede que a primeira colabore com a segunda (por exemplo, para a resolução de ambigüidade estrutural). Seu analisador sintático (chamado "Paragram") funciona *in tandem*[17] com o analisador semântico (chamado "Absity"). Isto é, quando o "Paragram" possui uma estrutura sintática bem formada (por exemplo, um sintagma nominal), ele a passa para o "Absity" que irá então construir a representação semântica para tal estrutura.

O processo (tanto o sintático quanto o semântico) é composicional, no sentido de o todo ser a soma dos pedaços, e de haver uma formalização para a regra de composição desses pedaços. Tal formalização *assegura* que, dados três objetos sintáticos A, B e C que possuem representações semânticas D, E e F respectivamente, se A em composição com B resulta em C, então D em composição com E *resultará* em F. O que o "Absity" faz, na realidade, é seguir as regras de composição estabelecidas com relação à sua representação semântica (cujos objetos são representados como *frames*[18], *slots* e *instances* [Charniak81]).

2.2.1 O Problema da Ambigüidade

O processo de interpretação semântica do "Absity" seria direto caso não houvesse ambigüidade semântica em decorrência de uma palavra poder ter vários significados. O problema da ambigüidade consiste em, dado um objeto sintático que o "Paragram" envia para o "Absity", poder mapeá-lo para vários objetos semânticos, contrariando a regra de composição apresentada.

A solução para esse problema é a utilização de "*polaroid words*" como um objeto semântico diferente que pode esconder a *não existência* de um objeto semântico que possua sentido único.

---

17 Isto é, em seqüência.

18 Uma definição informal para esse objetos pode ser encontrada em [Hirst87] à página 13: "*We can think of a generic frame as a representation of a concept and an instance as one specific occurrence of the concept. Slots can be thought of as arguments or parameters of the concept, so that the filled slots of an instance describe or qualify that instance*".

A metáfora é a de uma foto "*polaroid*" que possui forma (objeto semântico), mas cujo conteúdo ainda não está completo e vai sendo revelado aos poucos. Essa revelação é, na realidade, o processo de resolução da ambigüidade da palavra, que será visto na próxima seção.

2.2.2 Utilizando *Polaroid Words*

Há duas maneiras de caracterizar uma "*Polaroid Word*" (PW): como um objeto semântico e como um processo que faz a resolução da ambigüidade desse objeto semântico. Como objeto semântico ela é capaz de representar as ambigüidades de sentido que uma palavra pode ter. Como processo ela é encarregada de obter com o "Paragram" (o analisador sintático) um objeto sintático do nível de palavra (substantivo, verbo, adjetivo etc.) e passar como resultado para o "Absity" (o analisador semântico) o objeto semântico final (sem ambigüidade). A representação das PWs é diferenciada com relação a todas as categorias sintáticas (ou seja, existem PWs do tipo substantivo, do tipo verbo etc.) e o processo de resolução de ambigüidade é ajustado com relação a essas categorias.

As possibilidades de trabalho de uma PW estão relacionadas à existência ou não de ambigüidade. Se a palavra não é ambígua, então a PW passa o objeto semântico (único) para o "Absity" e termina sua execução. Se a palavra é ambígua, então existem duas possibilidades.

A primeira acontece quando o "*Marker Passer*" (MP)[19], que vem analisando — à medida que as palavras são lidas pelo "Paragram" — os sentidos possíveis das palavras da sentença, encontrou apenas um sentido para a palavra associada à PW corrente. Assim, essa PW anuncia esse significado ao "Absity" e suspende sua execução.

A segunda acontece quando o MP não chegou a uma possibilidade conclusiva e, assim, a PW terá que encontrar algo no contexto em que sua palavra está situada para ver qual alternativa de sentido é a mais adequada. Para fazer isso, ela verifica as PWs anteriores na sentença

---

[19] Mecanismo utilizado para verificar se existe algum tipo de associação entre duas palavras.

para ver se elas podem fornecer alguma informação para a resolução do seu sentido. Usando as informações recolhidas, a PW irá eliminar tantas alternativas quantas forem possíveis. Se conseguir chegar a apenas uma possibilidade, irá anunciá-la ao "Absity" e terminar. Senão, irá anunciar as possibilidades que restaram e suspenderá sua execução.

Esse processo de anunciação é realizado com a PW colocando uma lista com suas possibilidades de sentido em uma memória global. A comunicação entre as PWs é restrita a essa verificação das possibilidades restantes associada ao fato de que elas só podem observar os dados das PWs anteriores e que estas têm de ser suas "amigas"[20].

Um exemplo de eliminação de possibilidades acontece quando uma PW anuncia que está ocupando o papel temático (cf. [Allen95], p. 244) de AGENTE em um determinado verbo. Quando isso acontece, as outras PWs poderão eliminar tal alternativa de sua lista, haja vista que um verbo pode ter apenas um AGENTE. Outro exemplo acontece quando uma PW verifica as possibilidades de complementação (em termos semânticos) que um verbo exige e elimina as alternativas que não satisfazem tal exigência.

Ao final de um ciclo de processamento de uma PW, ou ela terá chegado a um único significado e terminado, ou ela terá chegado a mais de um e, então, irá anunciar suas novas possibilidades e suspenderá sua execução. Isso fará com que uma de suas amigas que ainda não tenham terminado seja acordada para ver se as novas informações podem ajudá-la no seu processo de resolução de sua ambigüidade. Esse processo se estende até que todas as PWs tenham chegado a um significado único.

---

[20] A noção de "amizade" não foi formalmente definida por Hirst, mas especifica que "*verbs are friends with the prepositions and nouns they dominate; prepositions are friends with the nouns of their prepositional phrase and with other prepositions; and noun modifiers are friends with the noun they modify*" [Hirst87, p. 107].

2.3 "*Parallel Expert Parser — PEP*"

Este trabalho foi a continuação dada pelo grupo de Geert Adriaens para o "*Word Expert Parser*" e concentrou-se na tentativa de uma implementação do sistema em um ambiente de programação paralela [Devos88, Devos94]. Sua descrição será feita de modo um tanto quanto sucinto.

As revisões principais dizem respeito:

- *à especificação de uma linguagem para descrição dos agentes e suas funções de forma declarativa*: isso permite que um lingüista a utilize para a confecção dos agentes, ao invés de ter que programar em Lisp como era o caso do WEP;
- *à utilização de agentes para representar outros níveis além do nível da palavra, como o sintagmático e o sentencial*: isso possibilita, segundo os autores, uma melhora na modularidade dos agentes;
- *à possibilidade de um agente poder se comunicar diretamente com outro*[1]: o que permite manter a comunicação sob controle, não sendo um gargalo para o sistema; e
- *à utilização de uma estrutura de representação do conhecimento do agente com a divisão da parte sintática, semântica e léxica.*

O processo de interpretação se dá de uma forma de baixo para cima ("*bottom-up*"). Os agentes no nível de palavra (por exemplo, artigos, substantivos, adjetivos etc.) são inicializados e trocam informações a fim de refinar seus conceitos e atingir níveis mais elevados (como ação, agente, tema etc., até o nível de sentença).

Quando os agentes conseguem agregar seus conhecimentos de maneira a formar um conceito de um nível superior, um novo agente (correspondente a esse novo nível) é criado e inicializado com as informações dos agentes do nível inferior que, então, são desativados. Quando o nível de sentença é atingido, o sistema é parado.

---

[1] A comunicação no WEP era através de "dutos de comunicação", onde todos os agentes podiam escutar.

No nível de representação dos agentes o sistema fez significativos progressos; no tocante à análise de sentenças com outros fenômenos lingüísticos não tratados pelo WEP, pode-se dizer que o avanço foi nulo, o que de maneira nenhuma desmerece o que foi realizado. O prosseguimento dado a esse trabalho (chamado de PEP2, [Bourgois91]) seguiu outra linha de pesquisa (baseado em modelos conexionistas) e sua descrição não será de utilidade para o presente trabalho.

2.4 Considerações Finais Sobre o Capítulo

É interessante notar que existe uma direção a ser seguida por um *sistema baseado na palavra* que pode ser derivada das seguintes observações:

- É possível, como visto na seção 2.2, utilizar conhecimento generalizado para a resolução de ambigüidade semântica, assim como para a resolução de ambigüidade estrutural (cf. [Hirst87], Capítulos 6 e 7, não apresentados nesta dissertação);
- Formas sintáticas complexas não foram suficientemente tratadas no *Word Expert Parser*, o que sugere novas investigações, embora a forma de abordagem por palavras do WEP não seja interessante pelas razões expostas na seção 2.1.3.

O resultado disso é que fica extremamente interessante tentar estender a idéia de *Polaroid Words* (não só a do objeto que ainda não tem forma final, mas a do processo baseado nas classes de palavras que elas representam) para o tratamento tanto da parte semântica como da parte sintática das sentenças (formação das estruturas sintáticas), visando ao tratamento de sentenças que apresentam estruturas sintáticas (que serão vistas no Capítulo 3) não tratadas pelo WEP.

Assim, a idéia de agentes lingüísticos que será apresentada no Capítulo 4 seguirá esta linha, isto é, será uma abordagem centrada nas funções das classes de palavras, onde cada agente terá o comportamento da classe em que sua palavra está inserida. Os agentes terão que realizar associações no sentido de formar estruturas mais complexas que representarão, no final, o significado da sentença. Será visto que esse processo de associação entre agentes deverá ser precedido por uma busca dos agentes mediada por um determinado número de restrições.

3. *Considerações sobre Processamento Lingüístico II*

Apresentação de alguns fenômenos lingüísticos

"O cérebro eletrônico faz tudo faz quase tudo faz quase tudo mas ele é mudo".[22]

Este capítulo irá apresentar alguns fenômenos lingüísticos e mostrar o que eles irão exercitar na arquitetura do sistema (que será apresentada de forma completa no próximo capítulo). Seu conteúdo será basicamente expositório, ou seja, para cada problema serão apresentados exemplos de sentenças onde eles aparecem.

A escolha dos fenômenos que irão ser aqui retratados seguiu a forma como a implementação das idéias do sistema foram construídas — partiu-se de um esquema sentença bem simples e cada próximo fenômeno foi introduzido com a idéia de deixar o sistema mais flexível. Por exemplo, quando se percebeu que a busca das subcategorizações por parte dos agentes verbais impedia problemas de inversão de posicionamento ("topicalização" e "posposição de sujeito" — ver seções 3.2.1 e 3.2.2 à frente), então uma solução para estes fenômenos foi pensada. A idéia principal era chegar a um sistema que fosse capaz de lidar com casos em que o objeto mais importante para o processamento da sentença — ou seja, a *palavra* — estivesse ausente (ver "reduções" e "*gapping*" nas seções 3.2.6 e 3.2.7).

O objetivo deste capítulo será, então, verificar quais operações terão que ser introduzidas para, partindo-se de uma arquitetura básica que lida com sentenças estruturadas em uma forma

simples, chegar-se a uma arquitetura que permita o tratamento dos diversos tipos de fenômenos lingüísticos.

O esquema do capítulo será o seguinte. Um modelo simplificado para o tratamento de orações simples que já utiliza a idéia de agentes por classes de palavras será apresentado e, à medida que os fenômenos lingüísticos forem apresentados, as características que forçaram um avanço neste modelo simples serão comentadas.

3.1 Um Esboço de Processamento para Orações Simples

Um modelo de processamento simples neste contexto *pode ser* aquele em que as sentenças possuam apenas alguns elementos simples em suas posições sintáticas padrões, por exemplo, *sujeito*, *predicado* e *objeto direto*, nesta ordem. Para processar tal tipo de sentença em um modelo de processamento semelhante ao que foi aventado no final do capítulo anterior (um agente por classe de palavra), é necessário fazer com que os agentes que representam as palavras da sentença se encontrem e formem associações que, em última instância, resultarão na interpretação da sentença.

Para uma oração como

(1) O menino matou o gato.

haverá, então, agentes associados às palavras da seguinte maneira: um agente determinante (ag_1) associado à palavra o; um agente nominal (ag_2) associado à palavra menino; um agente verbal (ag_3) associado à palavra matou; um agente determinante (ag_4) associado à palavra o e, finalmente, um agente nominal (ag_5) associado à palavra gato. O processo de encontro dos agentes no sentido de formar associações será estabelecido com cada agente realizando uma busca pelo agente ao qual ele deve se associar.

---

[22] Gilberto Gil na música "Cérebro Eletrônico".

Para cada classe de palavra é possível estabelecer padrões que modelam seu comportamento e, por conseguinte, o comportamento do agente. Assim, agentes verbais buscam por agentes nominais para se associarem e o mesmo ocorre com os agentes determinantes em relação aos agentes nominais. Em suas buscas, cada um leva em consideração a posição em que os outros agentes devem ser encontrados. Por exemplo, o agente determinante irá procurar seu nominal em uma posição à sua direita. Já o agente verbal levará em consideração as posições sintáticas necessárias para seu preenchimento[13]: se for sujeito, ele procurará à sua esquerda e, se for objeto direto, à sua direita.

Os agentes trocarão mensagens no sentido de se conhecerem e checarem certas restrições (por exemplo, se o agente determinante tiver instanciada uma palavra no singular, ele só poderá se associar a um agente nominal cuja palavra também esteja no singular). Quando todos os agentes estiverem com suas associações estabelecidas, então uma solução terá sido encontrada. Uma representação para os agentes já associados pode ser vista na Figura 2, onde as setas indicam a estrutura hierárquica formada, ou seja, o ag_1 é subordinado ao ag_2 e o ag_4 é subordinado ao ag_5. Os agentes ag_2 e ag_5 são subordinados ao ag_3.

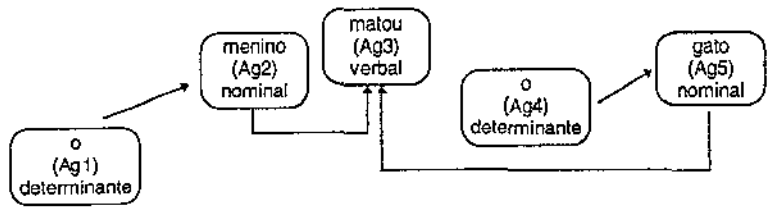

Figura 2 — Representação das ligações estabelecidas entre os agentes da sentença (1).

Assim, para a análise de sentenças simples como a apresentada, basta que o agente olhe para sua vizinhança próxima para encontrar o(s) vizinho(s) com quem deverá se associar, troque informações para verificar se o vizinho é o ideal e, em caso positivo, estabeleça as ligações com

---

[13] Aqui entram questões como subcategorização e restrição semântica, que serão abordadas apenas no próximo capítulo.

ele. Alguns problemas surgirão para esse tipo de análise quando se tentar interpretar sentenças um pouco mais complexas, como as que serão vistas logo a seguir.

3.2 Exemplos de Fenômenos Lingüísticos

Nesta seção serão apresentados os seguintes fenômenos lingüísticos: *topicalização, posposição de sujeito, "pp-attachment", construção relativa, coordenação, redução* e *"gapping"*.

3.2.1 Topicalização

Topicalização pode ser entendida como a mudança de um termo para o início da oração. [Perini96] define topicalização baseando-se na noção de correspondência total que estabelece quando duas orações são, por assim dizer, idênticas. Assim "sempre que em uma oração houver um termo marcado [+ant][24], haverá também uma oração idêntica a essa, com a diferença de que o termo em questão está no início da frase" [Perini96, p. 213]. O objeto direto é um exemplo de termo marcado como [+ant]. Assim, para a oração

(2) Leo comeu o peixinho.

haverá uma outra oração que será considerada idêntica a ela, que é:

(3) O peixinho, Leo comeu.

Não apenas o objeto direto pode ser topicalizado, mas também outros termos como o complemento do predicado, que pode ser visto nos seguintes exemplos:

(4) Leo está irritado.

(5) Irritado, Leo está.

---

[24] Onde [+ant] significa "passível de anteposição".

A motivação para tratar tal tipo de problema é o fato de que certos termos podem estar fora de sua posição usual na oração e, mesmo assim, têm que ser encontrados. Dessa maneira, certos tipos de agentes têm que estar preparados para procurar em outras direções caso não encontrem o que desejam. Este é o caso do agente verbal quando procura por um agente nominal para a posição de objeto direto, mas já não é o caso do agente determinante quando procurando por um agente nominal ao qual deve se associar.

3.2.2 Posposição de Sujeito

Normalmente o sujeito da oração vem antes do núcleo do predicado, mas há situações em que ele pode vir após o núcleo do predicado, como nas seguintes orações, onde está sublinhado:

(6) Caiu o menino.

(7) Onde está o menino?

(8) Chegaram vários carregamentos de cachaça.

[Perini96, p. 221] cita que "os fatores que governam a posposição do sujeito são ainda misteriosos e que, embora vários fatores sintáticos, léxicos, semânticos e até discursivos tenham sido propostos, ainda não se chegou a uma solução cabal".

Existem, de qualquer modo, restrições à posposição do sujeito. Uma delas é que ela não pode ocorrer se a oração possuir objeto direto, como na seguinte sentença

(9) * Comeu uma pizza o menino

ou, mesmo em orações sem o objeto direto, se o verbo for impeditivo, como na seguinte sentença

(10) * Ronca meu primo favorito

em contraposição a

(11) Chegou meu primo favorito.

A motivação para a utilização de tipos de sentença que envolvam esse problema é a mesma que para o problema da topicalização, ou seja, testar se o sistema consegue encontrar o sujeito mesmo se este não estiver em sua posição usual, anterior ao predicado.

3.2.3 "*PP-Attachment*"

O problema do "*pp-attachment*" (ou ligação da frase preposicional) está relacionado à ambigüidade estrutural, ou seja, à possibilidade de uma oração ser representada por mais de uma árvore sintática (ou árvore de "*parsing*"). A oração

(12) O menino viu a menina com o telescópio

possibilita dois tipos de interpretação: a primeira com o menino utilizando o telescópio para ver a menina e a segunda com a menina segurando o telescópio. Em termos de estrutura, esta oração pode ser representada como na Figura 3.

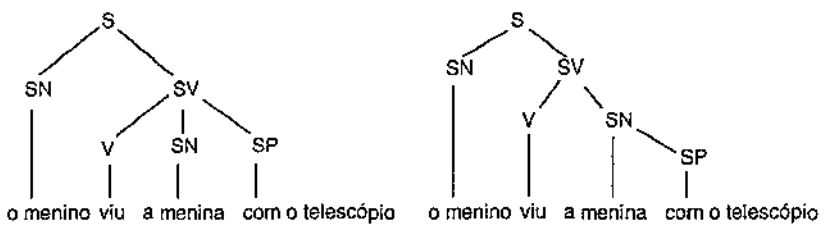

Figura 3 — Árvores representando ambigüidade estrutural relativa ao sintagma preposicional (SP)[25].

Existem alguns princípios [Allen95, p. 160] que podem forçar uma determinada interpretação em detrimento de outra, como o da *ligação mínima* ("*minimal attachment*"), que assinala existir uma preferência pela análise sintática que cria o menor número de nódulos em uma árvore sintática, ou o da *associação à direita* ("*right association*"), que assinala que, todas as outras coisas sendo iguais, os novos constituintes tendem a ser interpretados como fazendo parte do constituinte que está sendo construído, ou seja, o mais à direita em um processamento da esquerda para a direita). Podem existir preferências oriundas do léxico que sobrescrevem esses princípios,

como quando o verbo subcategoriza obrigatoriamente uma frase preposicional. Um exemplo é o verbo *assistir*, no sentido de *ver alguma coisa*, que exige a preposição "a", como na sentença "o menino assistiu ao filme".

A motivação para o tratamento desse fenômeno é verificar como é possível mudar a representação da estrutura formada levando em consideração que os agentes são os nódulos da árvore formada e as ligações estabelecidas entre esses agentes são as arestas, e que a obtenção de uma outra árvore, nesse contexto, exige que algumas ligações sejam desfeitas e outras sejam criadas.

3.2.4 Construção Relativa

A construção relativa é caracterizada por [Perini96, p. 140] como:

- a presença de um relativo (os relativos são *que*, *o qual*, *quem*, *onde*, *cujo*), precedido às vezes de uma preposição;
- a presença de uma estrutura oracional *aparentemente incompleta* logo após o relativo;
- a articulação de um *elemento nominal* (parte de um sintagma nominal), mais *o relativo*, mais *a estrutura oracional mencionada*, formando uma seqüência que é um sintagma nominal.

Um exemplo de oração com construção relativa é:

(13) O menino que a menina ama caiu.

Nesta oração, "o menino" é o elemento nominal, "que" é o relativo e "a menina ama" é a estrutura oracional aparentemente incompleta.

Tal tipo de construção é interessante porque traz consigo uma série de problemas, entre os quais:

- a possibilidade de haver uma longa distância entre o sujeito e o predicado da oração

---

[25] Nesta figura, S significa *sentença*, SN significa *sintagma nominal*, SV, *sintagma verbal* e V, *verbo*.

principal[26] — o que poderá ser traduzido como "não basta ao agente verbal procurar pelos agentes nominais em sua vizinhança próxima; tem de haver um mecanismo de busca mais elaborado e abrangente";

- a possibilidade de haver ambigüidade estrutural[27] (cf. seção 3.2.3); e
- a necessidade de preenchimento do espaço encontrado na "estrutura aparentemente incompleta" — o que pode ocasionar mudanças na direção em que o agente verbal irá procurar por um certo termo (por exemplo, se o que está faltando é o objeto direto, ele teria que procurá-lo à sua esquerda e não mais à direita[28]).

3.2.5 Coordenação

A noção de coordenação utilizada pelo sistema está intimamente ligada à de *coordenador* que [Perini96] define como "a palavra que liga dois constituintes da mesma classe, formando o conjunto[29] um constituinte da mesma classe[30] que os dois primeiros". Os exemplos abaixo ilustram a possibilidade de coordenação entre sintagmas nominais, adjetivais, adverbiais e orações, respectivamente, onde os elementos coordenados estão sublinhados:

(14) Míriam e Thiago fazem Belas-Artes.

(15) Casas de muros brancos e de telhados vermelhos ...

(16) Já morei em Fortaleza e em Natal.

(17) João ama Maria e Pedro ama Antônia.

---

26 A oração principal no exemplo é "o menino caiu", onde "o menino" é o sujeito e "caiu", o predicado.

27 Um exemplo é a seguinte oração: "o menino da bola que caiu...", onde tanto a bola como o menino podem ter caído.

28 Uma solução alternativa será mostrada no próximo capítulo.

29 O coordenador e os constituintes que ele coordena.

30 No português não há a possibilidade de coordenação de constituintes de classes diferentes como há no inglês. Veja o exemplo "*Pat is a Republican and proud of it*" [Sag84].

O interessante neste tipo de problema é que o elemento coordenador poderá assumir diferentes tipos de comportamento, dependendo da classe dos elementos que ele estiver coordenando, ou seja, o agente coordenação poderá representar mais de um papel.

Um outro motivo para utilizar sentenças com coordenação é que o elemento coordenador está sempre presente quando alguns outros fenômenos lingüísticos ocorrem, como é o caso dos próximos dois que serão abordados: redução e "*gapping*".

3.2.6 Redução

Um dos casos do que aqui se chama de redução pode ser encontrado em [Perini96] como "oração coordenada sem sujeito" (p. 297), conforme o exemplo[31]:

(18) César impressionava os homens e [ ] beliscava as mulheres.

Outro caso é o que em [Perini96] é chamado de "oração sem objeto direto" (p. 299), como no seguinte exemplo:

(19) João fez [ ] e Nadir vendeu os bolinhos.

O problema nesses dois casos é encontrar os elementos que estão faltando para que a interpretação da oração seja possível. A motivação para trabalhar com tal tipo de oração é verificar como uma abordagem estritamente *baseada na palavra* pode dar conta de sentenças nas quais há palavras faltando, ou seja, onde não haverá agentes representando-as.

3.2.7 "*Gapping*"

"*Gapping*" é um tipo de elipse que ocorre em orações coordenadas onde o elemento verbal da segunda oração é omitido, conforme os seguintes exemplos

(20) João ama Maria e Pedro, [ ] Antônia.

---

[31] Os colchetes indicam o lugar onde deveria estar o elemento faltante.

(21) João comeu a maçã e Pedro, [ ] a banana.

O primeiro problema é verificar, do ponto de vista local (isto é, de cada agente), o que caracteriza uma situação de "*gapping*". Depois disso, o problema passa a ser o da reconstituição do(s) elemento(s) faltante(s) e o da interpretação do elemento reconstituído junto aos que já existiam.

A motivação em termos do que esse problema trará para a arquitetura do sistema é a mesma do problema da redução: verificar como a arquitetura terá que lidar com a ausência de palavras/agentes.

3.3 Considerações Finais sobre o Capítulo

Neste capítulo foram vistos alguns fenômenos que podem causar problemas para um sistema com as características do foi esboçado no início deste capítulo. No próximo capítulo, um modelo lingüístico será elaborado de maneira a permitir uma abordagem distribuída que utiliza agentes com conhecimento lingüístico semelhantes aos dos sistemas apresentados no Capítulo 2. Será visto como o processo de interpretação é realizado com esse modelo de processamento e como os problemas apontados neste capítulo foram solucionados.

4. *Um Sistema Multi-Agentes para Processamento de Linguagem Natural*

(...) Não há palavra que corresponda à palavra lua, *mas há um verbo que seria em espanhol* lunecer *ou* lunar. Surgiu a lua sobre o rio *diz-se* hlör u fang axaxaxas mlö *ou seja em sua ordem: para cima atrás duradouro-fluir lualuziu.*[32]

Neste capítulo serão apresentadas as idéias de agentes para o Processamento Distribuído de Linguagem Natural. Um modelo lingüístico — que permite a implementação dos agentes de forma distribuída e que segue a idéia de segmentação de sentença e uso de conhecimento lingüístico apresentados na seção 4.2 — será apresentado na seção 4.3. Na seção 4.4 será apresentado como esse modelo foi mapeado para ser implementado em uma *fase de busca* (em que os agentes procuram por outros aos quais devem se associar) e uma fase de *construção de estruturas mais complexas* (em que as associações são efetivadas). Na seção 4.5 serão apresentados exemplos do funcionamento dos agentes com construções que foram vistas no Capítulo 3. Na seção 4.6 será apresentado o modo como o sistema detecta se a sentença que está sendo analisada pode ser considerada gramatical ou não. Finalmente, na seção 5 serão apresentados os aspectos de como os agentes foram implementados. Deve-se dizer que a maneira como os agentes foram descritos foi um tanto quanto informal e, às vezes, até antropomórfica. Espera-se que isso não comprometa o entendimento do trabalho realizado.

---

[32] Jorge Luis Borge no ensaio "Tlön, Uqbar, Orbis Tertius" em *Ficções*, Editora Globo, 3ª Edição, 1982.

4.1 Motivações

A principal motivação para este projeto foi tentar descobrir as características dos agentes (em termos de Sistemas Multi-Agentes — SMA, cf. Capítulo 1) que um sistema para Processamento Distribuído de Linguagem Natural iria utilizar. Dessa maneira, várias das noções apresentadas no primeiro capítulo foram verificadas no sistema como *resultado* da pesquisa e não como um parâmetro escolhido de forma predefinida para ser utilizado pelo sistema.

Outra motivação foi tentar utilizar e, se possível, estender as abordagens dos *sistemas baseados na palavra* apresentados no Capítulo 2. A possibilidade de utilização de agentes com as características dos sistemas baseados na palavra foi resultado de uma escolha que começou a partir das várias formas de segmentar uma sentença, que será visto a seguir.

4.2 As Possibilidades de Segmentação

As principais formas de segmentar uma sentença para poder associar agentes que irão trabalhar com esses segmentos são:

1. *não segmentar a sentença*;
2. segmentar a sentença em *sintagmas*; e
3. segmentar a sentença em *palavras*.

Para agentes associados a sentenças sem segmentação (isto é, cada agente trabalhando com sentenças completas), pode-se ter uma abordagem com *agentes por fase de processamento ou por fenômeno lingüístico*, onde cada agente fica responsável ou por uma fase do processamento (como morfologia, sintaxe, semântica etc.) ou por um fenômeno (como coordenação, elipse, negação etc.). Essa abordagem, apesar de certa semelhança com os sistemas de PLN tradicionais, explora a não seqüencialidade do processamento com o objetivo de reduzir o número de soluções (parciais ou não) que são trocadas entre uma fase e outra, isto é, entre um agente e outro.

Uma implementação desse tipo de sistema sob o enfoque de SMA pode ser encontrada em [Stefanini93, Stefanini95]. Tal sistema utiliza leis para estabelecimento das regras globais (para comunicação, negociação etc.), e para o estabelecimento das regras locais[33] (que foram chamadas de gramáticas locais). Neste sistema, os agentes são cognitivos e utilizam um protocolo de comunicação que prevê negociação entre os agentes para a resolução de conflitos.

No caso de sentenças segmentadas em sintagmas, cada agente fica responsável por um sintagma da sentença. O objetivo é fazer com que esses agentes formem associações no sentido de juntar todas as partes da oração e encontrar sua interpretação. Para poder fazer a inicialização dos agentes com esse tipo de estrutura é preciso que haja um pré-processamento para que esses sintagmas sejam encontrados. Com isso, não é possível pensar em um sistema multi-agentes homogêneo[34] partindo-se dessa abordagem.

Para sentenças segmentadas em palavras, tem-se dois tipos de abordagem. Em ambas, cada agente fica responsável por uma palavra da oração e, assim como no item anterior, eles têm de formar associações para tentar encontrar a interpretação da oração. A diferenciação das duas abordagens fica por conta da forma como o conhecimento é embutido nos agentes: em uma delas o agente representa a *própria palavra* (cf. [Small80], visto na seção 2.1) e na outra ele representa uma *classe de palavras* (cf. [Hirst87], visto na seção 2.2). A Figura 4 apresenta um panorama das abordagens vistas, com ênfase nos dois extremos (quadros em destaque à direita na figura).

---

[33] As regras globais controlam a sociedade, enquanto as locais controlam os agentes.

[34] Poderia haver um agente que faria esse pré-processamento, mas ele seria totalmente diferente dos que iriam trabalhar com os sintagmas. Para uma primeira abordagem, é muito mais fácil pensar em um sistema todo homogêneo.

Figura 4 — Tipos de associações que os agentes podem ter com as segmentações possíveis para a sentença.

Pelos motivos apresentados na seção 2.4 (à página 2-23), e também pela já existência de uma implementação da abordagem "agentes por fases de processamento e fenômenos lingüísticos" (mesmo sendo para o francês [Stefanini93, Stefanini95]), o sistema escolhido para essa dissertação foi o de "agentes por classes de palavras", cujas idéias serão expostas a seguir.

4.3 Modelo de Forças de Atração e Mecanismos de Restrição

Partindo-se do pressuposto que as palavras são o eixo central da abordagem escolhida, é necessário utilizar um modelo lingüístico que seja capaz de captar as variações existentes na língua, em diferentes aspectos, e que seja capaz de ser implementado, como no contexto atual, de forma distribuída. O modelo utilizado nesta dissertação pode ser caracterizado por duas idéias que se complementam: *forças de atração* e *mecanismos de restrição*.

4.3.1 Forças de Atração

Forças de atração (FA) captam a idéia de que algumas palavras em uma oração exercem influência sobre outras. O modo como isso pode ser modelado é dependente do nível em que as pala-

vras estão sendo tratadas. No presente caso, que as trata no nível de *classes de palavras*, o objetivo é modelar *classes de palavras atraindo classes de palavras*.

Como as classes com as quais se está trabalhando nesta dissertação são verbos, substantivos, preposições, determinantes, adjetivos, pronomes relativos e coordenadores, então o que se quer é um mapeamento de quais outras classes uma determinada classe pode atrair, o que pode ser visto na Tabela 2.

Classe	*Classe(s) Atraída(s)*
• verbos	• substantivos, preposições, pronomes relativos
• substantivos	• determinantes, adjetivos, preposições
• preposições	• substantivos, verbos[35]
• coordenadores	• verbos, substantivos
• pronomes relativos	• substantivos

Tabela 2 — Mapeamento das atrações entre classes de palavras.

Essa idéia de FA tem uma certa semelhança com a idéia de "*dependency grammar*" (DG) [Hays64, Milward94] no sentido expresso pelo seguinte parágrafo de [Hays64, p. 511]:

> *"The insight formalized by dependency theory is that particular occurrences of minimal units are directly related to one another*[36]*. Occurrence of an adjective before a noun, for example is more closely related to the noun occurrence than to anything else. In describing the distribution of a class of adje-*

---

[35] Este caso de preposição atraindo verbo não está sendo tratado. Um exemplo pode ser visto na seguinte sentença: "Isto foi feito *para cortar* legumes". Os exemplos tratados utilizam apenas a idéia para substantivos, como em "O menino *com* a *bola*...".

[36] Grifos do autor da presente dissertação.

tives, therefore, one naturally refers to the classes of nouns with which the adjectives occur. Similarly, an occurrence of a preposition is generally most directly related to two other occurrences, its object and the unit modified by the prepositional phrase".

4.3.2 Mecanismos de Restrição

Apenas com a idéia de forças de atração não se consegue obter uma análise completa de uma sentença porque, se assim fosse, os verbos iriam atrair todos os substantivos (se existir mais de um), e o mesmo ocorreria com as outras classes. Assim é necessário a utilização de restrições que impeçam as possíveis associações erradas.

No presente trabalho estão sendo utilizados três tipos de mecanismos — subcategorização, vizinhança e restrição semântica — que podem ser assim descritos:

- *Subcategorização* pode ser entendida como o mecanismo que especifica o tipo de argumento (neste contexto, em termos da classe de palavras) que deve (pode) preencher uma determinada posição sintática (sujeito, objeto etc.) no verbo[37];
- *Vizinhança* pode ser entendida como o mecanismo que limita a esfera de influência que uma classe de palavras tem sobre outra em termos espaciais.
- *Restrição semântica* pode ser entendida como o mecanismo que restringe a categoria semântica de uma palavra de acordo com o que ela espera da outra palavra que ela atrai (ou pela qual é atraída).

Na próxima seção será apresentado como essas idéias de forças de atração e de mecanismos de restrição foram integradas em um processo de interpretação que utiliza a noção de agentes.

---

[37] Está-se utilizando subcategorização apenas para verbos.

4.4 Processo de Interpretação para Agentes

O processo de formação do significado de uma sentença — no modelo de forças de atração e mecanismos de restrição — pode ser compreendido como o modo que as palavras se inter-relacionam em uma sentença *direcionadas* pelas forças de atração e *limitadas ou reforçadas* pelos mecanismos de restrição apresentados.

Assim, o *processo de interpretação* de uma sentença deve ser visto como um *processo de busca* pelas palavras que modificam outras (ou pelas quais são modificadas), direcionado pelas *forças de atração* e limitado (ou reforçado) pelos *mecanismos de restrição* apresentados logo acima.

Quando essa busca é bem sucedida, então *ligações* (ou *associações*) devem ser estabelecidas formando estruturas mais complexas e, quando *todas* as palavras estiverem ligadas, pode-se partir para o processo da *construção da forma lógica* (cf. [Allen95], Capítulos 8 e 9) que representará o significado da sentença no nível semântico.

Esse processo de interpretação — busca, estabelecimento de ligações e construção da forma lógica — pode ser mapeado diretamente para uma representação que utilize agentes (cf. seção 1.2, à página 1-6). Cada agente será associado a uma palavra da sentença e terá seu comportamento representado pela classe morfológica da palavra em questão.

Assim, manuseando as informações relativas à sua palavra[38], cada agente irá utilizar sua capacidade de raciocínio e de comunicação para perfazer as três etapas citadas. As duas primeiras serão elaboradas mais detalhadamente nas próximas seções e a terceira não será tratada nesta dissertação.

---

[38] Neste sentido, o dicionário (ver um exemplo na seção 4.5.1.1) deve ter uma representação bem rica, visto que é a única fonte de informação utilizada pelos agentes.

4.4.1 A Fase de Busca

A fase de busca pode ser analisada de acordo com o comportamento de cada tipo de agente e a extensão espacial que cada tipo de agente utiliza na busca (associada ao tipo de comunicação utilizado pelos agentes).

4.4.1.1 *Mapeando o Comportamento dos Agentes*

O *agente verbal* tem por papel encontrar os agentes que preenchem os seus argumentos e o processo de busca, no processamento de qualquer sentença, é iniciado por ele, pois esse tipo de agente tem a noção exata do que deve procurar[39]. Em mensagens endereçadas a todos os outros agentes, ele envia requisições de agentes que satisfaçam suas necessidades em termos de posicionamento, da categoria semântica esperada e do tipo do agente (ou seja, da classe morfológica). Os agentes que satisfizerem as condições enviarão proposta(s) candidatando-se ao preenchimento de um determinado argumento. O agente que tiver a melhor proposta será o escolhido para o estabelecimento de ligação (que será visto na seção 4.4.2).

O *agente nominal* também é importante, mas seu comportamento é passivo em relação aos outros agentes. Em vista das mensagens com requerimentos que ele recebe dos agentes verbais, o agente nominal retorna proposta(s) (caso satisfaça as condições especificadas nos requerimentos) candidatando-se à(s) posição(ões) requisitada(s). Caso seja escolhido, ele receberá do agente verbal um pedido de estabelecimento de ligações. Com relação aos agentes preposicionais, o nominal é ainda mais passivo — o preposicional é que realiza todas as operações necessárias, como será visto a seguir.

---

[39] De acordo com a sua subcategorização associada às restrições semânticas e de vizinhança. No sistema KERNEL [Palmer93] as coisas se passam de uma maneira parecida: "*The quest to fill in the arguments of the matrix verb provides the driving force for the* clause analysis *process*".

O *agente preposicional* deve primeiramente encontrar o agente nominal que tem de estar a sua direita e se ligar a ele. Depois, pode agir de duas maneiras, na seguinte ordem: (1) verificar se há requisições de agentes verbais[40] — comportando-se passivamente (assim como os agentes nominais); ou (2) procurar pela sua vizinhança esquerda no sentido de encontrar os tipos de agentes que ele pode modificar (nominais e verbais) e, no caso de encontrar, pedir o estabelecimento de ligações — comportando-se ativamente.

O *agente pronome relativo* possui um comportamento semelhante ao preposicional, no sentido de ter que fazer a busca por dois agentes, um de cada lado. Ou seja, ele tem que encontrar um agente verbal à sua direita e verificar como deve se ligar a ele de acordo com o "buraco" ("*gap*") que este possui[41], utilizando para isso o fato de existir ou não algum agente nominal entre eles (agente pronome relativo e agente verbal); depois disso, deve encontrar o agente nominal que deverá estar à sua esquerda e se ligar a ele.

O *agente coordenação* tem por função encontrar dois agentes do mesmo tipo, um de cada um dos seus lados, no sentido de formar uma estrutura coordenada (por exemplo, coordenação de dois agentes nominais ou de dois verbais). Se ele estiver coordenando dois agentes nominais, então ainda terá que se comportar como um agente nominal, respondendo a requisições de agentes verbais, por exemplo.

O *agente determinante* tem por função encontrar o agente nominal que ele especifica e que deve estar à sua direita. Os últimos três tipos de agentes se comportam de maneira ativa no processo de busca — isto é, são eles que tentam localizar os agentes que eles modificam (ou pelos quais são modificados).

---

[40] Por exemplo, quando o verbo subcategoriza uma preposição obrigatoriamente, como o verbo "assistir a" (ver).

[41] O que fez com que [Perini96] caracterizasse tal parte da construção relativa como uma "estrutura oracional *aparentemente incompleta*" (cf. seção 3.2.4, à página 3-30 desta dissertação).

A questão do comportamento também pode ser relacionada à questão de o agente conseguir estabelecer suas ligações, o que será melhor comentado na seção 4.4.2.

4.4.1.2 *A Questão da Extensão da Vizinhança e de Como Fazer o Endereçamento*

A noção de vizinhança utilizada no sistema diferencia a busca apenas com relação à direção em que ela deve ser realizada. Não se utilizou no sistema uma medida restritiva em termos do número máximo de palavras (ou, no caso, de agentes) que um determinado agente pode enxergar. Entretanto, pensou-se que seria interessante utilizar dois modos diferentes de endereçamento de mensagens para que a comunicação não se tornasse um gargalo para o desempenho do sistema: endereçamento individual (para um agente específico) e endereçamento global (para todos os agentes — "*broadcast*").

O *endereçamento individual* é utilizado por todos os agentes (exceto pelo agente verbal) e o que se espera é que o agente procurado esteja na vizinhança próxima. O *endereçamento global* é utilizado apenas pelo agente verbal para os termos obrigatórios que ele subcategoriza devido à possibilidade de esses termos estarem a uma distância muito grande do agente, como ocorre, por exemplo, em sentenças que contêm a construção relativa (cf. seção 3.2.4, à página 3-30).

As mensagens de busca seguem dois padrões, de acordo com o tipo de endereçamento (individual ou global) utilizado. Para o endereçamento individual, que corresponde à idéia de *busca próxima*, o agente utiliza uma mensagem que corresponde à sua necessidade de encontrar um *vizinho de um determinado tipo* (por exemplo, nominal, verbal etc.) relativo a uma determinada direção, iniciando em uma determinada posição (por exemplo, a do próprio agente).

Para o endereçamento global, que corresponde à idéia de *busca distante*, utilizou-se um protocolo semelhante ao da *rede de contrato*, visto na seção 1.3.2, à página 1-9. Enquanto em [Davis83] tal protocolo foi usado para a alocação de tarefas, aqui ele foi utilizado, apenas pelos agentes verbais, para a escolha do melhor agente com quem se ligar.

O esquema de mensagens utilizado no endereçamento global, correspondente ao citado na seção 1.3.2, é o seguinte: o agente verbal anuncia a necessidade de um determinado tipo de agente[42], espera as mensagens com as propostas[43] (que anteriormente foram chamadas de "ofertas"), e escolhe o agente que melhor satisfaz as características de que ele necessita. Há de se lembrar que os agentes verbais utilizam este tipo de busca apenas para as posições sintáticas obrigatórias (ou seja, relativas às suas subcategorizações obrigatórias).

4.4.1.3 *Considerações Finais Sobre a Fase de Busca*

É interessante, neste momento, enfatizar onde são utilizadas as idéias do modelo de *forças e restrições* exposto na seção 4.3. Quando um agente vai procurar por outro, essa busca já leva em consideração qual o tipo de agente que ele deverá procurar, que é a tradução para a prática da idéia de forças de atração.

O mecanismo de *subcategorização* é utilizado pelos agentes verbais no que foi caracterizado como processo de busca global e é de extrema importância porque é a informação mais precisa quando do começo do processamento.

O mecanismo de *vizinhança* pode ser visto de duas maneiras. A primeira é pela *direção* em que a busca deve ser feita. A segunda é pela *distância* do agente que se procura. Se a busca é distante (cf. seção anterior), quando vários agentes enviam propostas, então um dos fatores para a escolha da melhor proposta (melhor agente) é a proximidade do agente que enviou a proposta. Se a busca é próxima, esse fator já está embutido no processo.

---

[42] A mensagem especifica exatamente: o *identificador do emissor*, o *tipo do agente* necessitado (por exemplo, nominal, preposicional etc.), a *posição sintática* que o agente deverá ocupar (por exemplo, "sujeito", "objeto" ect.), a *direção* relativa ao agente que faz a busca (esquerda, direita ou qualquer), e o *tipo semântico* do agente necessitado.

[43] Que, na realidade, retorna o *identificador do agente* que responde, o seu *tipo semântico*, que pode ser diferente do tipo necessitado (por exemplo, mais especializado) e a *posição sintática* (ver nota 42).

O mecanismo de *restrição semântica* é utilizado não apenas para conferir se o agente encontrado condiz com a categoria semântica do que é procurado, mas também, como em [Hirst87] (cf. seção 2.2.2, à página 2-21 desta dissertação), para resolver os casos de ambigüidade semântica, escolhendo o melhor sentido para uma palavra, representada por um determinado agente, com relação à(s) necessidade(s) de outra(s) palavra(s), ou seja, com relação ao(s) outro(s) agente(s).

4.4.2 Formação de Estruturas Mais Complexas

A formação de estruturas mais complexas consiste na criação de uma hierarquia que segue basicamente a aparência de uma árvore de interpretação sintática (árvore de "*parsing*"). Essa hierarquia também foi pensada no sentido de que a formação da forma lógica, representando a interpretação semântica, fosse facilitada.

4.4.2.1 *Construção da Hierarquia*

Há duas questões que têm de ser analisadas na construção de uma hierarquia em um sistema que inicia o seu processamento pela palavra como o dessa dissertação. A *primeira questão* diz respeito à necessidade de criação ou não de agentes de nível diferente do nível de classe de palavra como, por exemplo, agentes sintagmas. Acredita-se que isso não seja necessário, visto que a função que esse novo agente exerceria pode ser realizada por um agente já existente na sentença, que é o agente que atrai os outros (por exemplo, para um sintagma nominal, seria o agente nominal e para a sentença, seria o agente verbal). Essa foi a idéia utilizada nessa dissertação e, com isso, evitou-se a duplicação de esforços desnecessários.

A *segunda questão* diz respeito às maneiras de implementar a estrutura hierárquica: utilizando ligações ou agregando agentes. Na primeira, o agente mantém uma representação em sua estrutura de dados em que assinala apontadores para seus superiores e seus subordinados. Na segunda, o agente incorporador copia a representação dos outros agentes (seus dados) que, então, passa a representar: qualquer pergunta referente a esses dados deve, então, ser respondida pelo agente incorporador, visto que o agente incorporado é desativado.

Dois exemplos de formação de uma estrutura do tipo *sintagma nominal* a partir de um agente determinante e um agente nominal, utilizando a idéia da não necessidade de criação de agentes de nível diferente do nível de classe de palavras, podem ser vistos na Figura 5 (por agregação) e na Figura 6 (por ligação).

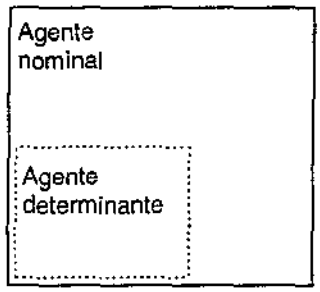

Figura 5 — Formação de uma estrutura do tipo sintagma nominal usando agregação.

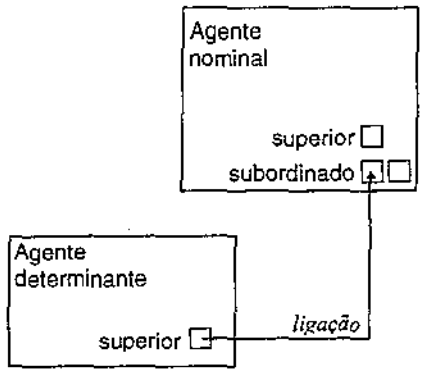

Figura 6 — Formação de uma estrutura do tipo sintagma nominal usando ligações.

As principais diferenças entre as duas formas de implementar a hierarquia podem ser resumidas pelos seguintes fatores:

- Na agregação, quando um agente incorpora os dados de um outro, este passa a não ter mais função e, então, é desativado (cf. [Devos94], visto na seção 2.3). Por isso, o agente incorporador terá que saber lidar com as informações que vêm dos outros agentes. Quando as combinações possíveis são várias, isso torna o novo agente difícil de ser gerenciado. O mesmo não ocorre com a utilização de ligações, visto que os agentes que estão se juntando para formar a estrutura mais complexa continuarão a existir e poderão responder por suas atividades específicas;
- Se a estrutura formada não for a adequada, por uma razão qualquer, então será necessário voltar a uma representação anterior. No caso das ligações, basta apagá-las nos agen-

tes envolvidos. No caso da agregação é preciso apagar as estruturas que foram copiadas para dentro do agente incorporador e fazer com que os agentes desativados tenham seu funcionamento restabelecido.

Por esses dois motivos é que se decidiu trabalhar nesta dissertação com a formação de ligações, ao invés de agregação. Assim, a sentença "o menino caiu ..." seria representada como apresentado na Figura 7.

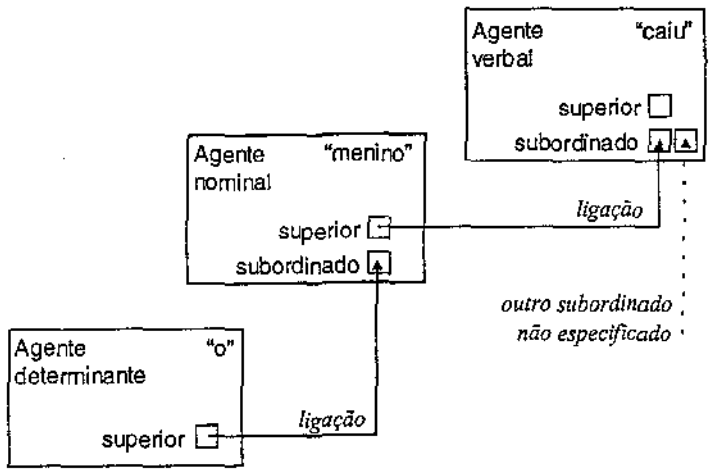

Figura 7 — Representação da hierarquia, com ligações, para a sentença "o menino caiu ...".

4.4.2.2 *A Importância das Ligações como Comportamento nos Agentes*

A principal função de cada agente é estabelecer as ligações necessárias (que serão vistas posteriormente). A metáfora que pode usada é a seguinte: *um agente sem ligação é como uma palavra que não tem função em uma sentença.*

Assim, a função de cada agente *só termina* quando ele consegue estabelecer todos os laços necessários, o que varia para cada tipo de agente. Para *agentes verbais* isso significa que eles preencheram todos os argumentos (considerados como subordinados) especificados como "obrigatórios" (isto é, de subcategorização obrigatória) no dicionário. Para *agentes nominais*, si-

gnifica que eles possuem um único superior[44], não importando o tipo (verbal ou preposicional). Para *agentes preposicionais* e *pronomes relativos* significa que eles têm de ter um superior (verbal ou nominal para os preposicionais, e verbal ou preposicional[45] para os relativos) e um subordinado (nominal). Para os *agentes determinantes*, que eles possuem um superior (um nominal) e nenhum subordinado. Para os *agentes coordenação*, que eles possuem dois subordinados e, no caso de uma coordenação de nominais, que eles (agentes coordenação) possuem um superior (porque irão se comportar como um agente nominal).

É importante salientar, então, que uma condição de parada do sistema (para as frases gramaticais) é todos os agentes conseguirem fazer as ligações necessárias[46]. A partir desse ponto, seria possível fazer a construção da forma lógica resultante através de outros tipos de interação entre os agentes.

4.4.2.3 *Considerações Finais sobre a Formação de Estruturas*

Viu-se que uma grande importância do processamento do sistema está depositada sobre as ligações que os agentes realizam. Entretanto, como as decisões que cada agente toma são baseadas em visões locais[47], é possível que uma ligação realizada por um agente não esteja coerente com a situação global da sentença e isso pode gerar conflitos (por exemplo, com dois agentes querendo se ligar a um mesmo agente que só aceita uma ligação). Tais situações serão melhor descritas na

---

44 Há uma exceção para frases que possuem pronome relativo, no sentido de um agente nominal poder ter mais de um superior. Entretanto, ele deverá ter um único superior (*principal*) que não seja pronome relativo e poderá ter vários superiores deste tipo. Isso será visto no final da seção 4.5.4.

45 Embora não tenha sido implementada a interação de agente pronome relativo com preposicional, um esboço para sua implementação pode ser visto na nota de rodapé 53, à página 59.

46 Isso implica, basicamente, que todo agente tem que possuir um superior (principal, cf. nota de rodapé 44), com exceção dos agentes verbais, que só irão possuir superior em frases coordenadas.

47 Nenhum agente possui uma visão global da sentença.

seção 4.5.7 à página 4-66 desta dissertação, onde serão apresentadas soluções para esse problema.

A seguir serão apresentados vários exemplos de sentenças que exercitarão as características do modelo até então apresentado.

4.5 Exemplos

Nessa seção serão dadas soluções, em termos das operações dos agentes, aos problemas apresentados no Capítulo 3. Além disso, alguns exemplos terão sua execução simulada para que as mensagens trocadas entre os agentes possam ser descritas em um contexto mais compreensível.

As estratégias de busca utilizadas pelos agentes não esgotam as possibilidades de frases construídas com os tipos de agentes utilizados neste trabalho. Nesta seção só serão apresentadas as frases que os sistema consegue processar e as que ele não consegue serão apresentadas no Capítulo 6, com esboços de solução para algumas delas.

4.5.1 Um Exemplo Simples Completo

Nesta seção será visto como a sentença

(22) O menino matou o gato.

apresentada no Capítulo 3, é processada pelo sistema de forma completa. No presente sistema, assume-se que já foi realizado um pré-processamento para a resolução de *ambigüidade categorial*, que acontece quando uma palavra pertence a mais de uma categoria (como verbo e substantivo). Dessa maneira, quando se associa um agente a uma palavra, assume-se que essa palavra pode ter vários significados, mas todos dentro de uma mesma categoria morfossintática. A razão para isso é que modelar o conhecimento dos agentes de acordo com o comportamento de apenas uma categoria morfossintática faz com que o código para os agentes fique mais estruturado e

estratégias específicas para seu processamento sejam possíveis. Duas formulações para lidar com ambigüidade categorial (uma utilizando agentes e outra não) serão esboçadas no Capítulo 6.

O sistema é inicializado com todas as palavras da sentença sendo lidas e suas informações obtidas no *dicionário*. Assim, para cada palavra, será criado um agente de acordo com a categoria morfossintática da palavra lida. Os agentes serão postos para funcionar ao mesmo tempo, cada um iniciando seu *processo de busca* e tentando *estabelecer as ligações* necessárias. Serão apresentadas a seguir as informações que se pode obter no dicionário para as palavras da sentença (22).

4.5.1.1 *Dicionário*

Uma representação do dicionário para as palavras da sentença (22) pode ser visto abaixo:

verbo "matou"	obrigatório	(	suj,	[(agente: humano), (instrumento: arma)],
			obj,	[(tema: ser_vivo)])
	opcional	(	obj_ind,	[(preposição: com, instrumento: arma)])
substantivo "menino"	tipo(ser_humano), gênero(masculino), número(singular)			
substantivo "gato"	tipo(felino), gênero(masculino), número(singular)			
determinante "o"	tipo(definido), gênero(masculino), número(singular)			

Nas informações relativas aos verbos, o dicionário possui dois campos que especificam as subcategorizações obrigatórias e as opcionais. O verbo matou subcategoriza para *sujeito* ou um humano (neste caso, ocupando o papel temático de AGENTE) ou uma arma (neste caso, ocupando o papel temático de INSTRUMENTO). Essas duas possibilidades cobrem casos como o do exemplo acima e também como o da sentença "o revólver matou o gato". Para *objeto direto*, o verbo tem que ter um ser_vivo, que terá então o papel temático de TEMA. Opcionalmente, o verbo poderá ter um *objeto indireto* formado com a preposição com associada a uma arma, que ocupará então o papel temático de INSTRUMENTO e que pode ser visto em um exemplo como "o menino matou o gato com o revólver".

Nas informações relativas aos substantivos, o dicionário possui informação acerca do seu significado, do gênero e do número. Se uma palavra possuir mais de um significado, então eles constituirão uma lista como a que pode ser vista abaixo para a palavra cravo, que tanto pode ser uma flor, quanto um tipo de prego, um instrumento musical etc[48].

substantivo "cravo" tipo([flor, prego, instrumento musical,...]),
gênero(masculino), número(singular)

Nas informações relativas aos determinantes, o dicionário possui seu tipo (está-se trabalhando apenas com os tipos definido e indefinido) e informações sobre o gênero e o número. Todas essas informações do dicionário serão utilizadas para guiar os agentes na fase de busca.

4.5.1.2 *A Fase de Busca*

Para a sentença (22), serão instanciados agentes do seguinte modo: um agente determinante (ag_1) associado à palavra o; um agente nominal (ag_2) associado à palavra menino; um agente verbal (ag_3) associado à palavra matou; um agente determinante (ag_4) associado à palavra o; e, finalmente, um agente nominal (ag_5) associado à palavra gato.

De acordo com o comportamento descrito na seção 4.4.1.1 (à página 4-41), o agente verbal irá enviar mensagens globais para cada um dos significados de suas subcategorizações obrigatórias. Assim, ele enviará mensagens que seguem o padrão:

preciso(tipoAgente, local, emissor, tipoSintático[49], tipoSemântico)

Para sujeito, esta mensagem ainda teria especificado o gênero e o número e, para a sentença (22), teriam a seguinte forma:

preciso(nominal, menor, idAgVerbal, sujeito, 3_pessoa, singular, arma)
preciso(nominal, menor, idAgVerbal, sujeito, 3_pessoa, singular, humano)

---

[48] O mesmo ocorre para os verbos que, possuindo vários sentidos, pode ter subcategorizações diferentes também.

[49] TipoSintático pode ser "sujeito", "objeto (direto)" e "objeto indireto".

Isso significa que o agente verbal (idAgVerbal é o identificador do agente) procura por um sujeito à sua esquerda (como indica o segundo parâmetro: menor) que esteja na terceira pessoa do singular e que tenha como valor semântico arma e humano, para a primeira e a segunda mensagem respectivamente. A(s) mensagem(s) para objeto direto seguem o mesmo padrão.

Enquanto o agente verbal envia mensagens globais para suas subcategorizações obrigatórias, cada agente determinante busca por um agente nominal pela sua vizinhança utilizando mensagens diretas. Esse primeiro passo na interpretação da sentença pode ser visto na Figura 8.

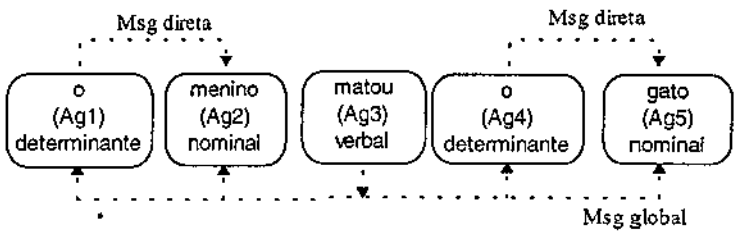

Figura 8 — Primeiro passo na interpretação de uma sentença.

Os agentes nominais, recebendo precisos do agente verbal, responderão com propostas. O agente nominal (ag_2) irá responder à mensagem que especifica o tipo semântico como humano e nenhum agente irá responder à mensagem que especifica o tipo arma. Assim, o sujeito da sentença terá o papel temático AGENTE. Cada agente determinante, encontrando o agente nominal à sua direita, irá pedir estabelecimento de ligação com este agente. Esse segundo passo pode ser visto na Figura 9.

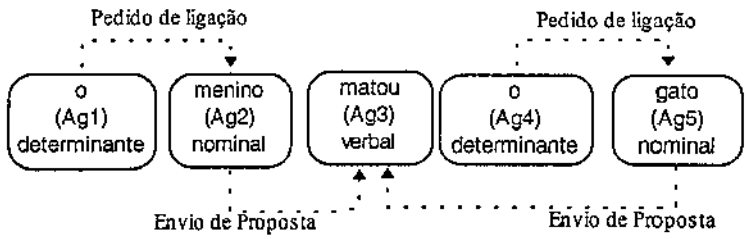

Figura 9 — Segundo passo na interpretação de uma sentença.

Com esses passos, dá-se por encerrada a fase de busca. Não necessariamente os acontecimentos se passam em termos de "passos", mas eles servem como uma ilustração do sistema funcionando. Após essa fase de busca, os agentes irão tentar estabelecer ligações, o que será visto a seguir.

4.5.1.3 *Criando uma Hierarquia*

Quando os agentes começam a estabelecer ligações, eles estão, em realidade, construindo uma hierarquia que representará a estrutura final da sentença interpretada. O começo desta fase acontece, para cada agente, quando ele envia em uma mensagem um pedido de estabelecimento de ligação para outro agente (ver Figura 9).

Se o agente que recebeu o pedido (na Figura 9, seria o ag_2) aceita-o, então ele assinala em sua estrutura que possui agora um subordinado (ou superior, dependendo do caso) e envia uma mensagem para o agente que enviou o pedido (ag_1) informando o aceite (não apresentado na figura). O agente que enviou o pedido (ag_1) então irá atualizar seus dados assinalando que agora possui um superior (ou subordinado). Se o agente que recebeu o pedido (ag_2) não o aceita, então apenas envia uma mensagem informando o *não aceite.*

Para a sentença (22), a estrutura final é a representada pela Figura 10, onde as flechas saem do subordinado em direção ao superior.

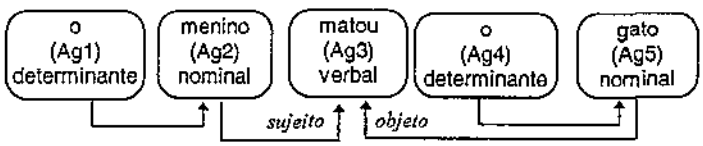

Figura 10 — Estrutura hierárquica formada com as ligações dos agentes.

Quando todos os agentes conseguem estabelecer suas ligações, então eles terminaram sua função e o sistema pára. A partir de então começaria a fase de interpretação semântica, que não foi implementada.

Nas próximas seções serão apresentados exemplos de como foram resolvidos os problemas gerados pelos fenômenos lingüísticos apresentados no Capítulo 3. Nesses exemplos, não serão

apresentados todos os passos da resolução, mas sim os pontos principais para a solução do problema específico.

4.5.2 Topicalização e Posposição do Sujeito

Esse dois fenômenos lingüísticos foram aqui agrupados porque geram o mesmo problema em termos de agentes, ou seja, o fato de um agente não estar na posição esperada na sentença. Além disso, eles estão relacionados ao agente verbal e ao tipo de busca utilizada por este tipo de agente.

O agente verbal faz a busca pelos agentes de suas subcategorizações obrigatórias de forma indireta (cf. seção 4.4.1.2) enviando uma mensagem (para cada termo subcategorizado obrigatoriamente) para todos os agentes especificando as informações necessárias de forma que os outros agentes saibam se podem ou não se candidatar a tal posição subcategorizada. De interesse neste momento é a *direção* especificada para o agente procurado.

Assim, para objeto direto, o agente verbal sempre especifica que este virá a sua direita (ou seja, depois dele). Dessa maneira, todos os agentes nominais[50] que não possuírem um superior (cf. seção 4.4.2.1) e que estiverem após a posição do agente verbal irão retornar uma mensagem candidatando-se a ser o objeto direto deste agente verbal.

Se nenhuma mensagem for retornada ou se não for possível estabelecer ligações com o agente que teve a melhor proposta (porque pode ocorrer que, nesse meio tempo entre as trocas de mensagens, o agente da melhor proposta tenha se ligado a outro), então o agente verbal necessitará de um mecanismo para verificar a possibilidade de o objeto direto estar topicalizado.

A solução encontrada foi fazer com que o agente verbal anunciasse novamente a posição objeto direto com a diferença que, agora, ele *não irá especificar a direção* dessa busca — tal solu-

---

[50] Porque, para objeto direto, é especificado o tipo de agente como sendo nominal.

ção recebeu o nome de "*flexibilização*". Ou seja, após a flexibilização, qualquer agente nominal da sentença que possuir os requisitos semânticos necessários e que não possuir superior poderá candidatar-se à posição, visto que não há restrição quanto à posição.

O critério para a escolha da melhor proposta para objeto direto no caso da posição normal, já contando com o fato de que só retornam mensagens os agentes que satisfazem as restrições semânticas e a direção de busca, é a proximidade do candidato em relação ao agente verbal pela direita. Ou seja, dados dois candidatos, o que estiver mais próximo ao agente verbal será o escolhido. Já o critério usado para o caso da topicalização é também estar o mais próximo do agente verbal pela esquerda, mas é necessário levar em consideração que, se existir sujeito na oração, então esse objeto direto não poderá estar entre o sujeito e o verbo (ver [Perini96, p. 90], onde são apresentados os traços sintáticos do objeto direto).

A questão do sujeito posposto foi resolvida da mesma maneira. Na seção 3.2.2 desta dissertação, algumas restrições à posposição do sujeito foram apresentadas: não existência de objeto direto na oração[51] e necessidade de o verbo ser marcado como permissível de sujeito posposto. Assim, se o verbo subcategorizar um objeto direto, então o agente verbal não irá flexibilizar a posição de sujeito, no caso de não haver candidato para a posição. A questão de os verbos serem marcados como permitindo a posposição não foi levada em consideração devido à falta de um estudo sobre exatamente quais verbos devem e quais não devem ser assim marcados [Perini96, p. 233]. De qualquer maneira, a introdução dessa característica não traria dificuldade alguma em termos da mudança que deveria ser feita no funcionamento do agente verbal.

4.5.3 "*PP-Attachment*"

O fenômeno "*pp-attachment*" faz com que haja a necessidade de o sistema conseguir gerar mais de uma interpretação. Como o agente preposicional não tenta utilizar as estratégias da "ligação

---

[51] [Perini96] não analisa a questão com relação ao objeto indireto.

mínima" ou da "associação à direita" (cf. seção 3.2.3) para decidir com qual agente ele deve se ligar, então a primeira questão a ser resolvida é a detecção da dupla possibilidade de ligação.

Seguindo a descrição do comportamento para o agente preposicional à página 4-42 desta dissertação, este agente possui duas possibilidades. Se houver um verbo que subcategoriza obrigatoriamente sua preposição (o que implica que o agente verbal enviará uma mensagem global para tal subcategorização), então o agente preposicional age de uma forma passiva, candidatando-se à posição requisitada pelo agente verbal. Se não houver, ele deve fazer uma busca pela sua vizinhança à esquerda (anterior a ele) e verificar a existência de algum agente verbal e/ou nominal. Se houver os dois, então há a possibilidade de dupla interpretação.

Primeiramente, o agente preposicional tenta estabelecer ligação com o agente verbal. Se não for aceita, então existe apenas uma possibilidade — tentar estabelecer ligação com o agente nominal. Se a ligação com o agente verbal for aceita, então ainda haverá a possibilidade de ligação com o nominal.

Para entender a questão de como a segunda interpretação será realizada, é preciso entender quando se chegou a uma solução. No sistema, a condição para isso é que todos os agentes tenham conseguido estabelecer suas ligações necessárias. Os agentes então iriam passar suas informações semânticas[52] e uma solução (forma lógica) poderia ser construída.

Para forçar uma outra interpretação, o agente que tiver essa possibilidade deverá anunciá-la (enviando mensagem endereçada a todos os agentes) para evitar que todos os outros agentes sejam desativados com o fim da primeira interpretação. Dessa maneira o término é evitado e, então, o agente pode pedir ao agente a quem ele está ligado para quebrar sua ligação. Após isso, ele deve tentar estabelecer ligações com o outro agente.

---

[52] A fase de interpretação semântica não foi implementada.

Para a sentença

(23) O menino viu a menina com o telescópio

os passos podem ser vistos nas figuras abaixo. A Figura 11 representa a verificação das duas possibilidades (o que pode ser visto pelas linhas tracejadas). A Figura 12 apresenta a interpretação (ou seja, as ligações) em que o menino utiliza o telescópio como instrumento de visualização (além do envio da mensagem outra interpretação) e a Figura 13, a interpretação em que a menina está segurando o telescópio.

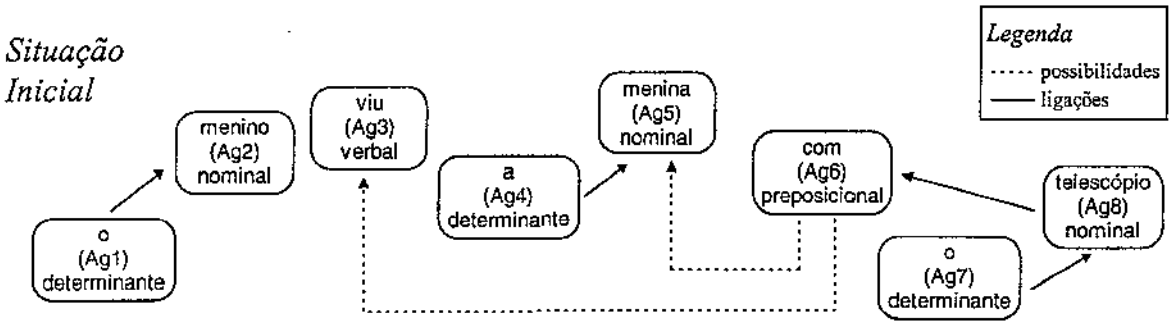

Figura 11 — Agente preposicional verificando dupla possibilidade de interpretação.

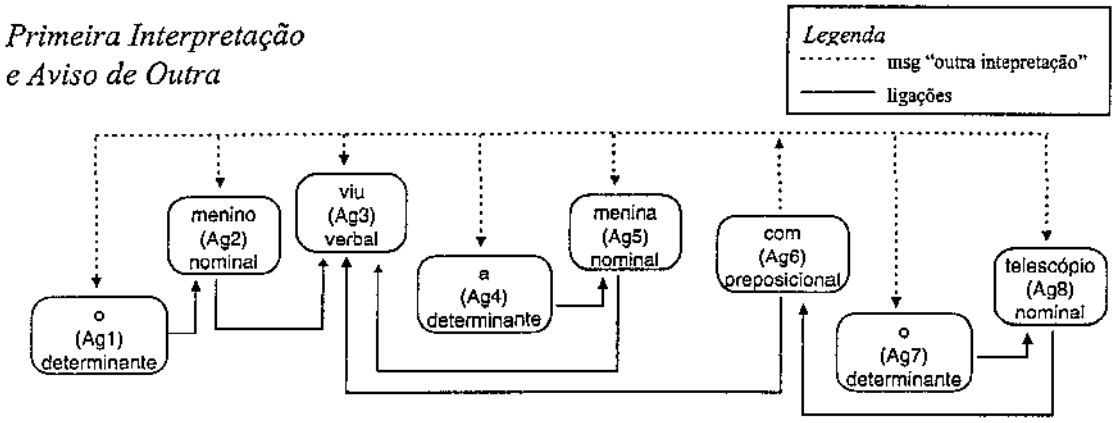

Figura 12 — Ligação do agente preposicional ao agente verbal como primeira interpretação.

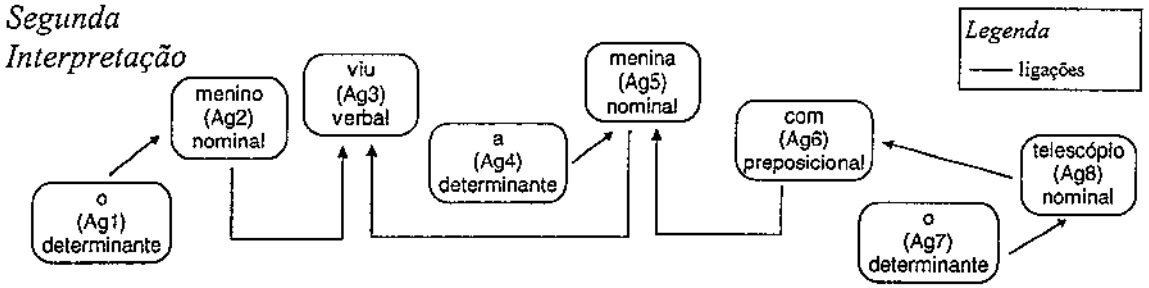

Figura 13 — Ligação do agente preposicional ao agente nominal como segunda interpretação.

A solução apresentada utiliza então uma nova mensagem global (outra interpretação) com o anúncio da existência de uma outra interpretação.

4.5.4 Construção Relativa

Na seção 3.2.4, à página 3-30, foram apontados três motivos pelos quais o tratamento da construção relativa seria interessante: o problema da distância, o problema da ambigüidade estrutural e o da incompletude estrutural. O problema da distância, pela estratégia de busca global apresentada na seção 4.4.1.2, não chegou a ser realmente um problema. O da ambigüidade estrutural para o agente pronome relativo não chegou a ser implementado, mas sua solução seguiria as mesmas diretrizes que as utilizadas para o "*pp-attachment*". O problema da estrutura incompleta será analisado a seguir.

Em sentenças como

(24) O menino que ama a menina caiu.

(25) O menino que a menina ama caiu.

já é possível delinear um papel inicial para o agente pronome relativo. A primeira providência desse agente é encontrar o agente verbal a sua direita com quem ele deve se ligar. Para os exemplos apresentados, tal agente verbal seria o primeiro agente à direita, mas em outros exemplos, como o apresentado em (26), outros fatores teriam que ser analisados:

(26) O menino que$_1$ a menina que$_2$ caiu ama comprou uma bicicleta.

Na sentença (26), a presença de um outro agente pronome relativo antes de um agente verbal mostra que nesta situação é necessária uma estratégia que lide com esse tipo de estruturação aninhada, ou seja, o primeiro pronome relativo (que_1) terá que se ligar ao último verbo (ama) e o último (que_2), ao primeiro (caiu). Na busca pela vizinhança realizada pelo agente pronome relativo, ele leva em consideração esses fatos.

A questão da estrutura incompleta pode ser resolvida de duas maneiras. A primeira é com o agente pronome relativo verificando qual termo está faltando para ser preenchido no agente verbal, o que pode gerar uma certa demora porque o agente verbal pode não ter realizado nenhuma ligação, ou seja, pode não ter nenhum de seus termos (sujeito, objeto etc.) ainda preenchidos. E isso pode atrasar o processamento do agente pronome relativo.

A outra maneira é o agente relativo verificar se há um agente nominal entre ele e o agente verbal. Se houver, então este agente nominal terá que ser o sujeito do agente verbal e o pronome relativo será o objeto direto[53]. Isso é o que ocorre na sentença (26). Se não houver um agente nominal, então o pronome relativo será o sujeito da sentença. Essa foi a estratégia utilizada pelos agentes do sistema.

Uma questão que é de interesse diz respeito ao tipo de ligação entre o agente pronome relativo e seu subordinado (agente nominal). O agente pronome relativo sempre será superior de um agente nominal, mas isso não irá ser levado em consideração por este agente nominal quando for verificar se já está ligado a algum agente. Ou seja, este agente nominal deverá possuir um

---

[53] Não foi tratado o caso para objeto indireto, mas este não traria dificuldade porque ou ele seria um dos casos de flexibilização das posições do agente verbal ou seria um caso de busca. O que ocorreria é o seguinte: o agente pronome relativo iria verificar a existência do agente preposicional a sua esquerda e se ligaria a ele. O agente preposicional é que teria que tentar se ligar, seguindo o esquema de ver se houve mensagem global de algum agente verbal (e aqui é que entraria a flexibilização, pois o agente estaria numa posição anterior a do agente verbal) ou fazendo a busca (agora levando em consideração que ele tem de procurar à direita, e não mais à esquerda).

outro superior (seja verbal ou preposicional), que será chamado de "superior principal" (cf. nota de rodapé 44 e seção 4.4.2.2). Assim, nas figuras que apresentarem agente pronome relativo, a ligação entre ele e o agente nominal (seu subordinado) será marcada de maneira especial, como pode ser visto na Figura 14 para a sentença "o menino que ama a menina caiu".

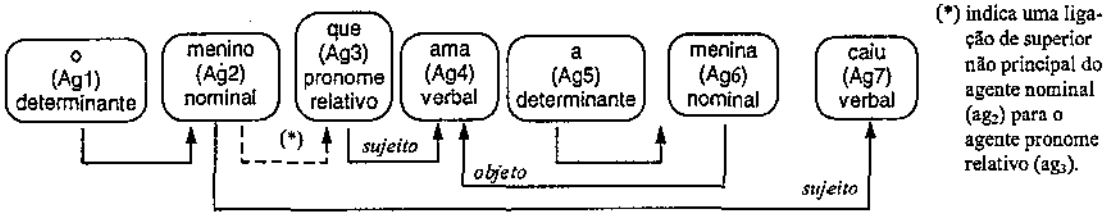

Figura 14 — Representação da ligação entre um agente pronome relativo e um nominal.

4.5.5 Coordenação

O interesse apontado na seção 3.2.5 à página 3-31 para a utilização de um agente coordenação consistia na possibilidade de o agente ter o comportamento de mais de uma classe (no caso, poder se comportar ou como agente nominal ou como verbal). Para saber qual comportamento utilizar, ele tem que verificar quais são seus vizinhos e decidir com quais deles ele deve se ligar. Essa fase de busca é a mais importante para não haver necessidade de retorno não necessário nas ligações que forem estabelecidas (ou seja, para não haver necessidade de quebra de ligações).

A estratégia de busca utilizada pelo agente coordenação consiste primeiramente na verificação da existência de agentes pronome relativo na sentença. Se houver, o agente coordenação estará coordenando agentes nominais e então ele terá que decidir quais agentes são esses. No exemplos abaixo, há duas possibilidades, que podem ser vistas na Figura 15.

(27) João que ama Paula e Malu caiu.

(28) João que ama Paula e Malu caíram.

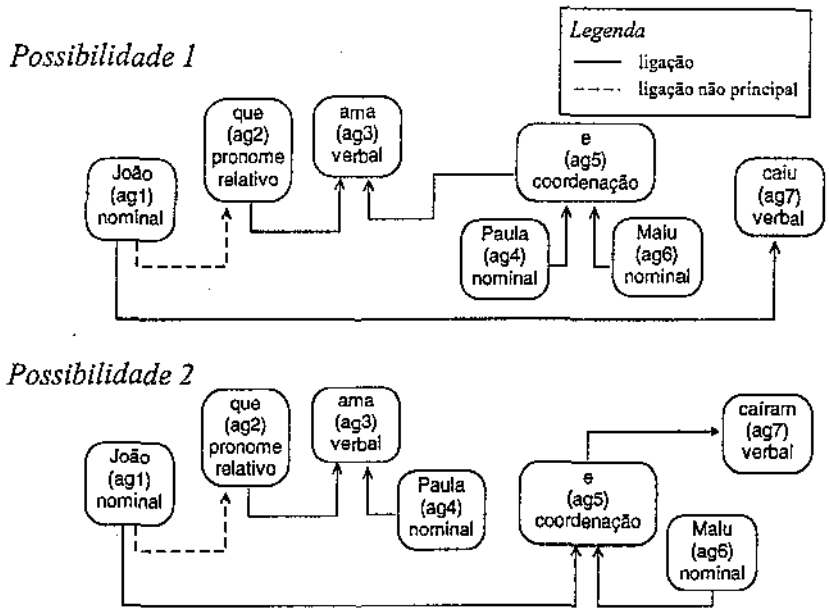

Figura 15 — Possibilidades de coordenação em sentenças com presença de relativos.

Se não houver agente pronome relativo, então o agente coordenação irá verificar a existência de agentes verbais em cada um dos seus lados. Se houver apenas um agente verbal e se este estiver a sua direita, então o agente estará coordenando agentes nominais. Se o agente verbal estiver a sua esquerda há duas possibilidades:

1. está ocorrendo "*gapping*" (ver a próxima seção) e o agente estará coordenando agentes verbais; ou
2. não está ocorrendo "*gapping*" e o agente estará coordenando agentes nominais.

Se houver mais de um agente verbal, então o agente estará coordenando agentes verbais. Exemplos de sentenças envolvendo esses últimos quatro casos são:

(29) João e Antônio caíram. [apenas um verbo à direita]

(30) João ama Paula e Marcos, Malu. [apenas um verbo à esquerda com "*gapping*"]

(31) João ama Paula e Marcos. [apenas um verbo à esquerda sem "*gapping*"]

(32) João ama Paula e ama Malu. [dois verbos]

Se o agente coordenação estiver ligando dois agentes verbais ele terá terminado seu papel. Se estiver ligando dois nominais, então terá ainda que ter um superior. Sua representação in-

terna após a ligação estará refletindo a união realizada, ou seja, ele terá seu parâmetro "número" marcado como plural, seu "gênero" será masculino se um dos seus subordinados o for (feminino, caso contrário) e, como "tipo semântico", ele terá aquele que comporte o tipo de cada um dos subordinados e que seja, ao mesmo tempo, o mais especializado possível[54].

4.5.6 Redução e "*Gapping*"

Os problemas gerados por sentenças com "*gapping*" ou redução (cf. seções 3.2.6 e 3.2.7) dizem respeito à: (1) detecção das situações que caracterizam os dois problemas e (2) reconstrução das partes faltantes na sentença. A ordem de apresentação será invertida para facilitar a exposição da parte de detecção.

4.5.6.1 *Reconstrução*

Há duas maneiras de lidar com a falta de um agente. Por exemplo, a sentença

(33) João ama e Pedro detesta sorvete.

poderia ser representada de duas maneiras, conforme mostram as Figura 16 e Figura 17.

---

[54] Por exemplo, se a base de conhecimento possui os seguintes tipos semânticos mortal, humano, homem e mulher, e tem-se estabelecido as seguintes relações do tipo isa "humano isa mortal", "homem isa humano" e "mulher isa humano", então se a coordenação for entre um agente do tipo homem e outro do tipo mulher, o tipo semântico que será associado à coordenação será o tipo humano, que é mais especializado que mortal, por exemplo.

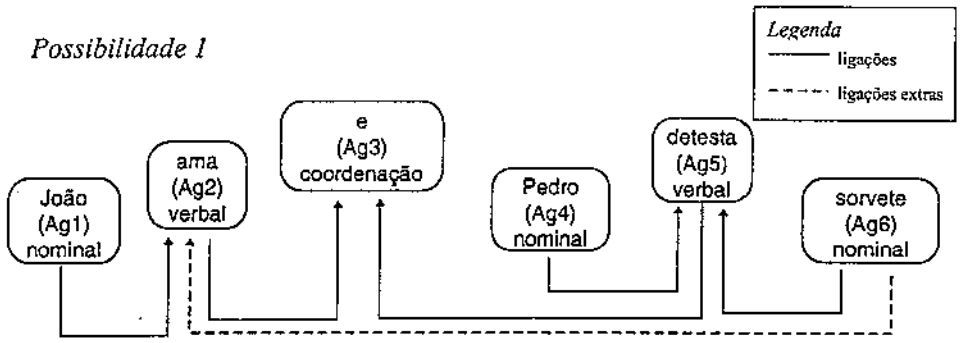

Figura 16 — Reconstrução com duplicação de ligação.

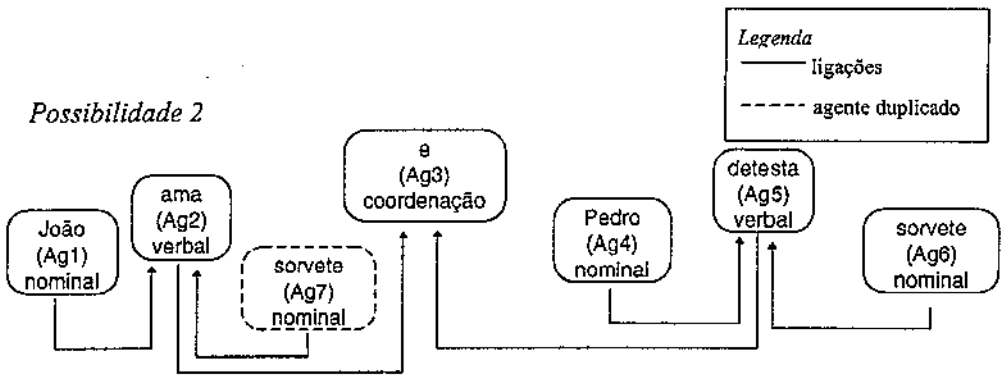

Figura 17 — Reconstrução com duplicação de agente.

Para os agentes nominal e verbal não haveria dificuldade de gerenciar uma nova ligação do tipo visto na Figura 16. Para uma sentença como

(34) João ama Maria e Daniel, Mônica. ,

a dificuldade para o agente verbal gerenciar outras ligações seria de extremo custo (pois ele estaria gerenciando duas ligações para sujeito, duas para objeto etc.), o que resultou na adoção da estratégia de duplicação de agentes, como mostrado na Figura 17. Outro problema para a duplicação de ligações ocorreria com o agente coordenação (ver Figura 18). Este teria que possuir duas ligações vindas de um mesmo agente (pois ele estaria coordenando dois agentes verbais, representados pelo mesmo agente), o que poderia ser resolvido por diferenciação de tipos de ligação; entretanto, essa seria uma complicação desnecessária, visto que nada disso é preciso quando se faz a duplicação de um agente.

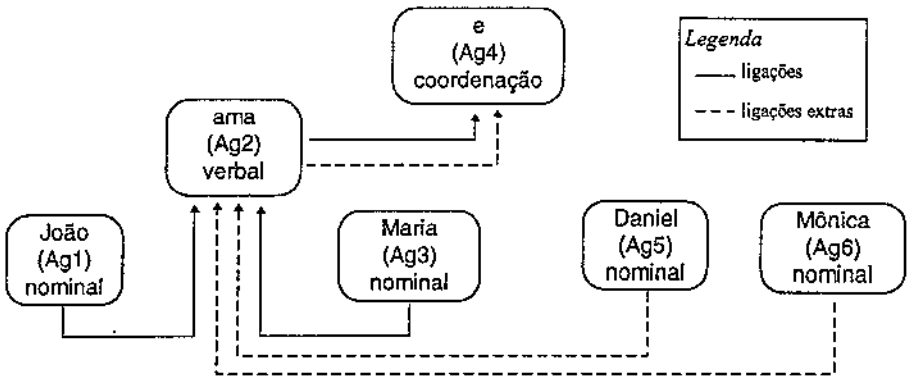

Figura 18 — Duplicação de ligações para substituir um agente verbal.

4.5.6.2 *Detecção*

Para o caso de "*gapping*", a detecção pode ser feita tanto pelo agente coordenação (como visto na seção 4.5.5) quanto pelos agentes nominais. Para estes (agentes nominais), a situação pode ser visualizada em termos da necessidade de estar ligado e não conseguir, seja porque não houve pedidos por parte de agentes verbais, seja porque esses pedidos não resultaram em ligação.

Foi visto na seção 4.4.1.1 que, a princípio, os agentes nominais têm um comportamento passivo, ou seja, eles esperam que algum outro agente tente estabelecer ligação com eles. Entretanto, quando isso não ocorre, é preciso que o agente tome a iniciativa de ver o que pode estar ocorrendo. E é dessa maneira que ocorre a detecção da situação de "*gapping*": o agente nominal não conseguiu se ligar ainda e então toma a iniciativa de verificar quais são seus vizinhos.

Tomando como exemplo a sentença (34) e focando atenção no agente Daniel (ver o agente em destaque na Figura 19), então, se há um agente coordenação a sua esquerda e um agente nominal a sua direita, o agente Daniel irá encontrar o agente verbal que está antes do agente coordenação e pedir sua reconstrução de acordo com o que foi exposto na seção anterior.

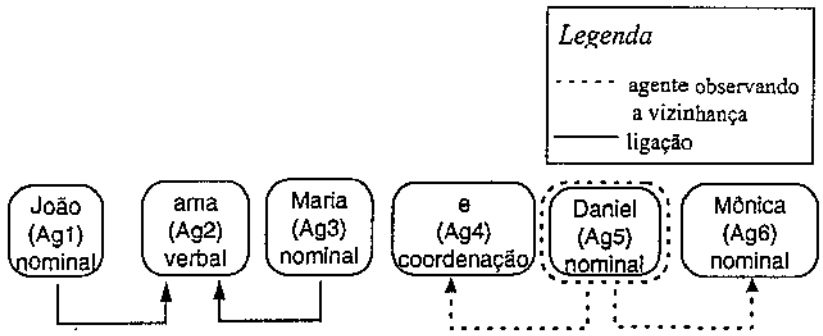

Figura 19 — Detecção da situação de "*gapping*".

Assim, uma cópia do agente ama (ag_2) sem suas ligações será criada na posição entre os agentes Daniel (ag_5) e Mônica (ag_6) e passará a funcionar do mesmo modo como um agente verbal em seu funcionamento normal agiria: enviará mensagens globais para todas suas subcategorizações obrigatórias, esperará as propostas, e estabelecerá ligações com os agentes que enviarem as melhores propostas. A estrutura final para a sentença (34) pode ser vista na Figura 20.

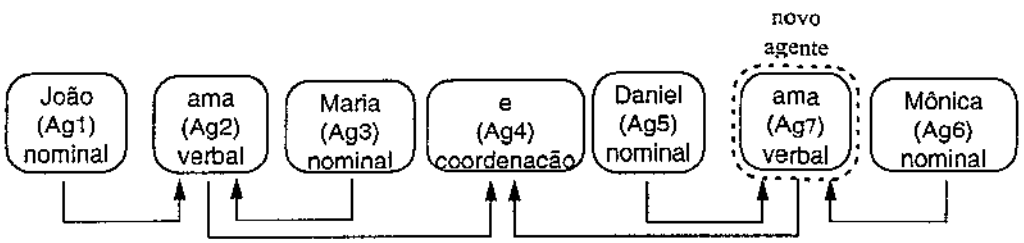

Figura 20 — Representação final para a sentença com "*gapping*".

Na situação de *redução*, a detecção é feita pelo agente verbal. Na sentença

(35) João comprou bala e pagou Mário ,

o agente verbal pagou não terá conseguido fazer ligação com nenhum agente para a posição de sujeito, tentará flexibilizar (cf. seção 4.5.2) e, mesmo assim, não conseguirá encontrar o sujeito. Assim, ele irá partir para uma estratégia direta da verificação de sua vizinhança para ver se existe um agente coordenação a sua esquerda. Se existir, ele tentará encontrar o agente verbal que estiver antes desse agente coordenação para saber qual o sujeito deste agente verbal.

Encontrado esse agente nominal, que é o sujeito da primeira oração, o agente verbal irá verificar as informações desse agente e pedir que uma cópia sua seja feita na posição que ele (agente verbal) esperaria para seu sujeito. Então ele estabelecerá ligações com esse novo agente criado. Esses passos podem ser visto na Figura 21, onde está representado o agente verbal (ag_5) dialogando com seus vizinhos (a ordem não está expressa), e na Figura 22, onde o agente nominal já foi duplicado (ag_7) e já está ligado ao agente verbal (ag_5).

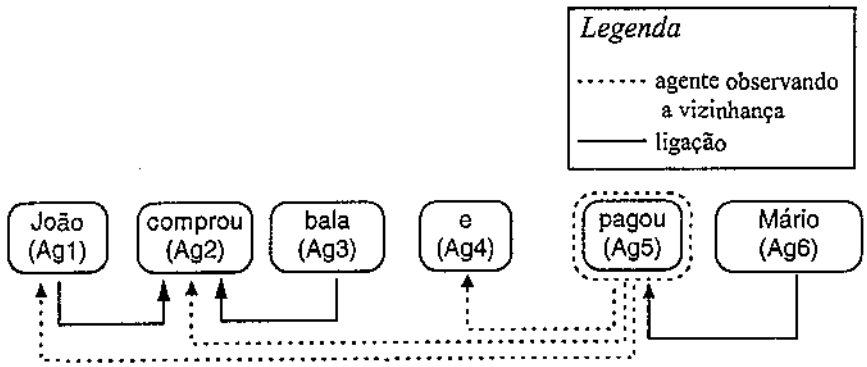

Figura 21 — Detecção da situação de redução.

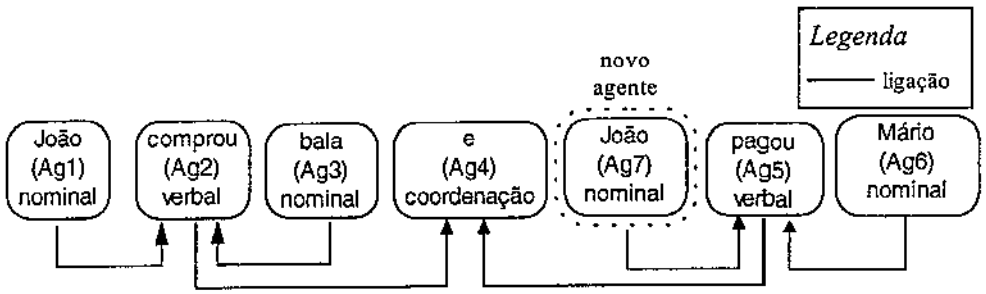

Figura 22 — Representação final para a sentença com redução.

4.5.7 Possibilidade de Quebra de Ligações

Na seção 4.4.2.3 (à página 4-48) foi dito que havia a possibilidade de os agentes poderem estabelecer ligações erradas de um ponto de vista global, apesar de localmente (ou seja, para os agentes) elas estarem corretas. Nesta seção será visto quando é que isso pode ocorrer e como essa situação é resolvida. Por exemplo, na sentença

(36) O menino que ama a menina caiu.

quando os agente verbais ama e caiu enviam suas subcategorizações, o agente nominal menina irá responder tanto para um (ama), candidatando-se a objeto direto, quanto para outro (caiu), candidatando-se a sujeito.

Assim, dependendo da velocidade de processamento dos dois agentes verbais, pode ser que um ou outro consiga estabelecer ligação com o agente nominal menina. Supondo que o agente caiu tenha sido mais rápido que o agente ama e, além disso, que o agente nominal menino tenha sido mais rápido que o agente pronome relativo que, então a situação é a que pode ser vista na Figura 23.

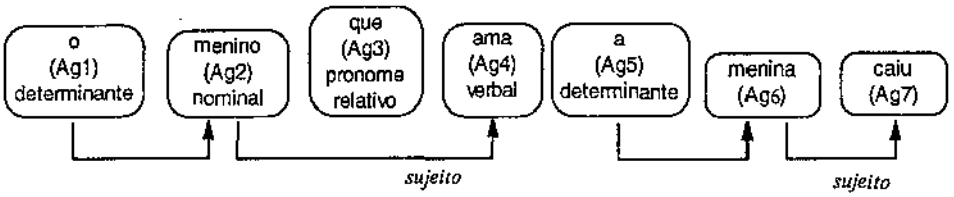

Figura 23 — Exemplo de situação que resultará em conflito.

Situações como essa são modificadas utilizando *quebra de ligações*. A execução desse processo em que são desfeitas ligações será iniciada pelo agente que (Ag_3). Este agente irá verificar (observando que não há agente nominal entre ele e o agente verbal) que deverá ocupar a posição de sujeito no agente ama (Ag_4) e pedirá para que este agente (Ag_4) quebre sua ligação com o agente que é o sujeito na situação atual (Ag_2).

O agente que também pedirá que o agente verbal (Ag_4) forneça as informações acerca dessa posição (sujeito) para que ele (agente que) possa encontrar o agente nominal (a sua esquerda) para essa posição. A nova configuração, com o agente que já tendo encontrado o agente nominal menino para sujeito do agente ama, pode ser vista na Figura 24.

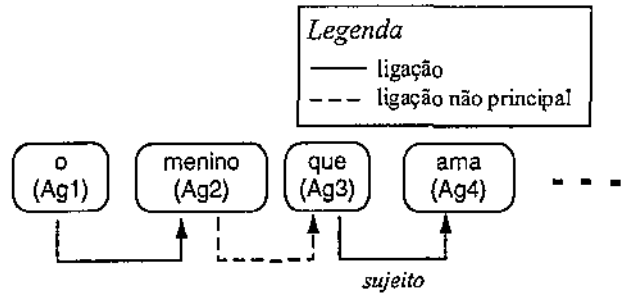

Figura 24 — Aspecto parcial resultante da quebra de ligações.

Com relação à posição de objeto, o agente verbal ama (ag_4), verificando que só o agente nominal menina (ag_6) pode preenchê-la e que ele (ag_6) está ligado ao agente verbal caiu (ag_7), pede a este (ag_7) que quebre suas ligações com aquele (ag_6). Deve-se dizer que isto é uma decisão *puramente local* para o agente (ag_4) que não vê outra solução disponível. Quando a quebra acontece, o agente ama imediatamente estabelece ligação com o agente menina.

Quando ocorre uma quebra deste tipo, em que o agente que faz o pedido (ag_4) não irá ocupar a posição no agente que recebeu o pedido, então este agente (no caso, ag_7) terá que enviar requisições (em mensagens globais) para esta posição novamente. Com isso, os agentes nominais que estiverem sem ligação irão responder a esse novo pedido, e é isso o que o agente nominal menino (ag_1) faz. O processo continua até que todas as ligações tenham sido feitas. A representação final para a sentença pode ser vista na Figura 25.

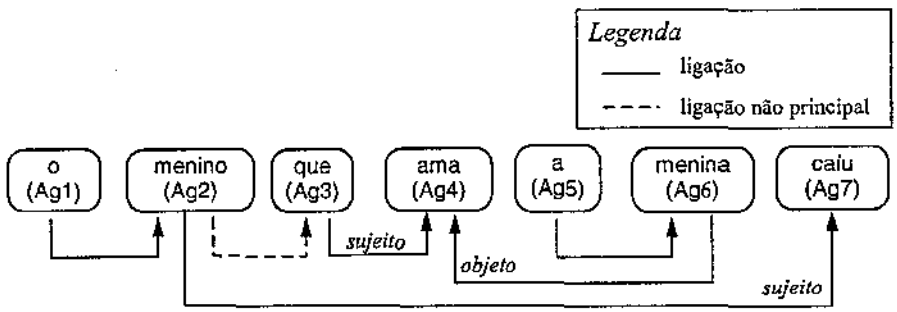

Figura 25 — Situação final após a resolução de alguns conflitos.

4.6 Detecção da Gramaticalidade e da Agramaticalidade de uma Sentença

Na seção 4.4.2.2 (à página 4-47) foi apresentada a importância que as ligações têm para os agentes: *um agente sem ligação é como uma palavra que não tem função em uma sentença* Foi utilizando esta metáfora que se definiu um estado para a percepção do término da execução do sistema. Quando nenhum agente consegue realizar mudanças em seu estado e ainda há pelo menos um agente que não fez as ligações necessárias, então a sentença é considerada *agramatical.* Ou seja, cada agente já utilizou todas as suas estratégias para tentar estabelecer ligações, mas ou não encontrou nenhum agente ou seu pedido de estabelecimento de ligação não foi aceito. Quando todos os agentes conseguem estabelecer suas ligações, então a sentença é considerada *gramatical.* Quando uma dessas duas situações é observada, o sistema pára. Enquanto isso não ocorre, ele se mantém funcionando.

Essa idéia pode ser pensada para um modelo distribuído utilizando quatro tipos de mensagens: uma para "término", outra para "espera", outra para "estou perdido" e uma última para o agente tornar a "verificar" o ambiente.

A mensagem de "*término*" é utilizada quando o agente consegue estabelecer as ligações necessárias e dá por encerrada sua execução[55]. Esse tipo de mensagem é utilizada apenas pelos agentes verbais em vista de terem a noção de completude necessária para a análise de uma sentença. Ou seja, quando um agente verbal estiver com suas subcategorizações preenchidas, o sentido essencial de sua sentença estará formado. O fato de o agente verbal ter terminado não significa que todos os outros agentes estejam ligados; pode haver agentes que ainda não terminaram e por isso é necessário haver outros tipos de mensagens.

---

[55] Entretanto, ele ainda poderá responder a perguntas vindas de outros agentes.

Quando algum agente enviou uma mensagem de término, pode ocorrer de existir algum agente que ainda não conseguiu estabelecer suas ligações necessárias, mas que ainda não esgotou todas as suas possibilidades neste sentido. A mensagem de "*espera*" é utilizada por esse agente nessa situação para que o término do sistema não seja anunciado sem que ele esteja ligado ou tenha esgotado suas possibilidades. Quando consegue se ligar, ele informa diretamente para o agente que enviou a mensagem de término que, então, pode anunciá-lo novamente. Se houver vários agentes que pediram para esperar, esse agente do término terá que esperar até que todos eles avisem que terminaram.

A mensagem "*estou perdido*" é utilizada quando um agente já esgotou todas as suas estratégias para tentar se ligar e não conseguiu. Em um dado momento pode ser que existam vários agentes nessa situação e, mesmo assim, pode ser que haja uma solução para o sistema. Pode ser que um agente, que ainda não esgotou suas possibilidades, faça uma modificação no ambiente que repercuta em vários agentes e isso pode alterar toda a configuração do sistema. A mensagem "*verifique o ambiente*" é emitida por um agente que conseguiu avançar em alguma direção após ter escutado uma mensagem "*estou perdido*".

Essa idéia de mensagens para controlar o término pode ser gerenciada de duas maneiras: centralizada ou distribuída. Na *versão centralizada*, um agente é escolhido para ficar monitorando essas mudanças e fica responsável por indicar se houve término por um dos seguintes motivos: (1) porque todos os agentes conseguiram se ligar — quando então a sentença é considerada *gramatical*; ou (2) porque não houve nenhum progresso e ainda há pelo menos um agente sem conseguir se ligar — quando a sentença é considerada *agramatical*.

Na *versão distribuída*, todos os agentes têm que ficar verificando as mudanças no ambiente, os avanços dos agentes que pediram "*espera*" e dos que anunciaram "*estou perdido*". Da mesma forma que na versão centralizada, a resposta teria de vir da observação dos dois motivos para término do funcionamento do sistema (itens 1 e 2 expostos para a versão centralizada).

4.7 Considerações Finais sobre o Capítulo

Neste capítulo foram apresentadas as características da parte conceitual do sistema. Partiu-se de um modelo de processamento lingüístico inicial (modelo de *forças e restrições*), apresentou-se um modelo de interpretação para essas idéias de processamento lingüístico que se adequa ao processamento distribuído em agentes (com algumas características de Sistemas Multi-Agentes) e mostrou-se exemplos de resolução de determinados fenômenos lingüísticos que exercitam as características do modelo de interpretação.

Com relação às estratégias de busca utilizadas pelos agentes no tratamento dos vários fenômenos lingüísticos, pôde-se ver que elas não esgotam as frases que podem ser criadas com as classes de palavras com que o sistema trabalha (isto é, com os tipos de agentes do sistema). No Capítulo 5 serão apresentadas as características da implementação do sistema. No capítulo 6 serão apresentados alguns exemplos de sentenças que o sistema não consegue analisar e outras estratégias serão esboçadas com o objetivo de delinear possíveis extensões para os tipos de agentes. Além disso, serão apresentados os resultados obtidos com a realização deste projeto, principalmente no que se refere ao tipo de agente obtido na construção do sistema.

5. *Aspectos da Implementação*

Quem tem a viola pra se acompanhar
não vive sozinho, nem pode penar.[56]

A implementação do sistema foi feita de modo a *simular* um sistema distribuído. A principal idéia é manter o não sincronismo do funcionamento dos agentes, o que é realizado utilizando-se uma função que gera uma ordem aleatória em que os agentes do sistema são acordados (isto é, postos em funcionamento).

O esquema geral utilizado para o processamento do sistema pode ser visto na Figura 26 abaixo e consiste em:

- uma primeira parte onde são feitas as inicializações de todas as informações que os agentes e o sistema precisam para poder começar a funcionar;
- uma segunda parte onde os agentes realizam suas interações;
- uma terceira parte onde os resultados são apresentados.

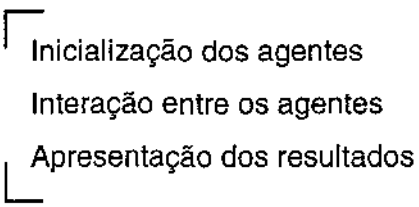

Figura 26 — Esquema geral do processamento.

[56] Zé Renato, Juca Filho, Cláudio Nucci e Xico Chaves na música "Quem Tem a Viola".

Das fases apresentadas no esquema geral acima, as duas primeiras serão apresentadas em detalhes a seguir; a última fase não possui nenhuma característica especial e, portanto, não será apresentada.

5.1 Fase de Inicialização

A inicialização pode ser dividida em duas partes, uma relacionada à inicialização dos agentes e outra, à inicialização das estruturas que controlam a parte de interação dos agentes. Essas partes serão apresentadas a seguir.

5.1.1 Informações dos Agentes

Em termos da estrutura de dados, cada agente mantém seu *identificador*, as *informações vindas do dicionário*, seu *significado final* (no caso de o dicionário especificar mais de um sentido para a palavra que o agente representa) e suas *ligações* (superior e subordinados). Estas informações ficam encapsuladas no próprio agente.

Na fase de inicialização, à medida que cada palavra da sentença que irá ser processada é lida, um agente correspondente à categoria dessa palavra é instanciado e abastecido com as informações do dicionário. Além disso, um identificador exclusivo é atribuído a esse agente e suas ligações — assim como seu significado final — são inicializadas com o valor "nulo".

Em termos do conhecimento de vizinhança, a idéia foi manter cada agente ciente apenas da sua vizinhança mais próxima devido à possibilidade de as mudanças causadas com a introdução de novos agentes tornarem o gerenciamento dessa informação um fator oneroso. O que se fez foi pensar no conjunto dos agentes como um encadeamento do modo como pode ser visto na Figura 27 abaixo.

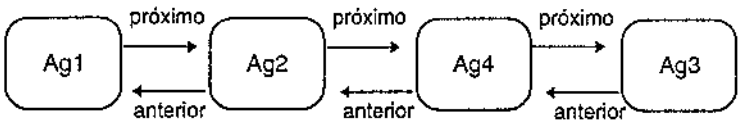

Figura 27 — Esquema de encadeamento dos agentes.

Em realidade, as informações de vizinhança e posição (que são utilizadas quando um agente está fazendo a busca por outro agente) ficaram centralizadas em uma estrutura de dados à parte (que pode ser pensada como um "agente" diferente) que ficou responsável por controlar a inserção de novos agentes (atualizando os apontadores etc.) e por responder a perguntas como "qual o identificador do agente de tal posição" ou "qual o identificador do próximo agente, dado tal identificador" etc.

Com relação aos tipos de agentes, apenas o agente verbal possui uma diferenciação na fase de inicialização que está relacionada a suas subcategorizações obrigatórias. Para cada um dos possíveis significados das subcategorizações obrigatórias de um verbo (ver seção 4.5.1.1 à página 4-50), é colocada em um espaço global uma mensagem que segue o esquema apresentado na página 4-51 (mensagem "preciso").

Não foi utilizado nenhum tipo de mecanismo de *memória das ações* que o agente já realizou. Entretanto pode ser que seja necessária a utilização de uma lista das ligações já realizadas e quebradas para evitar possíveis oscilações que, embora ainda não tenham sido detectadas, possam ocorrer devido à possibilidade de quebra de ligações. Isso ficará para uma próxima etapa do trabalho.

5.1.2 Informações Extra-Agentes

As principais informações extra-agentes são aquelas responsáveis pelo controle do término do "*loop*" de interação dos agentes: a lista viu_término e a lista não_fez_nada (seu uso na seção 5.2.3). Estas listas têm seu conteúdo inicializado como lista vazia.

5.2 Interação dos Agentes

Para a simulação do funcionamento do sistema como se este fosse distribuído, foi criado um "*loop*" no qual os agentes são *acordados* seguindo o resultado de uma função que gera o identificador de um agente de forma aleatória até que um dos seguintes critérios de parada seja alcançado:

1. todos os agentes conseguiram se ligar e a frase é considerada gramatical; ou
2. ocorreu um ciclo completo em que todos os agentes foram ativados, nenhum deles conseguiu avançar em alguma direção e ainda há agentes sem ligação. Assim, a frase é considerada agramatical.

Esse esquema pode ser visto na Figura 28.

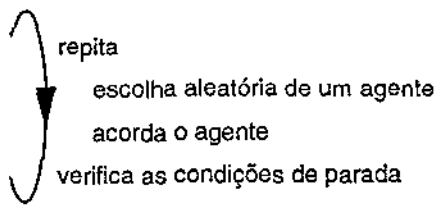

Figura 28 — Esquema de ativação dos agentes.

5.2.1 Características Gerais do Procedimento "Acorda"

É no procedimento "acorda", visto na Figura 28, que está inserido o comportamento dos agentes descrito na seção 4.4.1.1; dessa forma, para cada tipo de agente, existe um tipo de acorda. Entretanto, todos esses acordas seguem um esquema semelhante:

1. o agente verifica sua necessidade de ainda estabelecer ligações;
2. se houver ligações a serem feitas, o agente tenta encontrar o(s) agente(s) e estabelecer a(s) ligação(ões); e
3. se não houver (mais) ligações a serem feitas, o agente verifica se ele precisa se posicionar com relação ao término do programa (cf. seção 4.6).

O modo como esse procedimento acorda foi planejado levou em consideração a necessidade de o agente interagir com os outros agentes e, como não é uma implementação em paralelo, a necessidade de o agente passar o controle para que os outros agentes também possam ser executados.

O que se tentou com isso foi simular o que de modo distribuído e paralelo seria: o agente ficaria monitorando os acontecimentos no ambiente (causados tanto por suas ações quanto pelas ações dos outros agentes) e interagiria com os outros agentes de acordo com seu comportamento e com esses acontecimentos. Com a implementação realizada, o que se tem estabelecido é que, de tempos em tempos, cada agente é acordado, tem uma chance de satisfazer suas necessidades (em termos de ligações) e de interagir com outros agentes.

5.2.2 Implementação da Troca de Mensagens

O relacionamento entre agentes é feito por meio de envio de mensagens. Nessa simulação foram utilizados dois modos de trocar mensagens (de acordo com o tipo de endereçamento utilizado): individual ou global. As *mensagens individuais* foram implementadas como uma chamada de procedimento, ou seja, um agente A quer saber uma informação sobre um agente B e lhe envia uma mensagem direta. Com isso, o agente A pára seu processamento e o contexto é mudado para o processamento do agente B. Este então executa o procedimento que irá resultar na resposta para a mensagem do agente A e, quando este procedimento termina, o agente B retorna uma mensagem com a resposta e o controle da execução para o agente A. Esse esquema pode ser visto na Figura 29.

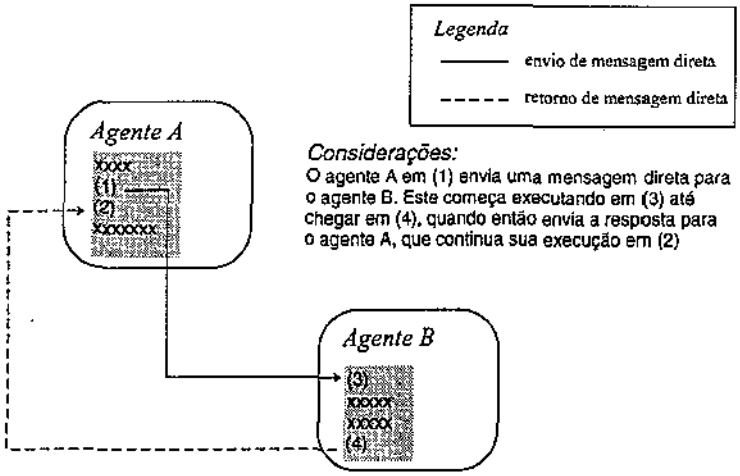

Figura 29 — Simulação de mensagem direta do Agente A para o Agente B.

As *mensagens globais* foram implementadas com a utilização de memória compartilhada no sentido de que todos os agentes tenham a possibilidade de lê-las. Assim, quando um agente A envia uma mensagem global, na realidade ele está colocando uma mensagem em uma área global que pode ser lida por qualquer agente quando este é acordado.

Dois tipos de mensagens globais estão sendo utilizados efetivamente: o primeiro diz respeito às mensagens "preciso" enviadas pelos agentes verbais para suas subcategorizações obrigatórias e o segundo, aos "*flags*" que sinalizam[57] alguns fatos como "a presença de pronomes relativos", "a descoberta de uma situação de *gapping*" etc.

Uma mensagem global do tipo "*flag*" não é para ser respondida — o agente apenas fica ciente de uma determinada situação. Já uma mensagem global "preciso" pode ser respondida por vários agentes (no caso específico, candidatando-se a uma posição) e, para essa mensagem de retorno, foi implementado um tipo de "caixa postal" onde os agentes podem depositar suas propostas que então serão lidas pelo agente que enviou a mensagem global. Esse esquema pode ser visualizado na Figura 30.

---

[57] [Small80] utiliza uma idéia semelhante que chama de "*control signals*" (ver página 2-16 desta dissertação) para casos de voz passiva.

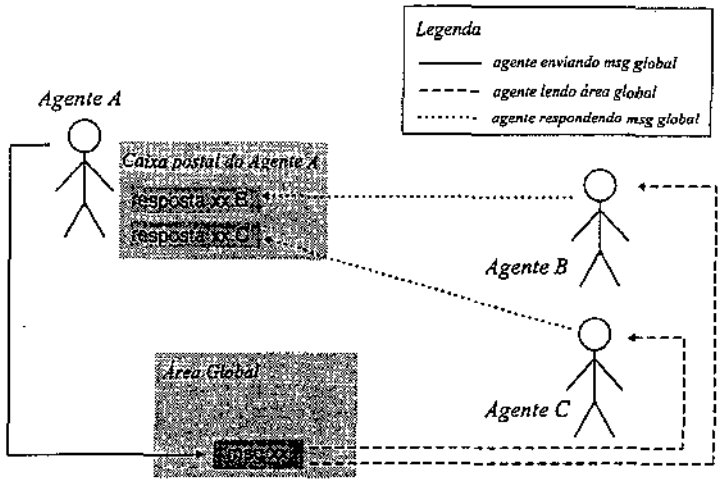

Figura 30 — Simulação de mensagem global.

5.2.3 Monitoramento do Término

O término do sistema é verificado a cada ciclo do "*loop*" de processamento (Figura 28, linha "verifica as condições de parada") simulando um gerenciamento centralizado (cf. seção 4.6).

Quando um agente verbal anuncia seu "término", essa mensagem é colocada em uma área global para que todos os outros agentes possam lê-la. O monitoramento do término é feito com duas listas: uma que vai armazenando o identificador de todos os agentes que viram o anúncio de término feito por um agente verbal (chamada de "lista viu_término") e outra que armazena o identificador dos agentes que, quando foram ativados, não realizaram nenhuma mudança em sua situação (chamada de "lista não_fez_nada").

A atualização dessas listas é feita da seguinte maneira. Quando um agente consegue estabelecer suas ligações, ele verifica se existe alguma mensagem de término e, se houver, ele assinala que viu o término inserindo seu identificador na lista viu_término. Toda vez que um agente realiza qualquer operação, mudando sua situação, a lista não_fez_nada é esvaziada, indicando que houve uma mudança na configuração do sistema.

Quando uma das duas listas está cheia o sistema é parado. Se for a lista viu_término, é porque todos os agentes viram uma mensagem de término (colocada por um determinado agente) e

estavam ligados. Se for a lista não_fez_nada, é porque houve um ciclo em que todos os agentes foram ativados pelo menos uma vez e não conseguiram realizar uma operação que mudasse a configuração do sistema.

Das mensagens apresentadas na seção 4.6 ("término", "espera", "estou perdido" e "verifique o ambiente"), apenas a de "término" é representada explicitamente neste modelo simulado, com a inicialização da lista viu_término, caso ela ainda não exista. A mensagem de "espera" pode ser observada no fato de nenhum agente assinalar que viu uma mensagem de término antes de ter feito as ligações necessárias ao mesmo tempo em que, conseguindo progredir, pede o esvaziamento da lista não_fez_nada.

A mensagem "estou perdido" é caracterizada pela inserção do identificador do agente na lista não_fez_nada[58]. A mensagem "verifique o ambiente" está totalmente implícita no funcionamento dos agentes e no esvaziamento da lista não_fez_nada. O fato de esvaziar a lista não_fez_nada possibilita que todos os agentes continuem tentando progredir. A lista cheia sinaliza um ciclo em que todos os agentes foram ativados e não realizaram mudança alguma em suas configurações.

5.3 Os Procedimentos "Acorda"

Nesta seção serão detalhados os esquemas do procedimento "acorda" para os seguintes tipos de agentes: nominal, verbal, determinante, pronome relativo e coordenação.

5.3.1 Acorda para Agente Nominal

Há duas situações tratadas pelo "acorda" do agente nominal. A primeira está relacionada ao comportamento normal do agente que se caracteriza por uma posição passiva, ou seja, o agente

---

[58] O que não quer dizer que quem tenha colocado seu identificador na lista não_fez_nada esteja perdido, isto é, não conseguiu se ligar.

responde às mensagens "preciso" enviadas por algum agente verbal ou o agente fica esperando que algum outro agente o encontre e peça para estabelecer ligações.

```
Acorda_Nominal (Situação Normal)

    verifica a não existência do flag de gapping
    verifica a necessidade de ligação
    se houver necessidade
        verifica se há msgs "preciso"
        (   se não há "preciso" algum
            (   força uma "espera"
                (ou seja, vai dormir)
            ;   verifica condições de gapping
                envia msg global flag de gapping
                assinala que "fez_algo"
                    (usado para não incrementar a lista "não_fez_nada")
            )
        ;   se há "preciso"
            envia "propostas" para aqueles em que os requisitos são satisfeitos
            assinala que "fez_algo"
        )

    se não houver necessidade de ligação
        trata msgs de "término", inserindo seu identificador
            listas "viu_término" que existirem

    se existe algum "fez_algo"
        apaga todas as ocorrências desse flag
    se não existe nenhum "fez_algo"
        incrementa a lista "não_fez_nada" com o identificador deste agente

fim.

Acorda_Nominal (Situação de Gapping)

    verifica a existência do flag de gapping
    verifica a necessidade de ligação
    se houver necessidade
        procura o agente coordenação e o 1º agente verbal à esquerda desse agente
        verifica sua própria posição em relação ao agente coordenação
        (isto e', verifica se há um outro nominal entre o coordenação e ele próprio)
        pede a criação do novo agente verbal baseado no agente verbal encontrado
            na posição verificada (o ag. verbal ficará entre dois ag. nominais)
        assinala que "fez_algo"

    se não houver necessidade de ligação
        trata msgs de "término", inserindo seu identificador nas
            listas "viu_término" que existirem

    se existe algum "fez_algo"
        apaga todas as ocorrências desse flag
    se não existe nenhum "fez_algo"
        incrementa a lista "não_fez_nada" com o identificador deste agente
    apaga o flag de gapping

fim.
```

5.3.2 Acorda para Agente Verbal

O esquema para os agentes verbais é o mais elaborado e é baseado nas subcategorizações obrigatórias. O agente, que na inicialização enviou mensagens globais "preciso" para cada uma delas, irá verificar se já possui essas subcategorizações preenchidas (como subordinados) e, caso não possua, irá verificar a existência de "propostas" de candidatos para cada uma delas. Caso as propostas não resultem em nenhum agente tentando estabelecer ligações, então começam a ser utilizadas algumas estratégias particulares de acordo com a subcategorização que se está verificando. No esquema abaixo, apenas a verificação da parte do sujeito será apresentada e as estratégias tentarão primeiramente "flexibilizar" (ver 4-55) a posição sujeito e, se não der certo, em seguida será verificada a possibilidade de a frase ter sofrido "redução" (ver seção 4.5.6 à página 4-62).

```
Acorda_Verbal

    verifica a necessidade de sujeito
    se ainda não existe sujeito
        apura as propostas para sujeito( MelhorSujeito )
        se existe ( MelhorSujeito )
            verifica a existência de 1 ag. coordenação entre o MelhorSuj e o ag. verbal
            se não existe
                tenta estabelecer ligação( ok )
                se ok
                    assinala "fez_algo"
                senão
                    retira proposta ( MelhorSujeito )
                    assinala "fez_algo"
        se não existe ( MelhorSuj ) e verbo não subcategoriza objeto direto
        (   se ainda não flexibilizou
                flexibiliza ( sujeito )
                assinala "flexibilizou_suj"
                assinala "fez_algo"
        ;   se só flexibilizou a primeira vez
                /* verificando a existência do flag "flexibilizou_suj" e
                   a ausência do flag "flexibilizou_2_suj" */
                se a sentença possui pronome relativo
                    /* tenta buscar o sujeito na vizinhança */
                    procura agente nominal à esquerda que será aqui chamado de "vizSuj"
                    se existe esse "vizSuj"
                        obtém o superior do "vizSuj", que aqui será chamado de "supVizSuj"
                        se "supVizSuj" não está em uma posição entre o agente verbal
                                e o "vizSuj"
                            pede a quebra de ligações de "vizSuj" e "supVizSuj"
                            o agente verbal tenta estabelecer ligação com "vizSuj" como
                                sujeito
                            assinala "fez_algo"
                    assinala "flexibilizou_2_suj"
                    assinala "fez_algo"
        ;   se já flexibilizou a segunda vez (verificando o flag "flexibilizou_2_suj")
```

```
            verifica a possibilidade de estar havendo "redução" (ou seja, a frase
                é coordenada e não há pendência para objeto direto)
            se existe a possibilidade
                procura o sujeito da frase anterior
                se existe o sujeito
                    pede a criação de um novo agente nominal
                    estabelece ligação com esse novo agente como sujeito
                    assinala "fez_algo"
                se não existe
                    não faz nada (ou seja, vai dormir)
        )
    se existe sujeito
        não faz nada (ou seja, vai dormir)

..... o código para objeto direto e indireto segue basicamente o esquema acima .....

    se não há mais necessidade de sujeito, objeto e objeto indireto
        se o agente verbal não possui um pronome relativo como subordinado
            o agente verbal assinala "terminei", criando uma lista "viu_término"
                com a inserção de seu próprio identificador nessa lista
        se o agente verbal possui um pronome relativo como subordinado
            trata msgs de "término", inserindo seu identificador nas
                listas "viu_término" que existirem

    se existe algum "fez_algo"
        apaga todas as ocorrências desse flag
    se não existe nenhum "fez_algo"
        incrementa a lista "não_fez_nada" com o identificador deste agente

fim.
```

5.3.3 Acorda para Agente Determinante

O procedimento para o agente determinante é, sem dúvida, o mais simples de todos os realizados. A idéia é que um agente determinante irá encontrar o primeiro agente nominal que estiver à sua direita e estabelecerá ligação com ele. Caso não consiga, é porque existe um erro. O código segue o esquema abaixo.

```
Acorda Determinante

    verifica a necessidade de ligação (ou seja, se ainda não possui superior)
        procura vizinho nominal à direita[59]
        se encontrou o vizinho
            tenta estabelecer ligação( ok )
            se ok
                assinala "fez_algo"
            senão
```

[59] Nessa busca, ele verifica a não existência de outros agentes, como um verbal ou um coordenação.

```
                erro
            se não encontrou o vizinho
                erro

        se não mais há a necessidade de ligação (ou seja, agora já possui superior)
            trata msgs de "término"

        se existe algum "fez_algo"
            apaga todas as ocorrências desse flag
        se não existe nenhum "fez_algo"
            incrementa a lista "não_fez_nada" com o identificador deste agente

fim.
```

5.3.4 Acorda para Agente Pronome Relativo

O procedimento "acorda" para o agente pronome relativo pode ser dividido em duas etapas. Na primeira, o agente pronome relativo encontra o agente verbal a sua direita que será seu superior (não está sendo tratado o caso para superior do tipo preposicional, como seria o caso da sentença "o menino de quem se falou... "), verifica a existência de agentes nominais entre esse agente verbal e ele próprio (o que servirá para determinar a subcategorização que ele estará preenchendo) e assinala que irá tentar encontrar um agente nominal com essas características para a subcategorização necessitada. Na segunda, o agente pronome relativo procura esse agente nominal e estabelece ligações com ele. Após isso, efetiva a ligação com o agente verbal passando as características específicas desse agente nominal encontrado.

```
Acorda_Pronome_Relativo

    verifica a não existência de um superior (que seria um agente verbal)
    /* não está sendo tratado o caso de pronome relativo ligado a ag. preposicional */
    se não existe
        procura pelo agente verbal à direita verificando a existência de algum agente
            nominal entre eles (isso indica a subcategorização que o pronome relativo
            irá preencher)
        assinala tentativa de estabelecimento de ligações, pegando as características
            desta subcategorização
        assinala "fez_algo"

    verifica a não existência de um subordinado (que seria um agente nominal)
    se não existe
        procura pelo agente nominal à esquerda utilizando as características exigidas
            pelo agente verbal
        estabelece ligações com o agente nominal encontrado
        efetiva a ligação com o agente verbal passando as características do ag. nominal e
            verificando se esse agente já não está ligado ao agente verbal
            se estiver
                quebra essa ligação que existia anteriormente
        assinala "fez_algo"
```

```
        se já possuir superior e subordinado
            trata msgs de "término"

        se existe algum "fez_algo"
            apaga todas as ocorrências desse flag
        se não existe nenhum "fez_algo"
            incrementa a lista "não_fez_nada" com o identificador deste agente

    fim.
```

5.3.5 Acorda para Agente Coordenação

O procedimento "acorda" para o agente coordenação também pode ser dividido em duas etapas. A primeira consiste na detecção dos agentes que estão sendo coordenados e a segunda, na realização das operações que condizem com o comportamento que o agente deve assumir de acordo com os agentes que coordena (comportamento de agente nominal ou de agente verbal). Aqui só irá ser apresentada a primeira parte (detecção) que é a mais importante. Quando o agente coordenação se comporta como um nominal, ele segue o esquema reproduzido acima para o agente nominal (seção 5.3.1). Quando se comporta como um verbal, ele apenas faz o tratamento das mensagens de término.

```
Acorda Coordenação (Situação de Detecção)

    verifica a necessidade de encontrar pelo menos um subordinado
    caso exista
        procura por pronomes relativos vizinhos à esquerda e à direita
        se existe pelo menos um agente pronome relativo
            /* coordenação dos nominais mais próximos */
            procura um vizinho nominal à direita
            se esse vizinho possui superior
                pede quebra de ligações entre esse vizinho e o seu superior
                procura um vizinho nominal à esquerda
                se esse vizinho possui superior
                    pede quebra de ligações entre esse vizinho e seu superior
                    agente coordenação tenta estabelecer ligação com esses vizinhos nominais
                    assinala "fez_algo"
            se esse vizinho não possui superior
                verifica a existência de um agente pronome relativo à esquerda
                se existe
                    procura pelo agente nominal à esquerda desse agente pronome relativo
                    tenta estabelecer ligação com esse agente nominal
                    assinala "fez_algo"
                se não existe
                    procura pelo primeiro agente nominal à esquerda do agente coordenação
                    tenta estabelecer ligação com esse agente nominal
                    assinala "fez_algo"

        se não existe agente pronome relativo
```

```
            procura agentes verbais à esquerda e à direita
            se existe apenas um agente verbal
                se esse agente verbal está à direita
                    /* coordenação de nominais */
                    procura os agentes nominais dos dois lados
                    tenta estabeler ligações ( ok )
                    se ok
                        assinala "fez_algo"
                    senão
                        erro
                se esse agente verbal está à esquerda
                    verifica a possibilidade de gapping
                    se não há
                        /* coordenação de nominais */
                        procura os agentes nominais dos dois lados
                        tenta estabeler ligações ( ok )
                        se ok
                            assinala "fez_algo"
                        senão
                            erro
                    se há possibilidade de gapping
                        vai dormir (esperando que ela seja detectada por algum agente
                            nominal e que este peça a criação do agente verbal)
            se existem dois agentes verbais
                /* coordenação de verbais */
                tenta estabelecer ligações com esse agentes verbais ( ok )
                se ok
                    assinala "fez_algo"

    se existe algum "fez_algo"
        apaga todas as ocorrências desse flag
    se não existe nenhum "fez_algo"
        incrementa a lista "não_fez_nada" com o identificador deste agente


fim.
```

5.4 Considerações Finais sobre o Capítulo

Neste capítulo foram apresentadas as principais características encontradas na programação do sistema implementado. O pseudo-código apresentado serviu para dar uma idéia do modo como se passam os acontecimentos quando um agente é acordado. Fora pequenas simplificações para a melhora da apresentação, o código, que foi feito em Prolog, segue o que foi apresentado no decorrer do capítulo e pode ser visto, por completo, no Apêndice.

6. *Resultados Obtidos e Problemas Não Resolvidos*

Encostei-me a ti, sabendo bem que eras somente onda.
Sabendo bem que eras nuvem, depus a minha vida em ti.
E como sabia bem tudo isso, e dei-me a teu destino frágil,
fiquei sem poder chorar, quando cai.[60]

Neste capítulo serão apresentados os resultados obtidos com a implementação do projeto, assim como os problemas que ainda não foram solucionados. As principais considerações sobre alguns dos avanços obtidos dizem respeito ao tipo de agente que se utilizou no sistema (cf. diferenciação feita na seção 1.2 — reativo e cognitivo), visto que este projeto se iniciou sem a menor pressuposição do que seria um agente para o processamento lingüístico distribuído. Pode-se dizer que se obteve um *agente misto*, com uma maior inclinação para o lado reativo. A principal razão para isso vem da observação do modelo de ação utilizado pelos agentes do sistema (cf. seção 1.3.1).

Apesar de os agentes trabalharem com informações no nível simbólico, e por isso terem características de agente cognitivo, o modelo de ação é baseado em um estilo *estímulo-reação* cujas características se assemelham às do sistema PACO[61] [Demazeau93a] (cf. seção 1.3.1 desta dissertação), seja por utilizar a idéia de "forças de atração" que têm uma correlação com a idéia de "campos de comunicação" do PACO (que guiam as interações entre os agentes), seja por utilizar a idéia de "vizinhança", que pode ser comparada às idéias de "campos de ação" e "campos de percepção".

---

60 Cecília Meireles, poema "Epigrama nº 8".

No que se refere à *comunicação* (cf. seção 1.3.2), os agentes apresentam um modelo estruturado para a obtenção de informações advindas de outros agentes (com o uso inclusive de um protocolo semelhante ao da *rede de contrato* — cf. seções 1.3.2 e 4.4.1.2). Neste aspecto, os agentes do sistema podem ser considerados como cognitivos.

A idéia de resolução de conflitos — *quebra de ligações* entre os agentes — utilizada no sistema deve ser caracterizada pensando que os agentes do sistema são benevolentes. Neste projeto, não se considerou isso como uma questão tão relevante.

Os agentes do sistema não possuem uma representação dos outros agentes do sistema, o que é uma característica forte de agentes reativos, mas isso poderá ter de ser revisto para evitar perda no desempenho do sistema. Por exemplo, se em uma determinada ativação (quando um agente é acordado) o agente (ag_1) precisou contactar um outro agente (ag_2) de determinado tipo que está em sua vizinhança, ele não irá manter o identificador deste agente, nem a posição que ele (ag_2) ocupa na sentença, o que significa que, na próxima ativação, se ele (ag_1) precisar interagir com este agente (ag_2) novamente, terá que fazer a busca por ele mais uma vez.

O principal motivo para, nesta primeira versão do sistema, manter essa característica foi a possibilidade de a configuração da sentença mudar devido à introdução de novos agentes no ambiente (com a criação de agentes). Por exemplo, se em um momento anterior um agente A era o primeiro agente do tipo nominal que estava à direita de um agente B, é possível que, após alguns instantes, o primeiro agente desse tipo passe a ser um outro (agente C). Seria necessário, com isso, ter um mecanismo de revisão do conhecimento do agente (conhecimento sobre a configuração da sentença), o que iria complicar a implementação sem trazer ganhos no desempenho do sistema, pois ainda não se tem uma versão implementada para rodar efetivamente em paralelo.

---

[61] Essa semelhança foi apontada por Jaime Sichman para o autor, que agradece a colaboração.

Devido à não implementação em paralelo, também não se tem uma noção exata de qual o ganho que o sistema poderia ter em relação a uma implementação "convencional" (ou seja, linear com divisão em módulos como morfológico, sintático, semântico etc.). Uma conjectura é que seriam formados grupos de agentes próximos — que corresponderiam à formação dos sintagmas da sentença[62] — onde a troca intensa de mensagens ficaria restrita e onde boa parte das ambigüidades seriam resolvidas. A melhora no desempenho estaria associada ao número de grupos formados: quanto mais grupos, melhor seria o desempenho. Essa idéia, deve-se reiterar, não foi testada na prática e constituirá um dos itens a serem considerados numa próxima etapa.

Com relação à comparação com os sistemas apresentados no Capítulo 2, uma das principais diferenciações está no modelo de busca utilizado pelo sistema. Enquanto a busca no WEP [Small80] é feita utilizando "expectativas" (cf. apresentado à página 2-15 desta dissertação), que é um processo em que os agentes têm de esperar pela resposta a uma "expectativa" emitida, a busca utilizada pelo sistema é feita tanto de um modo ativo (ou seja, os agentes tem uma noção do que devem procurar e vão atrás disso) quanto de um modo passivo (que é o que ocorre com os agentes nominais). Uma observação interessante sobre essa questão é o comportamento dos agentes nominais quando ocorre "*gapping*" (cf. seção 4.5.6.2, à página 4-64). Mesmo o agente possuindo um comportamento inicialmente passivo com relação à busca, é possível mudá-lo para ativo de acordo com seu estado (em termos das ligações que ele possui, por exemplo). Isso permite a detecção de certos fenômenos ("*gapping*" e outros tipos de elipses) que, de outra maneira, ficariam muito difíceis de serem detectados porque não haveria uma "expectativa" devido à inexistência do agente que deveria emiti-la.

Dessa maneira, pode-se dizer que, além de estender o modelo de "*Polaroid Words*" [Hirst87] no sentido de lidar com uma análise sintática que possui certa semelhança com a utilizada pelo WEP [Small80], tratou-se de outros fenômenos lingüísticos que testaram novas classes de problemas que os sistemas apresentados (WEP [Small80 e Adriaens86] e PEP [Devos88, Devos94])

---

62 Em uma análise tradicional.

não conseguiriam resolver (como "*gapping*"). As características para a resolução de ambigüidade semântica vistas em [Hirst87] foram mantidas, embora não tenha sido possível implementar mecanismos externos ao agente, como um "*marker passer*", que poderiam auxiliar no processo de resolução de ambigüidade.

Um dos principais problemas a serem resolvidos é o da ambigüidade categorial, que acontece quando uma palavra pertence a mais de uma categoria morfossintática (como verbo e substantivo[63]). Há duas possibilidades para a resolução desse problema. A primeira é utilizar um pré-processamento que defina a categoria em que a palavra pode ser enquadrada na sentença (por exemplo, utilizando métodos estatísticos, como "*part-of-speech tagging*" [Allen95, cap. 7]) que será então processada pelos agentes. A segunda é utilizar mais de um agente por palavra quando esta pertencer a mais de uma categoria. Tal mudança poderá afetar profundamente o funcionamento do sistema em vista da idéia de ligação utilizada: no fim do processamento de uma sentença que possua palavras semanticamente ambíguas sempre haverá agentes que não estarão com suas ligações necessárias preenchidas. Novos critérios de parada do sistema, que atualmente é baseado no preenchimento das ligações necessárias, terão que ser testados e novas interações entre esses agentes "duplicados" terão de ser utilizadas, provavelmente exigindo algum tipo de protocolo de resolução de conflitos. Essa é uma hipótese que terá que ser melhor analisada.

Algumas questões sobre as estratégias utilizadas pelos agentes na resolução de alguns dos problemas lingüísticos serão abordadas a seguir no sentido de tentar apresentar alternativas às análises utilizadas para a implementação do sistema. Serão abordadas as estratégias utilizadas na resolução do "*pp-attachment*" e das construções relativas.

No "*pp-attachment*", o principal problema encontrado foi a necessidade de conseguir expressar mais de uma interpretação para a sentença. A estratégia para forçar uma segunda interpre-

---

[63] Por exemplo, a palavra "almoço" tanto pode ser um verbo, como na sentença "eu almoço todos os dias", quanto pode ser um substantivo, como na sentença "o almoço estava bom hoje".

tação foi utilizar uma mensagem (outra interpretação) quando o sistema parasse após a primeira interpretação e utilizar quebra de ligações. Um dos problemas dessa estratégia, do modo como foi implementada, foi ter levado em conta a possibilidade de apenas um agente ter duas interpretações. Além disso, para mais de um agente com outra interpretação, essa estratégia de quebrar as ligações e se ligar a outro traria, quando ocorrendo com vários agentes, uma alternância que, devido às permutações necessárias, iria comprometer o desempenho do sistema. Uma outra forma terá que ser ainda trabalhada para resolver esse problema.

Nas construções relativas, a estratégia terá que ser melhorada para poder lidar com sentenças que envolvam a inter-relação com outros tipos de agentes, como com o agente coordenação, que pode ser visto no seguinte exemplo:

(37) O menino que a menina que caiu e chorou ama comprou uma bicicleta.

Essas inter-relações envolverão um estudo mais aprofundado e ficarão para um trabalho futuro.

Um problema que esse tipo de construção também pode trazer é a existência de sujeito oculto na sentença (o que não foi tratado pelo sistema), como no exemplo:

(38) O livro que [eu] comprei ficou na mesa da escola.[64]

Neste caso, a estratégia de o agente pronome relativo observar a existência de um agente nominal entre ele e o agente verbal não irá funcionar e será preciso, primeiramente, buscar informações no verbo com relação a ele já ter encontrado seu sujeito (ou objeto). É necessário deixar claro que o agente pronome relativo, no exemplo dado, não conseguiria, mesmo assim, realizar sua ligação com o agente nominal a sua esquerda (livro) como sujeito porque ele iria checar a concordância do agente nominal (sua palavra) com o verbal — "o livro comprei" resultaria em um erro, pois não há concordância de pessoa e número entre "livro" e "comprei". Assim, o sistema iria terminar com um erro.

---

[64] Onde o que está entre colchetes ("[eu]") não aparece na sentença.

Foi dito na seção 4.3.1 que há uma semelhança entre as idéias de Forças de Atração (FA) e de "*Dependency Grammar*" (DG). Entretanto, as idéias diferem porque as FA utilizam um direcionamento — quando da formação da estrutura hierárquica da sentença — que é diferente do utilizado pela DG[65]. Isso pode ser visto pela relação entre substantivos, verbos e determinantes nas duas abordagens e a noção de *saturação*. Os substantivos, que podem ser considerados como termos saturados (isto é, que *não dependem* de complementação), atraem os determinantes no modelo das FA, enquanto os determinantes é que atraem os substantivos no modelo da DG, visto que eles são insaturados (ou seja, *dependem* de uma complementação). Já os verbos, que também são insaturados, tanto no modelo das FA quanto no da DG, atraem os substantivos. Ou seja, a DG segue a noção de saturação à risca enquanto as FA não. Uma melhor análise terá que ser realizada quando a fase de interpretação semântica for implementada, o que ficará para uma próxima etapa do projeto.

---

[65] Essa diferenciação foi apontada por José Borges Neto e por Luiz Arthur Pagani para o autor, que agradece a colaboração.

Conclusão

Fico com a pureza da resposta das crianças:
— É a vida, é bonita, e é bonita![66]

Nesta dissertação foi apresentado um sistema para Processamento de Linguagem Natural com as características do paradigma Sistema Multi-Agentes com o objetivo de explorar a fronteira entre essas duas sub-áreas da Inteligência Artificial.

Foi apresentado um modelo de processamento lingüístico capaz de ser implementado de forma distribuída por *agentes lingüísticos* que são associados, um a um, às palavras da sentença e que têm o comportamento mapeado a partir das categorias morfossintáticas da palavra associada ao agente.

O comportamento dos agentes foi programado de forma a fazer com que os agentes fiquem tentando encontrar outros agentes com os quais devem estabelecer as ligações necessárias e, encontrado esses agentes, efetivem essas ligações. Foi estabelecido um critério de parada para o sistema relacionado aos agentes terem suas ligações necessárias estabelecidas, implicando que todas as partes da sentença estariam conectadas.

---

[66] Gonzaguinha na música "O que é o que é?".

Foram analisados alguns fenômenos lingüísticos que não haviam sido explorados suficientemente por outros sistemas que seguem a linha de *sistemas baseados na palavra* e exemplos foram apresentados no intuito de mostrar as possibilidades da implementação realizada.

Apêndice

O objetivo desta seção é expor o código Prolog completo que foi utilizado na implementação tanto dos agentes, que teve sua descrição feita no Capítulo 5, como das partes extra-agentes, no intuito de que isso seja útil para que outras pessoas possam testar, experimentar, encontrar erros, assim como validar os possíveis acertos que foram feitos no decorrer desta pesquisa.

```
% ------------------------------------------------------------------------
% Agentes 2.0 -- Nessa versao, cada palavra e' associada a um agente
%               que ficara' responsavel por encontrar as ligacoes
%               necessarias para que a palavra realize sua funcao.
%
%               "Categorial ambiguity" esta' sendo desprezada , o
%               que dispensou um parser. O problema de uma palavra
%               poder pertencer a mais de uma categoria morfossinta-
%               tica ficou em aberto.
%
%               O conceito de identificacao esta' sendo separado
%               do de posicao, para que seja possivel criar agentes
%               (no caso de elipse) sem que haja confusao entre a
%               posicao do agente e seu identificador. Se nao houvesse
%               a possibilidade de criacao de agentes, nao seria neces-
%               sario separar esses dois conceitos em dois campos, pois
%               eles seriam sempre iguais.
%
%               O processo de mapeamento da posicao de um agente
%               para sua identificacao fica centralizado em um tipo
%               de agente(*), diferente dos agentes que fazem o pro-
%               cessamento das palavras, que tem o trabalho de:
%
%                  - retornar a posicao de um dado agente (a propria
%                    posicao nao fica armazenada no agente);
%
%                  - retornar a identificacao do vizinho (direito ou
%                    esquerdo); e
%
%                  - administrar a criacao de um agente em uma deter-
%                    minada posicao (i.e., o agente de localizacao
%                    tem por funcao "rearranjar" a posicao dos agentes
%                    posteriores ao que esta' sendo inserido).
%
%               (*) nesta implementacao, ele e' apenas um "predicado"
%
%
%               A parte de resolucao de ambiguidade nao foi totalmen-
%               te implementada, nao por ser algo tao complicado (ver
%               Hirst), mas sim por, na tese, a parte semantica nao
%               ter tido o desmembramento necessario para que a im-
%               plementacao fosse tao importante. A parte de escolha
%               semantica de um significado foi utilizada, por exemplo,
%               quando um candidato a sujeito ou objeto possuia dois
%               ou mais significados e, pela necessidade imposta pelas
%               informacoes verbais, conseguia fazer uma ligacao devido
%               a apenas um desses significados.
%
%
%               Ainda ha' um comentario importante, que e' a respeito
%               do processo de escolha aleatoria dos agentes. A ideia
%               inicial e' que seria utilizada uma funcao que geraria
%               um numero inteiro aleatorio no intervalo de [1, numero
%               total de agentes] e que tal agente seria acordado. Como
%               a implementacao nao foi feita com um prolog "oficial",
%               nao houve manual que explicasse como a funcao "random"
%               -- que deveria ser tal funcao -- funcionava. A opcao
%               encontrada foi implementar uma pseudo-funcao de aleatorie-
%               dade utilizando listas com numeros "aleatoriamente" inse-
%               ridos. A cada chamada da funcao, e' retornado um numero
%               da lista e um marcador fica apontando para o proximo.
%               Quanto a lista termina, esse apontador e' reinicializado
%               com o primeiro da lista.
%
%               Ha' um fato que deve ser tornado explicito, e que em uma
%               implementacao distribuida deve ser modificado, relativo ao
%               controle de quais agentes ja' responderam as mensagens do
%               tipo "broadcast" enviadas por um determinado agente verbal.
%               Nesta implementacao, nao houve tal controle -- o que pode-
%               ria implicar em algum erro de ligacao, caso um agente ver-
%               bal fosse acordado antes que todos os outros o tivessem
%               respondido. Como tal possibilidade de erro e' muito mais
%               efetiva na primeira ativacao dos agentes (quando as mensa-
%               gens "broadcasts" ja' foram enviadas -- ver inicializacao
%               dos agentes verbais, especialmente seu "mapeamento de
%               necessidades"), optou-se por "mascarar" o problema com a
%               modificacao das listas aleatorias de acordo com a frase
%               que estaria sendo processada, no sentido de fazer com que
%               os identificadores dos agentes verbais fossem colocados
%               -- apenas na primeira vez em que aparecem, i.e., na pri-
%               meira ativacao -- apos todos os outros identificadores de
%               agentes nas listas aleatorias, implicando em eles (ag. ver-
%               bais) serem chamados apos todos os outros terem sidos cha-
%               mados pelo menos uma vez. Deve-se repetir que isso so'
%               acontece na primeira ativacao dos agentes verbais.
%
%               Seria isso um grave problema gravissimo?
%               Pensou-se que nao, pois se evitou mexer com uma
%               parte da implementacao que nao deveria estar relacionada
%               ao agente em si, mas sim ao modelo de comunicacao que ele
%               estaria utilizando em um determinado momento -- a implemen-
%               tacao desse tipo de controle nao deveria estar localizada
%               no codigo do funcionamento do agente. Alem do que, por se
%               algo muito importante na primeira ativacao e nao tanto
%               nas outras, nao foi necessario ficar "manipulando" todos
%               os numeros das listas aleatorias -- ou seja, nao se esta'
%               falseando, com isso, todo o funcionamento do sistema.
%               A perda que se teve com isso foi em termos de clareza de
%               como as coisas estao acontecendo. Ou seja, alguem poderia
%               se perguntar como e' que o programa estaria funcionando
```

```
%               corretamente sem que houvesse o controle de quem ja' res-
%               pondeu `as mensagens. Esse comentario tem exatamente tal
%               funcao.
%
% ------------------------------------------------------------------------


% ------------------------------------------------------------------------
%    FORMATO DOS DADOS DE UM AGENTE
% ------------------------------------------------------------------------

% ==> agente( IdAgente , Tipo , Palavra , InfoDic , Significado )

% ------------------------------------------------------------------------
%    FORMATO DE UM VOTO
% ------------------------------------------------------------------------

% ==> voto( IdAgenteDestino , TipoSintatico , TipoSemCandidato , IdCandidato )

% ------------------------------------------------------------------------
%    FORMATO DA POSICAO DE UM DADO AGENTE
% ------------------------------------------------------------------------

% ==> posicao( Posicao , IdAgente )


% ------------------------------------------------------------------------
%    FORMATO DAS LIGACOES
% ------------------------------------------------------------------------

% ==> superior( IdAgente , FuncSint , IdSup )   // apenas um superior
%                                               // por funcao sintatica

% ==> subordinado( IdAgente , FuncSint , IdSub ) // apenas um subordinado
%                                               // por funcao sintatica

% ------------------------------------------------------------------------
%    FORMATO DAS MENSAGENS DE CONTROLE
% ------------------------------------------------------------------------

% ==> outraInterpretacao( IdAgente ou "fail" )
% ==> todosViramTermino( "true" ou "fail" )
% ==> terminei( IdAgtTermino , ListaDosQueNaoViram )
%
%     obs. 1: quem pode colocar um "terminei": --> agente verbal
%                                              --> agente coordenacao
%
%     obs. 2: todo agente tem que tratar a mensagem "terminei" colocada
%             por qualquer outro agente (seja esse quem for). O que e' tratar
%             uma mensagem? E' verificar se ele (o agente que esta' executan-
%             do) ja' terminou o que tem de fazer e, se isso tiver
%             acontecido, retirar o seu "nome" da ListaDosQueNaoViram,
%             indicando que ele (o agente executante) viu o termino e, mais,
%             que ele tambem terminou. Se ele nao tiver terminado, ele nao
%             retirara' seu "nome" da lista e o sistema nao terminara'. O
%             ultimo que ainda nao viu o termino, quando vir, assertara' o
%             fato "todosViramTermino( true )". O predicado que faz esse
%             trabalho tem o nome "tratamentoDeMsgTermino"


% -----
  inicio :- dicionario , geral , ramdom , frases , vizinhanca .


% ------------------------------------------------------------------------
%    PREDICADOS COM AS FRASES DE TESTE
% ------------------------------------------------------------------------

  frases :- ['frases.ari'] .


% ------------------------------------------------------------------------
%    PREDICADOS DO DICIONARIO
% ------------------------------------------------------------------------

  dicionario :- ['dicionar.ari'] .

% ------------------------------------------------------------------------
%    PREDICADOS DA FUNCAO DE ALEATORIEDADE
% ------------------------------------------------------------------------

  ramdom :- ['ramdom.ari'] .

% ------------------------------------------------------------------------
%    PREDICADOS DE VIZINHANCA
% ------------------------------------------------------------------------

  vizinhanca :- ['vizinhos.ari'] .

% ------------------------------------------------------------------------
%    PREDICADOS GERAIS
% ------------------------------------------------------------------------

  geral :- ['geral.ari'] .


% ------------------------------------------------------------------------
%    PRINCIPAL: inicializacao, loop de acordas e finalizacao
% ------------------------------------------------------------------------
```

```
test( String ) :-

    create( ArqResult , 'result.ari' ) ,

    write( String ) ,
    nl ,

    inicializaListaNaoFezNada ,

%   inicializacao dos agentes
    inicializacaoDosAgentes( String ) ,

%   inicializacao das listas de agentes / esta' no arquivo "random.ari"
    inicializaListasAgentes ,

%   inicializaza as listas aleatorias (usadas pelo predicado "random")
    inicializaListasAleatorias ,

%   Loop da escolha do agente que ira' trabalhar e condicao de parada.

%    enquanto houver mais interpretacoes possiveis diferentes
%       enquanto nao houver uma msg de termino que todos tenham visto faca
%          escolha_um_agente_aleatoriamente( Vn , Agente ) .
%          acorde( Agente )
%       fim_enquanto
%       dar resultado
%       quebra lacos para forcar nova interpretacao
%    fim_enquanto

    assert( todosViramTermino( fail ) ) ,

    repeat ,

%             ha' outra interpretacao
              (   (   outraInterpretacao( AgtOutraInt ) ,
                      write( 'outra interpretacao (' ) ,
                      write( AgtOutraInt ) ,
                      write( ')' ) , nl ,
%                     obtem o superior ... (*)
                      obtemSuperior( AgtOutraInt , SupAgtOutraInt )
                      ) ,

%           (*) ... para quebrar ligacao
                    retractall( superior( AgtOutraInt , _ , SupAgtOutraInt ) ) ,
                    assert( superior( AgtOutraInt , _ , 0 ) ) ,
                    retract( subordinado( SupAgtOutraInt , PosSint , AgtOutraInt ) )

                    assert( subordinado( SupAgtOutraInt , PosSint , 0 ) ) ,

%           atualiza a lista de interpretacoes na base de dados
                    retract( outraInterpretacao( AgtOutraInt ) ) ,
                    assert( segundaInterpretacao( AgtOutraInt ) ) ,
                    retractall( termineI( _ , _ ) ) ,
                    retractall( listaNaoFezNada( _ ) ) ,
                    assert( listaNaoFezNada( [] ) ) ,
                    retractall( todosViramTermino( true ) ) ,
                    assert( todosViramTermino( fail ) )

              ;   % else
                  % faz nada e continua
                  true
              ) ,

              loopDeAcordas ,

              apresentaResultado( ArqResult , String ) ,

              (   /* se existe uma outra interpretacao, continua */
                  outraInterpretacao( _Agt ) ,
                  fail
              ;   /* senao, termina */
                  not( outraInterpretacao( _ ) ) ,
                  true
              ) ,

    ! ,

    limpaBaseDeFatos ,

%   fecha o arquivo aberto
    close( ArqResult )

% fim teste
  .

  loopDeAcordas :-

    repeat ,
%       obtem o numero de agentes presentes no momento (o que pode mudar)
            numeroAgts( NumAgts ) ,
%       escolhe um agente aleatoriamente
            random( X , NumAgts ) ,
            tipoAgente( X , TipoDoAgente ) ,
            acorda( X , TipoDoAgente ) ,
            (   (   todosViramTermino( TodosViram ) ,
                    TodosViram
                )
            ;   /* mecanismo que verifica se o sistema  */
                /* esta' em loop sem acao, ou nao esta' */
                (   listaNaoFezNada( ListaNFN ) ,
```

```
                length( ListaNFN , TamListaNFN ) ,
                igual( TamListaNFN , NumAgts )
               ) ,
               write( 'ERRO: FRASE AGRAMATICAL' ) , nl
             ) ,
    !

% fim loopDeAcordas
  .

% ----------
  numeroAgts( NumAgts ) :- numAgts( NumAgts ) , ! .


% ------------------------------------------------------------------------
% inicializaListaNaoFezNada :- essa lista sera' utilizada para verifi-
%       car quem nao fez nada em sua execucao. Se o agente acordado
%       nao fez nada, ele coloca seu nome da lista. Se ele fez, entao
%       ele esvazia a lista. Quando a lista estiver com todos os agen-
%       tes, e' porque ninguem fez nada e isso indicara' que a frase
%       e' agramatical. Inicialmente a lista comeca vazia.


  inicializaListaNaoFezNada :- assert( listaNaoFezNada( [] ) ) , ! .



% ------------------------------------------------------------------------
% InicializacaoDosAgentes :- faz a inicializacao e chamada do loop
%                            que ira' inicializar os dados de cada
%                            agente.


  inicializacaoDosAgentes( String ) :-

     V0 is 0 ,
     assert( numAgts( 0 ) ) ,
     assert( posicao( 0 , 0 ) ) ,
     inicializaAgentes( V0 , Vn , String )

% fim inicializacaoDosAgentes
  .


% ------------------------------------------------------------------------
% InicializaAgentes :- para cada palavra, pega as informacoes no dicio-
%                      nario e, de acordo com o tipo, inicializa o agente


% ----------------
  inicializaAgentes( V0 , V0 , [] ) :-

%    V0 e' o numero total de agentes existentes na inicializacao
     assert( numAgts( V0 ) )

% fim inicializaAgentes /caso final
  .


% ----------------
  inicializaAgentes( V0 , Vn , [Palavra|Restante] ) :-

     V1 is V0 + 1 ,

%    pega as informacoes no dicionario
     dicionario( [palavra=[Palavra] , cat=[Categoria]|Resto_Info] ) ,

%    de acordo com a CATegoria da palavra, inicializa um tipo de agente
     (   Categoria = det ,
%                    ->> neste momento, para todo agente, posicao e'
%                        igual `a identificador ( V1 , V1 ... )
                inicializa_det( V1 , V1 , Palavra , Resto_Info )

     ;   Categoria = nome ,
                inicializa_nom( V1 , V1 , Palavra , Resto_Info )

     ;   Categoria = verbo ,
                inicializa_verb( V1 , V1 , Palavra , Resto_Info )

     ;   Categoria = prep ,
                inicializa_prep( V1 , V1 , Palavra , Resto_Info )

     ;   Categoria = conjuncao ,
%                   "coord" de "coordenacao"
                inicializa_coord( V1 , V1 , Palavra , Resto_Info )

     ;   Categoria = pron_rel ,
                inicializa_pron_rel( V1 , V1 , Palavra , Resto_Info )
     ) ,

%    chamada recursiva
     inicializaAgentes( V1 , Vn , Restante )

% fim inicializaAgentes /caso recursao
  .


% ------------------------------------------------------------------------
% ApresentaResultado( ArqResult , String ) :- aqui agrupa e apresenta
%                       o resultado final  do trabalho dos agentes en-
```

```
%                       viando para o arquivo de saida associado ao
%                       "handle"  ArqResult.


  apresentaResultado( ArqResult , String ) :-

     write( ArqResult , String ) ,
     nl( ArqResult ) ,

     foreach( agente( IdAg , TipoAgt , Palavra , _ , _ ) ,

                    (    nl( ArqResult ) ,
                         write( ArqResult , '-------------------' ) ,
                         nl( ArqResult ) ,
                         write( ArqResult , '(' ) , write( ArqResult , IdAg ) ,
                         write( ArqResult , ') "' ) , write( ArqResult , Palavra ) ,
                         write( ArqResult , '" ' ) ,
                         nl( ArqResult ) ,
                         write( ArqResult , '    (' ) ,
                         write( ArqResult , TipoAgt ) , write( ArqResult , ') ') ,
                         foreach( subordinado( IdAg , Tipo , Sub ) ,
                               ( write( ArqResult , '   /   (') ,
                                 write( ArqResult , Sub ) ,
                                 write( ArqResult , ' ' ) ,
                                 write( ArqResult , Tipo ) ,
                                 write( ArqResult , ' ' ) ,
                                 write( ArqResult , IdAg ) ,
                                 write( ArqResult , ')' )
                               )
                              ) ,
                         superior( IdAg , Tipo , Sup ) ,
                         (   Sup \= 0 ,
                             write( ArqResult , '   /   (') ,
                             write( ArqResult , IdAg ) , write( ArqResult , ' ' ) ,
                             write( ArqResult , Tipo ) , write( ArqResult , ' ' ) ,
                             write( ArqResult , Sup ) , write( ArqResult , ')' )
                         ) ,
                         nl( ArqResult )
                    )
                  ) .

     !

% fim apresentaResultado
  .

% ------------------------------------------------------------------
% LimpaBaseDeFatos :- retira todos os fatos que foram colocados na memoria

  limpaBaseDeFatos :-

%    informacoes internas dos agentes (seus dados)
     retractall( agente( _ , _ , _ , _ , _ ) ) ,
     retractall( subordinado( _ , _ , _ ) ) ,
     retractall( superior( _ , _ , _ ) ) ,
     retractall( superior_que( _ , _ ) ) ,

%    informacoes externas dos agentes
     retractall( posicao( _ , _ ) ) ,
     retractall( numAgts( _ ) ) ,

%    informacoes complementares para o sistema.
     retractall( listaNaoFezNada( _ ) ) ,
     retractall( agtFezAlgo( _ ) ) ,
     retractall( listaAgentes( _ , _ ) ) ,
     retractall( listaInicial( _ , _ ) ) ,
     retractall( listaAleatoria( _ , _ ) ) ,

%    mensagens indiretas utilizadas pelos agentes
     retractall( voto( _ , _ , _ , _ , _ ) ) ,
     retractall( mem( _ ) ) ,

%    pendencia e' usada apenas pelo agente verbal
     retractall( pendencia( _ , _ ) ) ,

%    "flags" do sistema de termino
     retractall( todosViramTermino( _ ) ) ,
     retractall( terminei( _ , _ ) ) ,

%    "flag" para forcar a quebra de lacos para outra interpretacao
     retractall( outraInterpretacao( _ ) ) ,
     retractall( segundaInterpretacao( _ ) ) ,

%    "flags" de diversos tipos
     retractall( flag_que ) ,
     retractall( subordinado_que( _ ) ) ,
     retractall( relativo( _ ) ) ,
     retractall( coordenacao( _ ) ) ,
     retractall( espera( _ ) ) ,
     retractall( espera2( _ ) ) ,
     retractall( gapping( _ ) ) ,
     retractall( flexibilizouSuj( _ ) ) ,
     retractall( flexibilizouObj( _ ) ) ,
     retractall( flexibilizouObjInd( _ ) ) ,
     retractall( flexibilizou_2_Suj( _ ) ) ,
     retractall( flexibilizou_2_Obj( _ ) ) ,
     retractall( flexibilizou_2_ObjInd( _ ) )

% fim limpaBaseDeFatos
  .
```

```
% ------------------------------------------------------------------
%   PREDICADOS DE INICIALIZACAO ESPECIFICOS DE CADA AGENTE
% ------------------------------------------------------------------

% ____
% NOME

  inicializa_nom( IdAg , Pos , Palavra , InfoDic ) :-

%   insere o novo agente na base de dados prolog
    assert( agente( IdAg , nominal , Palavra , InfoDic , [] ) ) ,

%   acrescenta o agente na "tabela" de posicoes
    acrescimoAgente( IdAg , Pos ) ,

%   inicializa as listas de ligacoes do agente
    assert( superior( IdAg , _ , 0 ) ) ,
    assert( subordinado( IdAg , _ , 0 ) )

% fim inicializa_nom
  .


% _____
% VERBO

  inicializa_verb( IdAg , Pos , Palavra , InfoDic ) :-

%   insere o novo agente na base de dados prolog
    assert( agente( IdAg , verbal , Palavra , InfoDic , [] ) ) ,

%   acrescenta o agente na "tabela" de posicoes
    acrescimoAgente( IdAg , Pos ) ,

%   inicializa o mapa de necessidades do agente verbal
%   e coloca os "preciso"s na memoria centralizada
    retornaValorAtributo( significado , InfoDic , Significados ) ,
    mapeamentoNecessidades( IdAg , Significados , InfoDic ) ,

%   inicializa as listas de ligacoes do agente
    assert( superior( IdAg , _ , 0 ) ) ,
    assert( subordinado( IdAg , suj , 0 ) ) ,
    assert( subordinado( IdAg , obj , 0 ) ) ,
    assert( subordinado( IdAg , obj_ind , 0 ) )

% fim inicializa_verb
  .

% _________
% PREPOSICAO

  inicializa_prep( IdAg , Pos , Palavra , InfoDic ) :-

%   insere o novo agente na base de dados prolog
    assert( agente( IdAg , prep , Palavra , InfoDic , [] ) ) ,

%   acrescenta o agente na "tabela" de posicoes
    acrescimoAgente( IdAg , Pos ) ,

%   inicializa as listas de ligacoes do agente
    assert( superior( IdAg , _ , 0 ) ) ,
    assert( subordinado( IdAg , _ , 0 ) )

% fim inicializa_prep
  .


% ____________
% DETERMINANTE

  inicializa_det( IdAg , Pos , Palavra , InfoDic ) :-

%   insere o novo agente na base de dados prolog
    assert( agente( IdAg , determinante , Palavra , InfoDic , [] ) ) ,

%   acrescenta o agente na "tabela" de posicoes
    acrescimoAgente( IdAg , Pos ) ,

%   inicializa as listas de ligacoes do agente
    assert( superior( IdAg , _ , 0 ) ) ,
    assert( subordinado( IdAg , _ , 0 ) )

%   obs.: esse agente nao possuira' subordinados

% fim inicializa_det
  .

% ___________________
% CONJUNCAO/COORDENACAO

  inicializa_coord( IdAg , Pos , Palavra , InfoDic ) :-

%   insere o novo agente na base de dados prolog
    assert( agente( IdAg , coordenacao , Palavra , InfoDic , [] ) ) ,

%   acrescenta o agente na "tabela" de posicoes
    acrescimoAgente( IdAg , Pos ) ,
```

```
%    inicializa as listas de ligacoes do agente
     assert( superior( IdAg , _ , 0 ) ) ,
     assert( subordinado( IdAg , dir , 0 ) ) ,
     assert( subordinado( IdAg , esq , 0 ) )

% fim inicializa_coord
  .

% _____________
% PRONOME RELATIVO

  inicializa_pron_rel( IdAg , Pos , Palavra , InfoDic ) :-

%    insere o novo agente na base de dados prolog
     assert( agente( IdAg , pron_rel , Palavra , InfoDic , [] ) ) ,

%    acrescenta o agente na "tabela" de posicoes
     acrescimoAgente( IdAg , Pos ) ,

%    inicializa as listas de ligacoes do agente
     assert( superior( IdAg , _ , 0 ) ) ,
     assert( subordinado( IdAg , _ , 0 ) )

% fim inicializa_pron_rel
  .


% ------------------------------------------------------------------
%    PREDICADOS DE CRIACAO DE AGENTES
% ------------------------------------------------------------------

% ------------------------------------------------------------------
% criacaoAgente( TipoAgt , AgtModelo , PosCriacao , IdAgtCriado ) :- esse
%        predicado ira' criar um agente do tipo "TipoAgt" na posicao espe-
%        cificada por "PosCriacao" pegando os dados do "AgtModelo".
%        O identificador do novo agente e' retornado em "IdAgtCriado".


% _______
% VERBAL

  criacaoAgente( verbal , AgtModelo , PosCriacao , NovoNumAgts ) :-

%    pega o numero de agentes existentes no presente momento
     numAgts( NumAgts ) ,

%    pega as informacoes do agente modelo "AgtModelo"
     agente( AgtModelo , verbal , Palavra , InfoDic , _ ) ,

%    chama o predicado que faz a inicializacao
%    (que ja' cuida de por os "precisos" na memoria)
     NovoNumAgts is NumAgts + 1 ,
     inicializa_verb( NovoNumAgts , PosCriacao , Palavra , InfoDic ) ,

%    inicializa uma nova lista para a funcao de aleatoriedade
     inicializaListasAleatorias ,

     !

% fim criacaoAgente / verbal
  .


% ________
% NOMINAL

  criacaoAgente( nominal , AgtModelo , PosCriacao , NovoNumAgts ) :-

%    pega o numero de agentes existentes no presente momento
     numAgts( NumAgts ) ,

%    pega as informacoes do agente modelo "AgtModelo"
     agente( AgtModelo , nominal , Palavra , InfoDic , _ ) ,

%    chama o predicado que faz a inicializacao
%    (que ja' cuida de por os "precisos" na memoria)
     NovoNumAgts is NumAgts + 1 ,
     inicializa_nom( NovoNumAgts , PosCriacao , Palavra , InfoDic ) ,

%    inicializa uma nova lista para a funcao de aleatoriedade
     inicializaListasAleatorias ,

     !

% fim criacaoAgente / nominal
  .


% ------------------------------------------------------------------
% Acrescimo de um Agente
%       Ha' dois momentos:
%         1) Na fase de inicializacao
%         2) Apos o inicio do processamento


 acrescimoAgente( IdAg , Pos ) :-

     numAgts( NumAgs ) ,                    % obtem o numero de agentes
     (   Pos > NumAgs ,                     % se vale a desigualdade
               acrescimoInicializacao( IdAg , Pos )
     ;   acrescimoAposInicio( IdAg , Pos )  % se nao vale a igualdade
```

```
    )

% fim acrescimoAgente/2
  .


% ----
  acrescimoInicializacao( IdAg , Pos ) :-
     assert( posicao( Pos , IdAg ) ) ,
     numAgts( NumAgts ) ,
     retractall( numAgts( _ ) ) ,
     NovoNumAgts is NumAgts + 1 ,
     assert( numAgts( NovoNumAgts ) )
  .

  acrescimoAposInicio( IdAg , Pos ) :-

     numAgts( NumAgs ) ,
     acrescimoAgente( IdAg , Pos , NumAgs )

% fim acrescimoAposInicio /2
  .


% ------ CASO BASE
% Insere o novo agente
%
  acrescimoAgente( IdAg , Pos , PosMenos_1 ) :-

     PosMenos_1 is Pos - 1 ,
     assert( posicao( Pos , IdAg ) ) ,
     numAgts( NumAgts ) ,
     retractall( numAgts( _ ) ) ,
     NovoNumAgts is NumAgts + 1 ,
     assert( numAgts( NovoNumAgts ) ) ,
     !

% acrescimoAgente /3
  .


% ------ CASO RECURSAO / REPOSICIONAMENTO
% Reposiciona os agentes que ja' existiam
%
  acrescimoAgente( IdAg , Pos , PosAtual ) :-

     posicao( PosAtual , IdAgAtual ) ,    % obtem o id do agente da PosAtual
     NovaPos is PosAtual + 1 ,            % incrementa o posicao
     retract( posicao( PosAtual , IdAgAtual )) ,  % reposiciona o agente
     assert( posicao( NovaPos , IdAgAtual )) ,    % para a nova posicao
     PosAtual2 is PosAtual - 1 ,
     acrescimoAgente( IdAg , Pos , PosAtual2 )

% fim acrescimoAgente /3
  .


% ---------------------------------------------------------------------
%    PREDICADOS "ACORDA" DOS AGENTES
% ---------------------------------------------------------------------


% ----------------------
% Acorda( agente , tipo do agente ) :-
%       acorda o agente para ver se ele esta' sendo necessitado
%       por outro(s) e/ou para buscar algum agente para preenche-lo
%       ou ser preenchido por ele.



% ------------------------
% Acorda para AGENTE NOMINAL /* caso normal */

  acorda( AgtAcordado , nominal ) :-

%    verifica se nao e' o caso eliptico
     not( gapping( AgtAcordado ) ) ,

%        verifica se ele nao possui nenhum superior
     (   superior( AgtAcordado , _ , 0 ) ,

%        obtem a restricao semantica do agente acordado
            agente( AgtAcordado , _ , _ , InfoDic , _Signif ) ,
            retornaValorAtributo( significado , InfoDic , Significados ) ,
            retornaTodosValores( tip_sem , Significados , ListaRestrSem ) ,

%        trata as mensagens "preciso" disponiveis
            trataMsgsPrecisos( AgtAcordado , ListaRestrSem , 0 , NumMsgs ) ,

%           se nao havia nenhum "preciso"
              (   NumMsgs = 0 ,

                  /* verifica se as condicoes de gapping sao satisfeitas */
                  /* as condicoes com as quais estou trabalhando sao:
                  - nao haver vizinho "verbal"
                  - a sentenca ser coordenada com um "e"
                   */
                  verificaGapping( AgtAcordado , HaGapping ) ,
```

```
                (   /* caso existam as restricoes que indicam 'gapping' */
                    HaGapping ,
                    /* entao devera' ser tratado no acorda eliptico */
                    assert( gapping( AgtAcordado ) ) ,
                    /* indica que houve uma acao executada */
                    fezAlgo( AgtAcordado )

                ;   /* nao faz nada */
                    true
                )

            ;   /* se havia algum preciso, entao o agente votou nele */
                NumMsgs \= 0 ,
                fezAlgo( AgtAcordado )
            )
    ;   /*  ja' possui superior */
            true
    ) ,
    (   /* se possuir um superior ... */
            not( superior( AgtAcordado , _ , 0 ) ) ,
            /* ... entao trata as mensagens de termino */
            tratamentoDeMsgTermino( AgtAcordado )

    ;   /* senao, continua */
            true
    ) ,
    (   /* se fez algo, apenas apaga a ocorrencia */
            agtFezAlgo( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
            not( agtFezAlgo( AgtAcordado ) ) ,
            naoFezNada( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ) .
    !

% fim acorda( nominal ) /caso normal
  .

% - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% VerificaGapping : - verifica se as condicoes de gapping pro agente nominal
%       sao satisfeitas

  verificaGapping( AgtAcordado , Gapping ) :-

%   obtem um vizinho `a esquerda que seja do tipo "coordenacao"
%   se ele nao existir, ira' retornar "AgtCoord = 0"
    procuraVizinho( AgtAcordado , menor ,
                    coordenacao , AgtCoord ) ,

%   verifica a nao existencia de um vizinho verbal dos dois lados
    procuraVizinho( AgtAcordado , menor , verbal , AgtVerbEsq ) ,
    procuraVizinho( AgtAcordado , maior , verbal , AgtVerbDir ) ,

%   obtem as posicoes dos agentes encontrados
    posicao( PosAgtCoord , AgtCoord ) ,
    posicao( PosAgtVerbEsq , AgtVerbEsq ) ,
%   obs.: nao e' preciso verificar posicao para o AgtVerbDir, pois so' e'
%         necessario saber que ele nao existe, ie., AgtVerbDir = 0

%   testa as possibilidades de existencia dos agentes e suas posicoes
    (   (   /* existe um agente verbal `a esquerda da conjuncao
                    implicando, e' claro, na existencia da conjuncao */
                    PosAgtCoord > PosAgtVerbEsq ,
                    PosAgtVerbEsq > 0 ,
                    /* nao existe um agente verbal `a direita do agente nominal */
                    AgtVerbDir = 0
            ) ,
%       entao, as condicoes foram satisfeitas
            iguala( Gapping , true )

    ;   /* outras possibilidades */
            iguala( Gapping , fail )

    ) ,
    !

% fim verificaGapping
  .

% ------------------------
% Acorda para AGENTE NOMINAL /* caso eliptico */

  acorda( AgtAcordado , nominal ) :-

%   verifica se nao e' o caso eliptico
    gapping( AgtAcordado ) ,

%   verifica se ele nao tem superior
    (   superior( AgtAcordado , _ , 0 ) ,

%       procura o agente coordenacao `a esquerda (usado p/ achar o ag. verbal)
            procuraVizinho( AgtAcordado , menor , coordenacao , AgtCoord ) ,

%       procura um agente verbal `a esquerda do agente coordenacao
            procuraVizinho( AgtCoord , menor , verbal , AgtVerbal ) ,

%       tem que verificar se AgtVerbal existe, (0 e' quando nao existe)
%    (obs.) isso ja' foi testado na hora em que o agente
```

```
%           decidiu por "gapping" mas vou deixar assim mesmo
              (   AgtVerbal \= 0 ,

%           verifica se ele esta' "colado" ao ag. coord (nom1) ou se
%           ha' um ag. nominal entre eles (nom2), como no esquema abaixo:
%           esquema:[ (... verbal ...) (coord) (nom1) (nom2) ]
                procuraVizinho( AgtAcordado , menor , nominal , AgtNom1 ) ,

%           obtem as posicoes de cada agente para ...
                posicao( PosAcordado , AgtAcordado ) ,
                posicao( PosCoord , AgtCoord ) ,
                posicao( PosNom1 , AgtNom1 ) ,

%           ... testa as posicoes entre o AgtAcordado, AgtCoord e AgtNom1
                (   /* se PosNom1 < PosCoord, e' porque AgtAcordado e' "nom1" */
                 PosNom1 < PosCoord ,

%                pede ao AgtVerbal a criacao de um clone seu na posicao dada
                    PosCriacao is PosAcordado + 1 ,
                    criacaoAgente( verbal , AgtVerbal , PosCriacao , _IdNovoAgt ) ,
                    /* indica que houve uma acao executada */
                    fezAlgo( AgtAcordado )

                ;   /* se PosNom1 > PosCoord, e' porque AgtAcordado e' "nom2" */
                 PosNom1 > PosCoord ,

%                pede ao AgtVerbal a criacao de um clone seu na posicao dada
                    PosCriacao is PosAcordado ,
                    criacaoAgente( verbal , AgtVerbal , PosCriacao , _IdNovoAgt ) ,
                    /* indica que houve uma acao executada */
                    fezAlgo( AgtAcordado )
                   )

              ;   /* nao faz nada */
                  true

              )
    ;   /* nao faz nada */
              true
    ) ,

    (   /* se possuir um superior ... */
              not( superior( AgtAcordado , _ , 0 ) ) ,
              /* ... entao trata as mensagens de termino */
              tratamentoDeMsgTermino( AgtAcordado )
    ;   /* senao, nao faz nada */
              true
    ) .

    retractall( gapping( _ ) ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
              agtFezAlgo( AgtAcordado ) ,
              retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
              not( agtFezAlgo( AgtAcordado ) ) ,
              naoFezNada( AgtAcordado ) ,
              retractall( agtFezAlgo( AgtAcordado ) )
    ) ,
    !

% fim acorda( nominal ) /caso eliptico

% ------------------------
% Acorda para AGENTE VERBAL

  acorda( AgtAcordado , verbal ) :-

%   para cada posicao pendente, apura os votos e tenta:
%   ou 1) estabelecer lacos ou 2) flexibilizar os "precisos"

%   verifica a posicao de sujeito
    (   pendencia( AgtAcordado , suj ) ,
              apuraVotos( AgtAcordado , suj , MelhorSuj ) ,
              (   MelhorSuj \= 0 ,
                  (   not( subordinado( AgtAcordado , MelhorSuj ) ) ,
                   /* verifica a nao existencia de um agente coor-
                      denacao entre o MelhorSuj e o AgtAcordado */
                   procuraVizinho( AgtAcordado , menor , coordenacao , AgtCoord1 )
.
                   posicao( PosAgtCoord1 , AgtCoord1 ) ,
                   posicao( PosAgtAcordado , AgtAcordado ) ,
                   posicao( PosMelhorSuj , MelhorSuj ) ,
                   (   (   igual( AgtCoord1 , 0 )
                       ;   not(  ( ( PosMelhorSuj < PosAgtCoord1 ) ,
                                  ( PosAgtAcordado > PosAgtCoord1 ) )
                                )
                       ;   ( ( PosMelhorSuj < PosAgtCoord1 ) ,
                             ( PosAgtAcordado > PosAgtCoord1 ) ) ,
                        flag_que
                       ) ,
                       /* estabelecimento de lacos */
                       estabLacos_Verb_Nom( AgtAcordado , MelhorSuj , suj , Ok1 )
                   )

                  ;   /* else */
                   igual( Ok1 , fail )
                  ) .
```

```
        (   Ok1 ,
            retractall( pendencia( AgtAcordado , suj ) ) ,
            retractall( subordinado( AgtAcordado , suj , _ ) ) ,
            assert( subordinado( AgtAcordado , suj , MelhorSuj ) ) ,
            retractall( voto( AgtAcordado , suj , _ , _ ) ) ,
            fezAlgo( AgtAcordado )
        ;   not( Ok1 ) ,
            /* retira o voto do que nao foi aceito e vai dormir */
            retract( voto( AgtAcordado , suj , _ , MelhorSuj ) ) ,
            fezAlgo( AgtAcordado )
        )

    ;   /* nao houve uma boa proposta para sujeito */
        MelhorSuj = 0 ,
        (   /* flexibilizacao - primeira vez */
            not( flexibilizouSuj( AgtAcordado ) ) ,
            flexibilizaSuj( AgtAcordado ) ,
            assert( flexibilizouSuj( AgtAcordado ) ) ,
            fezAlgo( AgtAcordado )

        ;   (   flexibilizouSuj( AgtAcordado ) ,
                not( flexibilizou_2_Suj( AgtAcordado ) )
            ) ,
            (   /* se e' uma frase com pronome relativo */
                flag_que ,
                /* tenta buscar o sujeito na vizinhanca */
                procuraVizinho( AgtAcordado , menor , nominal , VizSuj ) ,
                (   VizSuj \= 0 ,
                    /* verifica se o superior dele esta' entre eles */
                    obtemSuperior( VizSuj , SupVizSuj ) ,
                    (   SupVizSuj \= 0 ,
                        posicao( PosAgtAcordado , AgtAcordado ) ,
                        posicao( PosVizSuj , VizSuj ) ,
                        posicao( PosSupVizSuj , SupVizSuj ) ,
                        PosAgtAcordado > PosSupVizSuj
                        /* nao faz nada */

                    ;   % ou SupVizSuj = 0,
                        % ou (     ( PosSupVizSuj < PosAgtAcordado )
                        %      AND ( PosSupVizSuj > PosVizSuj ) )
                        /* ... pede a quebra de lacos entre eles */
                        pedidoQuebraLacos( SupVizSuj , VizSuj ) ,
                        /* pede estabelecimento de lacos com o VizSuj */
                          estabLacos_Verb_Nom( AgtAcordado , VizSuj , suj ,
Ok2_1 ) ,
                        (   Ok2_1 ,
                            retractall( pendencia( AgtAcordado , suj ) ) ,
                            retractall( subordinado( AgtAcordado , suj , _ ) ) ,
                            assert( subordinado( AgtAcordado , suj , VizSuj ) ) ,
                            retractall( voto( AgtAcordado , suj , _ , _ ) ) ,
                            fezAlgo( AgtAcordado )
                        ;   not( Ok2_1 )
                            /* continua */
                        )
                    )
                ;   /* nao existe VizSuj */
                    VizSuj = 0
                    /* nao faz nada */
                )
            ;   /* nao e' frase com "que" => continua */
                true
            ) ,

            assert( flexibilizou_2_Suj( AgtAcordado ) ) ,
            fezAlgo( AgtAcordado )

        ;   flexibilizou_2_Suj( AgtAcordado ) ,

%       se ha' pendencia para sujeito, mas nao ha' para objeto
%       e a sentenca e' coordenada `a esquerda (para sujeito),
%       entao houve um tipo de "reducao" na sentenca e devera'
%       ser criado o agente nominal que foi "apagado" nessa
%       "reducao". Um exemplo de sentenca desse tipo:
%       - joao caiu e ama maria (joao caiu e  joao  ama maria)
%                o que teria sido apagado => |____|

            (   pendencia( AgtAcordado , suj ) ,
                not( pendencia( AgtAcordado , obj ) ) ,
                procuraVizinho( AgtAcordado , menor , coordenacao , AgtCo-
ord2 ) ,
                AgtCoord2 \= 0 ,
                procuraVizinho( AgtCoord2 , menor , verbal , AgtVerbal ) ,
                AgtVerbal \= 0
            ) ,
            obtemSujeito( AgtVerbal , AgtSuj ) ,
            (   /* o agente verbal possui um sujeito */
                AgtSuj \= 0 ,
                /* o novo agente sera' inserido no lugar do agtAcordado */
                posicao( PosNovoAgt , AgtAcordado ) ,
                criacaoAgente( nominal , AgtSuj , PosNovoAgt , IdNovoAgt )

                /* estabelece lacos com este novo agente */
                estabLacos_Verb_Nom( AgtAcordado , IdNovoAgt , suj , Ok4 )

                retractall( pendencia( AgtAcordado , suj ) ) ,
                retractall( subordinado( AgtAcordado , suj , _ ) ) ,
                assert( subordinado( AgtAcordado , suj , IdNovoAgt ) ) ,
                fezAlgo( AgtAcordado )
            ;   AgtSuj = 0
                /* nao faz nada */
```

```
                )
               )
            )
    :   /* nao ha' pendencia para a posicao "sujeito" */
            /* continua */
            true
    ) ,

%   verifica a posicao de objeto
    (   pendencia( AgtAcordado , obj ) ,
            apuraVotos( AgtAcordado , obj , MelhorObj ) ,
            (   MelhorObj \= 0 ,
                (   not( subordinado( AgtAcordado , MelhorObj ) ) ,
                 /* verifica a nao existencia de um agente coor-
                    denacao entre o MelhorObj e o AgtAcordado */
                 procuraVizinho( AgtAcordado , maior , coordenacao , AgtCoord2 )
.
                 (   (   iguala( AgtCoord2 , 0 )
                     :   posicao( PosAgtCoord2 , AgtCoord2 ) ,
                      posicao( PosAgtAcordado , AgtAcordado ) ,
                      posicao( PosMelhorObj , MelhorObj ) ,
                      not(  ( ( PosMelhorObj > PosAgtCoord2 ) ,
                           ( PosAgtAcordado < PosAgtCoord2 ) )
                          )
                     ) ,
                     /* estabelecimento de lacos */
                     estabLacos_Verb_Nom( AgtAcordado , MelhorObj , obj , Ok2 )
                 )

                :   /* else */
                 iguala( Ok2 , fail )

                ) ,
                (   Ok2 ,
                 retractall( pendencia( AgtAcordado , obj ) ) ,
                 retractall( subordinado( AgtAcordado , obj , _ ) ) ,
                 assert( subordinado( AgtAcordado , obj , MelhorObj ) ) ,
                 retractall( voto( AgtAcordado , obj , _ , _ ) ) ,
                 fezAlgo( AgtAcordado )
                :   not( Ok2 ) ,
                 /* retira o voto do que nao foi aceito e vai dormir */
                 retract( voto( AgtAcordado , obj , _ , MelhorObj ) ) ,
                 fezAlgo( AgtAcordado )
                )

            :   /* nao houve uma boa proposta para objeto */
                MelhorObj = 0 ,
                (   /* flexibilizacao - primeira vez */
                 not( flexibilizouObj( AgtAcordado ) ) ,
                 flexibilizaObj( AgtAcordado ) ,
                 assert( flexibilizouObj( AgtAcordado ) ) ,
                 fezAlgo( AgtAcordado )

                :   (   flexibilizouObj( AgtAcordado ) ,
                    not( flexibilizou_2_Obj( AgtAcordado ) )
                 ) ,
                 (   /* se e' uma frase com pronome relativo */
                    flag_que ,
                    /* tenta buscar o objeto na vizinhanca */
                    procuraVizinho( AgtAcordado , maior , nominal , VizObj ) ,
                    (   VizObj \= 0 ,
                     /* verifica se o superior dele esta' entre eles */
                     obtemSuperior( VizObj , SupVizObj ) ,
                     (   SupVizObj \= 0 ,
                         posicao( PosSupVizObj , SupVizObj ) ,
                         posicao( PosVizObj , VizObj ) ,
                         posicao( PosAgtAcordado , AgtAcordado ) ,
                         PosSupVizObj < PosVizObj ,
                         PosAgtAcordado < PosSupVizObj
                         /* nao faz nada */

                     :   % ou SupVizObj = 0,
                         % ou o SupVizObj nao esta' entre o AgtAcordado
                         % e o VizObj
                         /* ... pede a quebra de lacos entre eles */
                         pedidoQuebraLacos( SupVizObj , VizObj ) ,
                         /* pede estabelecimento de lacos com o VizObj */
                           estabLacos_Verb_Nom( AgtAcordado , VizObj , obj ,
Ok2_1 ) ,
                         (   Ok2_1 ,
                          retractall( pendencia( AgtAcordado , obj ) ) ,
                          retractall( subordinado( AgtAcordado , obj , _ ) ) ,
                          assert( subordinado( AgtAcordado , obj , VizObj ) ) ,
                          retractall( voto( AgtAcordado , obj , _ , _ ) ) ,
                          fezAlgo( AgtAcordado )
                         :   not( Ok2_1 )
                          /* continua */
                         )
                     )
                    :   /* nao existe VizObj */
                     VizObj = 0
                     /* nao faz nada */
                    )
                :   /* nao e' frase com "que" => continua */
                    true
                ) ,
                /* mesmo assim, sinaliza que flexibilizou pela 2nd vez */
                assert( flexibilizou_2_Obj( AgtAcordado ) ) ,
                fezAlgo( AgtAcordado )
```

```
                    ;   flexibilizou_2_Obj( AgtAcordado ) .
%                 se ha' pendencia para objeto, mas nao ha' para sujeito
%                 e a sentenca e' coordenada `a direita (para objeto),
%                 entao houve um tipo de "reducao" na sentenca e devera'
%                 ser criado o agente nominal que foi "apagado" nessa
%                 "reducao". Um exemplo de sentenca desse tipo:
%                  - joao ama e maria detesta pedro
%                      (joao ama  pedro  e maria detesta pedro)
%                                |_____| => o que teria sido apagado

                    (   pendencia( AgtAcordado , obj ) ,
                        not( pendencia( AgtAcordado , suj ) ) ,
                        procuraVizinho( AgtAcordado , maior , coordenacao , AgtCo-
ord3 ) ,

                        AgtCoord3 \= 0 ,
                        procuraVizinho( AgtCoord3 , maior , verbal , AgtVerbal ) ,
                        AgtVerbal \= 0
                    ) ,

                    obtemObjeto( AgtVerbal , AgtObj ) ,
                    (   /* o agente verbal possui um objeto */
                        AgtObj \= 0 ,
                        /* o novo agente sera' inserido na posicao do agtAcordado +
1 */

                        posicao( PosAgtAcorda , AgtAcordado ) ,
                        PosNovoAgt is PosAgtAcorda + 1 ,
                        criacaoAgente( nominal , AgtObj , PosNovoAgt , IdNovoAgt )
,

                        /* estabelece lacos com este novo agente */
                        estabLacos_Verb_Nom( AgtAcordado , IdNovoAgt , obj , Ok5 )
,

                        retractall( pendencia( AgtAcordado , obj ) ) ,
                        retractall( subordinado( AgtAcordado , obj , _ ) ) ,
                        assert( subordinado( AgtAcordado , obj , IdNovoAgt ) ) ,
                        fezAlgo( AgtAcordado )
                    ;   AgtObj = 0
                        /* nao faz nada */
                    )
                   )
                )
    ;   /* nao ha' pendencia para a posicao "objeto" */
                /* continua */
                true
    ) ,

%   verifica a posicao de objeto indireto
    (   pendencia( AgtAcordado , obj_ind ) ,
                apuraVotos( AgtAcordado , obj_ind , MelhorObjInd ) ,
                (   MelhorObjInd \= 0 ,
                    (   not( subordinado( AgtAcordado , MelhorObjInd ) ) ,
                        /* estabelecimento de lacos */
                        estabLacos_Verb_Nom( AgtAcordado , MelhorObjInd , obj_ind , Ok3
)

                    ;   /* o agente p/ sujeito ja' era subordinado do agtAcordado */
                        /* subordinado( AgtAcordado , MelhorObjInd ) */
                        iguala( Ok3 , fail )

                    ) ,
                    (   Ok3 ,
                        retractall( pendencia( AgtAcordado , obj_ind ) ) ,
                        retractall( subordinado( AgtAcordado , obj_ind , _ ) ) ,
                        assert( subordinado( AgtAcordado , obj_ind , MelhorObjInd ) ) ,
                        retractall( voto( AgtAcordado , obj_ind , _ , _ ) ) ,
                        fezAlgo( AgtAcordado )
                    ;   not( Ok3 ) ,
                        retract( voto( AgtAcordado , obj_ind , _ , MelhorObjInd ) ) ,
                        fezAlgo( AgtAcordado )
                    )

                ;   /* nao houve uma boa proposta para objeto indireto */
                    MelhorObjInd = 0 ,
                    (   /* flexibilizacao - primeira vez */
                        not( flexibilizouObjInd( AgtAcordado ) ) ,
                        flexibilizaObjInd( AgtAcordado ) ,
                        assert( flexibilizouObjInd( AgtAcordado ) ) ,
                        fezAlgo( AgtAcordado )

                    ;   (   flexibilizouObjInd( AgtAcordado ) ,
                            not( flexibilizou_2_ObjInd( AgtAcordado ) )
                        ) ,
                        assert( flexibilizou_2_ObjInd( AgtAcordado ) ) ,
                        fezAlgo( AgtAcordado )

                    ;   flexibilizou_2_ObjInd( AgtAcordado )
                    )
                )
    ;   /* nao ha' pendencia para a posicao "objeto indireto" */
                /* continua */
                true
    ) ,

    (   /* se nao houver pendencia para nada e nao possuir */
                /* um subordinado "que", sinaliza que terminou     */
                (   not( pendencia( AgtAcordado , suj ) ) ,
                    not( pendencia( AgtAcordado , obj ) ) ,
                    not( pendencia( AgtAcordado , obj_ind ) ) ,
                    not( terminei( AgtAcordado , _ ) ) ,
                    not( subordinado_que( AgtAcordado ) )
                ) ,
                assert( terminei( AgtAcordado , [AgtAcordado] ) )
```

```
    ;   (   (   subordinado_que( AgtAcordado )
                ;   not( superior( AgtAcordado , _ , 0 ) )
                ) ,
                not( pendencia( AgtAcordado , suj ) ) ,
                not( pendencia( AgtAcordado , obj ) ) ,
                not( pendencia( AgtAcordado , obj_ind ) )
            ) ,
            tratamentoDeMsgTermino( AgtAcordado )
    ;   /* ainda ha' pendencias */
            /* apenas vai dormir */
            true
    ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
            agtFezAlgo( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
            not( agtFezAlgo( AgtAcordado ) ) ,
            naoFezNada( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ) ,

    !

% fim acorda( verbal )
  .


% ----------------------------
% Acorda para AGENTE COORDENACAO

  acorda( AgtAcordado , coordenacao ) :-

%   O "flag" coordenacao( X ) esta' sendo usado para designar que o
%   agente ja' encontrou seus pares e que devera' se comportar como
%   um agente do tipo X. Na realidade, ha' diferentes tipos de "acorda"
%   para o agente coordenacao (implementando os diferentes comporta-
%   mentos) e este "flag" serve para selecionar o "acorda" que deve
%   ser utilizado. Neste "acorda", os pares ainda nao foram encontrados.

    (   not( coordenacao( nominal ) ) ,
            not( coordenacao( verbal ) )
    ) ,

    (   /* ainda esta' sem pelo menos um subordinado */
            subordinado( AgtAcordado , _ , 0 ) ,

%       procura por um vizinho pronome relativo dos dois lados
            procuraVizinho( AgtAcordado , maior , pron_rel , VizPrRelDir ) ,
            procuraVizinho( AgtAcordado , menor , pron_rel , VizPrRelEsq ) ,

        (   /* existe pelo menos um agente pronome relativo */
            /*  => coordenacao entre agentes nominais */
            (   VizPrRelDir \= 0
            ;   VizPrRelEsq \= 0
            ) ,
%           Pega o vizinho da direita (N3) para saber se ele possui supe-
%           rior. Se ja' possuir um superior (verbal - V2), e' porque ele
%           e' o "sujeito" deste verbo, e o verbo ja' estara' concordando
%           (concordancia de pessoa e numero) com este sujeito. Assim a
%           coordenacao so' podera' se dar entre os vizinhos mais proximos
%           (porque, sendo a coordenacao entre nominais, esta so' podera'
%           ser "objeto", onde nao ha' necessidade de concordancia) com o
%           verbo a esquerda
%
%           Esquema inicial:
%           N1 PronRel V1 N2 C N3 V2
%                           |___|(suj)
%
%           Resultado (entre parenteses):
%           N1 PronRel V1 (N2 C N3) V2
%                     |_____|(obj)
%
            procuraVizinho( AgtAcordado , maior , nominal , VizNomDir ) ,
            (   /* o vizinho da direita possui superior */
                (   obtemSuperior( VizNomDir , SupVizNomDir ) ,
                    SupVizNomDir \= 0
                ) ,
%               pede quebra de lacos entre o VizNomDir e seu superior
                pedidoQuebraLacos( SupVizNomDir , VizNomDir ) ,

%               pega o vizinho da esquerda
                procuraVizinho( AgtAcordado , menor , nominal , VizNomEsq ) ,
                obtemSuperior( VizNomEsq , SupVizNomEsq ) ,
                (   SupVizNomEsq \= 0 ,
                    pedidoQuebraLacos( SupVizNomEsq , VizNomEsq )
                ;   /* continua */
                    true
                ) ,

%               estabelece ligacoes com este vizinho da direita
                estabLacos_Coord_Nom( AgtAcordado , VizNomEsq ,
                                      VizNomDir , Ok1 ) ,

                (   Ok1 ,
%                   estabelece para si a nova ligacao
                    retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                    assert( subordinado( AgtAcordado , dir , VizNomDir ) ) ,
                    assert( subordinado( AgtAcordado , esq , VizNomEsq ) ) ,
```

```
%               sinalizacao de que ele devera' se comportar como um
%               agente nominal. Isso e' checado no comeco do "acorda".
%               Ha' um "acorda" especifico para esse comportamento
%               "nominal" do agente coordenacao
                    assert( coordenacao( nominal ) ) ,
                    fezAlgo( AgtAcordado )

                ;   not( Ok1 )
%               write( 'ERRO: estabelecimento de lacos nao aceito' ) , nl ,
%               write( 'entre o agente coord (' ) , write( AgtAcordado ) ,
%               write( ') e os agentes nominais (' ) , write( VizNomEsq ) ,
%               write( ') e (' ) , write( VizNomDir ) , write( ')' ) ,
%               nl , write( 'Local: Ok 1' ) , nl
                )

               ;   /* o vizinho da direita NAO POSSUI superior */
                (   obtemSuperior( VizNomDir , SupVizNomDir ) ,
                    SupVizNomDir = 0
                ) .

%           pega o vizinho nominal da esquerda
                (   /* se nao existe um pronome relativo `a esquerda */
                    VizPrRelEsq = 0 ,
                    /* entao o vizinho e' o primeiro */
                    procuraVizinho( AgtAcordado , menor , nominal , VizNomEsq )

                ;   /* existe um pronome relativo `a esquerda */
                    VizPrRelEsq \= 0 ,
                    /* entao o vizinho e' o que esta' `a esquerda do "que" */
                    procuraVizinho( VizPrRelEsq , menor , nominal , VizNomEsq )
                ) ,

%           estabelece ligacoes com este vizinho da direita
                estabLacos_Coord_Nom( AgtAcordado , VizNomEsq ,
                              VizNomDir , Ok2 ) ,

                (   Ok2 ,
%               estabelece para si a nova ligacao
                    retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                    assert( subordinado( AgtAcordado , dir , VizNomDir ) ) ,
                    assert( subordinado( AgtAcordado , esq , VizNomEsq ) ) ,

                    assert( coordenacao( nominal ) ) ,
                    fezAlgo( AgtAcordado )

                ;   not( Ok2 ) ,
                    write( 'ERRO: estabelecimento de lacos nao aceito' ) , nl ,
                    write( 'entre o agente coord (' ) , write( AgtAcordado ) ,
                    write( ') e os agentes nominais (' ) , write( VizNomEsq ) ,
                    write( ') e (' ) , write( VizNomDir ) , write( ')' ) ,
                    nl , write( 'Local: Ok 2' ) , nl
                )
               )

            ;   /* nao existem agentes pronome relativo */
                (   VizPrRelDir = 0 , VizPrRelEsq = 0
                ) .

%           procura agentes verbais dos dois lados
                procuraVizinho( AgtAcordado , maior , verbal , VizVerbDir ) ,
                procuraVizinho( AgtAcordado , menor , verbal , VizVerbEsq ) ,

                (   /* existencia de APENAS 1 verbo */
                 (   VizVerbDir = 0 , VizVerbEsq \= 0
                 ;   VizVerbDir \= 0 , VizVerbEsq = 0
                 ) ,
                 (   /* verbo `a direita => coordenacao entre nominais */
                     VizVerbEsq = 0 ,

%                pega os agentes nominais `a esquerda e `a direita
                     procuraVizinho( AgtAcordado , maior , nominal , VizNomDir )
.
                     procuraVizinho( AgtAcordado , menor , nominal , VizNomEsq )
,

%                estabelecimento de lacos com os agentes nominais
                     estabLacos_Coord_Nom( AgtAcordado , VizNomEsq ,
                                 VizNomDir , Ok3 ) ,

                     (   Ok3 ,
%                    estabelece para si a nova ligacao
                      retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                      assert( subordinado( AgtAcordado , dir , VizNomDir ) ) ,
                      assert( subordinado( AgtAcordado , esq , VizNomEsq ) ) ,

                      assert( coordenacao( nominal ) ) ,
                      fezAlgo( AgtAcordado )

                     ;   not( Ok3 ) ,
                      write( 'ERRO: estabelecimento de lacos nao aceito' ) , nl
,
                      write( 'entre o agente coord (' ) , write( AgtAcordado ) ,
                      write( ') e os agentes nominais (' ) , write( VizNomEsq )
,
                      write( ') e (' ) , write( VizNomDir ) , write( ')' ) ,
                      nl , write( 'Local: Ok 3' ) , nl
                     )

                 ;   /* verbo `a esquerda */
                     VizVerbEsq \= 0 ,
```

```
                    (   /* nao ha' gapping => coordenacao entre nominais */
                     (
%                       pega os agentes nominais `a esquerda e `a direita
                          procuraVizinho( AgtAcordado , maior , nominal , Vi-
zNomDir ) ,
                          procuraVizinho( AgtAcordado , menor , nominal , VizNo-
mEsq ) ,

%                       e verifica a nao existencia de mais de um agente
%                       nominal mais `a direita
                          procuraVizinho( VizNomDir , maior , nominal , VizNomA-
lemDir ) ,

                          VizNomAlemDir = 0
                     ) ,

%                    estabelecimento de lacos com os agentes nominais
                     estabLacos_Coord_Nom( AgtAcordado , VizNomEsq ,
                                   VizNomDir , Ok4 ) ,

                     (   Ok4 ,
%                       estabelece para si a nova ligacao
                         retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                         assert( subordinado( AgtAcordado , dir , VizNomDir ) )
.
                         assert( subordinado( AgtAcordado , esq , VizNomEsq ) )
.

                         assert( coordenacao( nominal ) ) ,
                         fezAlgo( AgtAcordado )

                     ;   not( Ok4 ) ,
                         write( 'ERRO: estabelecimento de lacos nao aceito' ) ,
nl ,
                         write( 'entre o agente coord (' ) , write( AgtAcordado
) ,
                          write( ') e os agentes nominais (' ) , write( VizNo-
mEsq ) ,
                         write( ') e (' ) , write( VizNomDir ) , write( ')' ) ,
                         nl , write( 'Local: Ok 4' ) , nl
                     )

                    ;   /* ha' gapping => coordenacao entre verbais */
                     /* vai dormir para esperar que */
                     /* o agente verbal seja criado */
                     true
                    )
                )

               :   /* existencia de DOIS verbos
                /* coordenacao entre agentes verbais */
                (   VizVerbDir \= 0 , VizVerbEsq \= 0
                    ) .

%                   estabelecimento de lacos com os dois agentes verbais
                        estabLacos_Coord_Verb( AgtAcordado , VizVerbEsq ,
                                    VizVerbDir , Ok5 ) ,

                        (   Ok5 ,
%                       estabelece para si a nova ligacao
                            retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                            assert( subordinado( AgtAcordado , dir , VizVerbDir ) ) ,
                            assert( subordinado( AgtAcordado , esq , VizVerbEsq ) ) ,

                            /* veja logo acima comentario sobre este assert */
                            /* so' que com "nominal" ao inves de "verbal" */
                            assert( coordenacao( verbal ) ) ,
                            fezAlgo( AgtAcordado )

                        ;   not( Ok5 ) ,
                            write( 'ERRO: estabelecimento de lacos nao aceito' ) , nl ,
                            write( 'entre o agente coord (' ) , write( AgtAcordado ) ,
                            write( ') e os agentes verbais (' ) ,
                            write( VizVerbEsq ) , write( ') e (' ) ,
                            write( VizVerbDir ) , write( ')' ) ,
                            nl , write( 'Local: Ok 5' ) , nl
                        )
                       )

                   :   /* ocorreu algum erro, continua */
                       true
                   )

               ;   /* ja' possui seus subordinados => continua */
                       true
               ) .

               (   /* se fez algo, apenas apaga a ocorrencia */
                       agtFezAlgo( AgtAcordado ) ,
                       retractall( agtFezAlgo( AgtAcordado ) )
               :   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
                       not( agtFezAlgo( AgtAcordado ) ) ,
                       naoFezNada( AgtAcordado ) ,
                       retractall( agtFezAlgo( AgtAcordado ) )
               ) .

               !

% fim acorda( coordenacao )
  .
```

```
acorda( AgtAcordado , coordenacao ) :-

%    E' uma coordenacao de verbos. Ver um comentario sobre
%    isso no primeiro "acorda" para coordenacao, logo acima.
     coordenacao( verbal ) ,

%    trata as mensagens de termino
     tratamentoDeMsgTermino( AgtAcordado ) ,
     naoFezNada( AgtAcordado ) ,

     !

% fim acorda( coordenacao ) / verbal
  .

acorda( AgtAcordado , coordenacao ) :-

%    E' uma coordenacao de nomes. Ver um comentario sobre
%    isso no primeiro "acorda" para coordenacao, logo acima.
     coordenacao( nominal ) ,

%        verifica se ele nao possui nenhum superior
     (   superior( AgtAcordado , _ , 0 ) ,

%        obtem a restricao semantica do agente acordado
            agente( AgtAcordado , _ , _ , InfoDic , _Signif ) ,
            retornaValorAtributo( significado , InfoDic , Significados ) ,
            retornaTodosValores( tip_sem , Significados , ListaRestrSem ) ,

%        trata as mensagens "preciso" disponiveis
            trataMsgsPrecisos( AgtAcordado , ListaRestrSem , 0 , NumMsgs ) ,

%           se nao havia nenhum "preciso"
            (   NumMsgs = 0 ,
                /* continua */
                true

            ;   /* se havia algum preciso, entao o agente votou nele */
                NumMsgs \= 0 ,
                fezAlgo( AgtAcordado )
            )
     ;   /*  ja' possui superior */
            true
     ) ,
     (   /* se possuir um superior ... */
            not( superior( AgtAcordado , _ , 0 ) ) ,
            /* ... entao trata as mensagens de termino */
            tratamentoDeMsgTermino( AgtAcordado )

     ;   /* senao, continua */
            true
     ) ,

     (   /* se fez algo, apenas apaga a ocorrencia */
            agtFezAlgo( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
     ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
            not( agtFezAlgo( AgtAcordado ) ) ,
            naoFezNada( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
     ) ,

     !

% fim acorda( coordenacao ) / verbal
  .



% ----------------------------
% Acorda para AGENTE PREPOSICAO

  acorda( AgtAcordado , prep ) :-

     not( segundaInterpretacao( AgtAcordado ) ) ,

     (   /* se nao possui subordinado */
            subordinado( AgtAcordado , _ , 0 ) ,

%        encontra o agente nominal `a sua direita
            procuraVizinho( AgtAcordado , maior , nominal , AgtNomDir ) ,

               (   /* encontrou um agente nominal */
                  AgtNomDir \= 0 ,
%           pega as restricoes semanticas do agente acordado
                  agente( AgtAcordado , _ , _ , InfoDic , Signif ) ,
                  retornaValorAtributo( significado , InfoDic , ListaSignif ) ,
                  retornaTodosValores( tip_sem , ListaSignif , RestrSemAA ) ,

%           verifica se o agente nominal nao possui superior
%           se possuir, pede a quebra da ligacao
                  obtemSuperior( AgtNomDir , SupAgtNomDir ) ,
                  (   SupAgtNomDir \= 0 ,
                   pedidoQuebraLacos( SupAgtNomDir , AgtNomDir )
                  ;   SupAgtNomDir = 0
                   /* continua */
                  ) ,

%           pede estabelecimento de lacos
                  estabLacos_1_Prep_Nom( AgtAcordado , RestrSemAA ,
```

```
                        AgtNomDir , RestrSemAN , Ok1 ) .
%               /---------------------< |_________|
%               \---> A restricao semantica do agente nominal e' retornada

                  (   Ok1 ,

                   retractall( subordinado( AgtAcordado , _ , _ ) ) .
                   assert( subordinado( AgtAcordado , prep , AgtNomDir ) ) .

%                 coloca no agente os dados do tipo semantico do complemento
                      /* pega os dados do agente, enquanto os retira da memoria */
                      retract( agente( AgtAcordado , prep ,
                               Palavra , InfoDic , Signif ) ) .
                      /* altera o significado, inserindo os dados do complemento */
                      insert( tip_sem=[RestrSemAN] , Signif , NovoSignif ) .
                      /* insere os dados na memoria novamente */
                      assert( agente( AgtAcordado , prep ,
                                Palavra , InfoDic , NovoSignif ) ) .
                      fezAlgo( AgtAcordado )

                     ;   not( Ok1 ) .
                      /* continua */
                      true
                     )

                ;   /* nao existe agente nominal `a direita do agente prep */
                    AgtNomDir = 0 .
                    /* continua */
                    true
                )
    ;   /* se ja' possui subordinado, entao continua */
            true
    ) .

    (   /* ja' possui subordinado, mas nao possui superior */
            (   not( subordinado( AgtAcordado , _ , 0 ) ) .
                superior( AgtAcordado , _ , 0 )
            ) .

%       pega as restricoes semanticas relativas ao seu complemento nominal
            agente( AgtAcordado , _ , Palavra , _ , Significado ) .
            retornaValorAtributo( tip_sem, Significado , RestrSemAP ) .

%       trata as mensagens "preciso"
            trataMsgsPrecisos( AgtAcordado , RestrSemAP , 0 , NumMsgs ) .

            (   /* se o numero de mensagem for diferente de zero, nao faz nada
*/
                NumMsgs \= 0 .
                fezAlgo( AgtAcordado )    /* tratou as mensagens */
```

```
        ;   /* se for igual a zero, tenta encontrar seu superior */
            NumMsgs = 0 .

%       encontra um agente verbal e um nominal `a sua esquerda
            procuraVizinho( AgtAcordado , menor , verbal , AgtVerb ) .
            procuraVizinho( AgtAcordado , menor , nominal , AgtNomEsq ) .
%       pega suas posicoes para verificar as possibilidades logo abaixo
            posicao( PosAgtVerb , AgtVerb ) .
            posicao( PosAgtNomEsq , AgtNomEsq ) .

%       possibilidades
            (   /* existe um agente verbal e ele esta' */
                /* na frente de um possivel nominal    */
                PosAgtVerb > PosAgtNomEsq .
%           a unica opcao e' estabelecer lacos com o agente verbal
                estabLacos_Prep_Verb( AgtAcordado , Palavra , RestrSemAP ,
                                AgtVerb , RestrSemRetornada , Ok2 ) .
                (   Ok2 ,
%               atualiza o superior
                    retractall( superior( AgtAcordado , _ , 0 ) ) .
                    assert( superior( AgtAcordado , verb , AgtVerb ) ) .
%               atualiza a restricao semantica que o agente possui

                    fezAlgo( AgtAcordado )

                   ;   not( Ok2 ) .
                    /* continua */
                    true
                )

                ;  /* existem duas possibilidades */
                (   PosAgtVerb < PosAgtNomEsq .
                    AgtVerb \= 0
                ) .
%           tenta estabelecer lacos com o agente verbal
                estabLacos_Prep_Verb( AgtAcordado , Palavra , RestrSemAP ,
                                AgtVerb , RestrSemRetornada , Ok3 ) .

                (   Ok3 .
%               atualiza o superior
                    retractall( superior( AgtAcordado , _ , 0 ) ) .
                    assert( superior( AgtAcordado , verb , AgtVerb ) ) .
%               e sinaliza que existe outra interpretacao
                    assert( outraInterpretacao( AgtAcordado ) ) .
%               assert( primeiraInterpretacao( AgtAcordado ) ) .
%               modifica o agente acordado assinalando a semantica final
                    retract( agente( AgtAcordado , prep ,
                              Palavra , InfoDic , Sig ) ) .
                    iguala( NovoSig , tip_sem=[RestrSemRetornada] ) .
```

```
                    assert( agente( AgtAcordado , prep , Palavra ,
                              InfoDic , NovoSig ) ) ,
                    fezAlgo( AgtAcordado )

                ;   not( Ok3 ) ,
%               estabelece lacos com o agente nominal
                    estabLacos_2_Prep_Nom( AgtAcordado , AgtNomEsq , Ok4 ) ,
                    (   Ok4 ,
%                   atualiza o superior
                     retractall( superior( AgtAcordado , _ , 0 ) ) ,
                     assert( superior( AgtAcordado , nom , AgtNomEsq ) ) ,
                     fezAlgo( AgtAcordado )
                    ;   not( Ok4 ) ,
                     /* continua */
                     true
                    )
                )

               ;   /* existe apenas a possibilidade do agente nominal */
                   PosAgtVerb < PosAgtNomEsq ,

%              estabelece lacos com o agente nominal
                   estabLacos_2_Prep_Nom( AgtAcordado , AgtNomEsq , Ok5 ) ,

                   (   Ok5 ,
%                  atualiza o superior
                       retractall( superior( AgtAcordado , _ , 0 ) ) ,
                       assert( superior( AgtAcordado , nom , AgtNomEsq ) ) ,
                       fezAlgo( AgtAcordado )
                   ;   not( Ok5 ) ,
                       /* continua */
                       true
                   )

                  ;   /* nao existe nenhuma das possibilidades acima */
                   /* entao apenas continua */
                   true
                  )
              )
    ;    /* se ja' possui subordinado e ja' possui superior, */
           /* entao continua para tratar termino */
           true
    ) ,

    (   not( superior( AgtAcordado , _ , 0 ) ) ,
          tratamentoDeMsgTermino( AgtAcordado )
    ;   superior( AgtAcordado , _ , 0 )
          /* continua */
    ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
          agtFezAlgo( AgtAcordado ) ,
          retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
          not( agtFezAlgo( AgtAcordado ) ) ,
          naoFezNada( AgtAcordado ) ,
          retractall( agtFezAlgo( AgtAcordado ) )
    ) ,

    !

% fim acorda( preposicao ) / caso "nao tem outra interpretacao"
  .

% ------------------------------------
  acorda( AgtAcordado , prep ) :-

    segundaInterpretacao( AgtAcordado ) ,
    (   /* ja' possui subordinado, mas nao possui superior */
          (   not( subordinado( AgtAcordado , _ , 0 ) ) ,
              superior( AgtAcordado , _ , 0 )
          ) ,
          procuraVizinho( AgtAcordado , menor , nominal , AgtNomEsq ) ,
%       estabelece lacos com o agente nominal
          estabLacos_2_Prep_Nom( AgtAcordado , AgtNomEsq , Ok ) ,
          (   Ok ,
%           atualiza o superior
              retractall( superior( AgtAcordado , _ , 0 ) ) ,
              assert( superior( AgtAcordado , nom , AgtNomEsq ) ) ,
              fezAlgo( AgtAcordado )
          ;   not( Ok ) ,
              /* continua */
              true
          )

    ;   /* senao, continua */
          true
    ) ,

    (   not( superior( AgtAcordado , _ , 0 ) ) ,
          tratamentoDeMsgTermino( AgtAcordado )
    ;   superior( AgtAcordado , _ , 0 )
          /* continua */
    ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
          agtFezAlgo( AgtAcordado ) ,
          retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
```

```
                not( agtFezAlgo( AgtAcordado ) ) ,
                naoFezNada( AgtAcordado ) ,
                retractall( agtFezAlgo( AgtAcordado ) )
    ) ,

    !

% fim acorda( prep )
  .


% ------------------------------------
  acorda( AgtAcordado , prep ) :-

    not( primeiraInterpretacao( AgtAcordado ) ) ,




    !

% fim acorda( prep )
  .



% ---------------------------------
% Acorda para AGENTE PRONOME RELATIVO

  acorda( AgtAcordado , pron_rel ) :-

%   processamento dos pronomes relativos da direita para esquerda
%   como se fosse uma prioridade
    procuraVizinho( AgtAcordado , maior , pron_rel , AgtPrRelDir ) ,

    (   not( relativo( AgtAcordado ) ) ,
            AgtPrRelDir \= 0 ,
            assert( relativo( AgtAcordado ) ) ,
            acorda( AgtPrRelDir , pron_rel )

    ;   /* continua */
            true
    ) ,

%       nao possui subordinado
    (   subordinado( AgtAcordado , _ , 0 ) ,

%       encontra o agente nominal `a sua esquerda
            procuraVizinho( AgtAcordado , menor , nominal , AgtNom ) ,
            (   /* encontrou um agente nominal */
                AgtNom \= 0 ,
%           pede estabelecimento de lacos
                estabLacos_PrRel_Nom( AgtAcordado , AgtNom , Ok1 ) ,

                (   Ok1 ,
                 retractall( subordinado( AgtAcordado , _ , _ ) ) ,
                 assert( subordinado( AgtAcordado , que , AgtNom ) ) ,
                 fezAlgo( AgtAcordado )

                ;   not( Ok1 ) ,
                 write( 'ERRO: acorda pron. rel. (' ) ,
                 write( AgtAcordado ) ,
                 write( ') / estab. lacos nominal ' ) ,
                 write( AgtNom ) , write( ' nao aceito' ) ,
                 nl
                )

            ;   /* nao existe agente nominal `a esquerda do agente pron-rel */
                AgtNom = 0 , true
%           write( 'ERRO: acorda pron. rel. (' ) ,
%           write( AgtAcordado ) , write( ') / nao existe agente nominal' ) ,
%           nl
            )
    ;   /* ja' possui subordinado, isto e', continua */
            true
    ) ,

    (   /* ja' possui subordinado, mas nao possui superior */
            (   not( subordinado( AgtAcordado , _ , 0 ) ) ,
                superior( AgtAcordado , _ , 0 )
            ) ,
%       encontra o agente verbal `a sua direita
            encontraAgtVerbal( AgtAcordado , AgtVerbal ) ,

            (   /* encontrou um agente verbal `a sua direita */
                AgtVerbal \= 0 ,

%           pega o agente subordinado (nominal) para, no "establacos",
%           comparar, em caso de o AgtVerbal ja' possui-lo como subordinado
                subordinado( AgtAcordado , _ , AgtSub ) ,

%           pega o primeiro agt nominal `a sua direita para fazer uma
%           "sugestao" de posicao sintatica que seu subordinado deve ocupar
                procuraVizinho( AgtAcordado , maior , nominal , AgtNomDir ) ,
                sugestaoPosSint( AgtNomDir , AgtVerbal , SugestPosSint ) ,

%           pede estabelecimento de lacos
                estabLacos_PrRel_Verb( AgtAcordado , AgtVerbal , AgtSub ,
                            FuncSint1 , SugestPosSint , Ok2 ) ,
```

```
                    (   /* estabelecimento de lacos aceito */
                        Ok2 ,
                        retractall( superior( AgtAcordado , _ , _ ) ) ,
                        assert( superior( AgtAcordado , FuncSint1 , AgtVerbal ) ) ,
                        fezAlgo( AgtAcordado )

                    ;   /* estabelecimento de lacos nao aceito */
                        not( Ok2 ) ,
                        write( 'ERRO: acorda pron. rel. (' ) ,
                        write( AgtAcordado ) ,
                        write( ') / segundo estab. lacos verbal nao aceito' ) , nl ,
                        write( ' pelo agente verbal ' ) , write( AgtVerbal ) ,
                        nl
                    )

                ;   /* nao encontrou um agente verbal `a sua direita */
                    AgtVerbal = 0 , true
%           write( 'ERRO: acorda pron. rel. (' ) ,
%           write( AgtAcordado ) , write( ') / nao existe agente verbal' ) ,
%           nl
                )

    ;   /* se possuir um superior e um subordinado ... */
            (   not( superior( AgtAcordado , _ , 0 ) ) ,
                not( subordinado( AgtAcordado , _ , 0 ) )
            ) ,
            /* ... entao trata as mensagens de termino */
            tratamentoDeMsgTermino( AgtAcordado )
    ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
            agtFezAlgo( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
            not( agtFezAlgo( AgtAcordado ) ) ,
            naoFezNada( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ) ,
    !

% fim acorda( pron_rel )
  .


% ------------------------------------------------------------------
% EncontraAgtVerbal( AgtPronRel , AgtVerbal ) :- esse predicado tem
%       por funcao retornar o AgtVerbal com o AgtPronRel faz "par".
%       A ideia e' encontrar um vizinho verbal `a direita e verificar
%       se nao existe um outro agente pronome relativo entre eles.
%       Se nao existir, o vizinho verbal encontrado e' o que se pro-
%       cura. Se existir, entao continua a busca para a direita, pas-
%       sando o vizinho verbal encontrado como o ponto de inicio da
%       nova busca. Isso resolve os casos de aninhamento de pronome
%       relativo simples (sem coordenacao).

  encontraAgtVerbal( AgtPronRel , AgtVerbal ) :-

    procuraVizinho( AgtPronRel , maior , verbal , VizVerbal ) ,
    procuraVizinho( AgtPronRel , maior , pron_rel , VizPronRel ) ,

    posicao( PosVizVerbal , VizVerbal ) ,
    posicao( PosVizPronRel , VizPronRel ) ,

    (   /*  vai prosseguir na busca */
            (   VizVerbal \= 0 , VizPronRel \= 0 ,
                PosVizPronRel < PosVizVerbal
            ) ,
            encontraAgtVerbal( VizVerbal , AgtVerbal )

            /*  vai retornar o primeiro agente verbal encontrado */
    ;   (   VizVerbal \= 0 , VizPronRel \= 0 ,
                PosVizPronRel > PosVizVerbal
            ) ,
            iguala( AgtVerbal , VizVerbal )

            /* nao encontrou e vai retornar 0 */
    ;   iguala( AgtVerbal , VizVerbal )
    ) ,
    !

% fim encontraAgtVerbal
  .

% ------------------------------------------------------------------
  sugestaoPosSint( AgtNomDir , AgtVerbal , SugestPosSint ) :-

%   pega as posicoes do AgtNomDir e do AgtVerbal para comparar
    posicao( PosAgtNomDir , AgtNomDir ) ,
    posicao( PosAgtVerbal , AgtVerbal ) ,
    conferePosicionamento( RelPos , PosAgtNomDir , PosAgtVerbal ) ,
    (   AgtNomDir \= 0 ,
            (   RelPos = maior ,
                iguala( SugestPosSint , obj )
            ;   RelPos = menor ,
                iguala( SugestPosSint , suj )
            )
    ;   AgtNomDir = 0 ,
            iguala( SugestPosSint , suj )
    ) ,
```

```
    !

% fim sugestaoPosSint
  .



% ------------------------------
% Acorda para agente DETERMINANTE

  acorda( AgtAcordado , determinante ) :-

%       nao possui superior
    (   superior( AgtAcordado , _ , 0 ) ,

%       encontra o agente nominal `a sua direita
            procuraVizinho( AgtAcordado , maior , nominal , AgtNom ) ,

            (   AgtNom \= 0 ,
%           pede estabelecimento de lacos
                estabLacos_Det_Nom( AgtAcordado , AgtNom , _ , Ok ) ,

                (   Ok ,
                 retractall( superior( AgtAcordado , _ , _ ) ) ,
                 assert( superior( AgtAcordado , det , AgtNom ) ) ,
                 fezAlgo( AgtAcordado )

                ;   not( Ok ) ,
                 write( 'ERRO: acorda determinante (' ) ,
                 write( AgtAcordado ) ,
                 write( ') / estab. lacos nao aceito' ) ,
                 nl
                )

                 /* nao existe agente nominal `a direita do agente determinante
*/
            ;   AgtNom = 0 , true
%           write( 'ERRO: acorda determinante (' ) ,
%           write( AgtAcordado ) ,
%           write( ') / nao existe agente nominal' ) ,
%           nl
            )
    ;   /* se possuir um superior ... */
            not( superior( AgtAcordado , _ , 0 ) ) ,
            /* ... entao trata as mensagens de termino */
            tratamentoDeMsgTermino( AgtAcordado )
    ) ,

    (   /* se fez algo, apenas apaga a ocorrencia */
            agtFezAlgo( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ;   /* se nao fezAlgo( AgtAcordado ) entao incrementa a listaNaoFezNada */
            not( agtFezAlgo( AgtAcordado ) ) ,
            naoFezNada( AgtAcordado ) ,
            retractall( agtFezAlgo( AgtAcordado ) )
    ) ,

    !

% fim acorda( determinante )
  .

% ------------------------------------------------------------------------
%    PREDICADOS DE VERIFICACAO DO FUNCIONAMENTO DOS AGENTES
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% FezAlgo( AgtAcordado ) :- esse predicado sinaliza que o agtAcordado
%       realizou alguma funcao e tambem esvazia a lista dos que nao
%       fizeram nada

  fezAlgo( AgtAcordado ) :-

    assert( agtFezAlgo( AgtAcordado ) ) ,
    retractall( listaNaoFezNada( _ ) ) ,
    assert( listaNaoFezNada( [] ) ) ,
    !

% fim fezAlgo
  .



% ------------------------------------------------------------------------
% NaoFezNada( AgtAcordado ) :- esse predicado sinaliza que o agtAcordado
%       nao realizou funcao alguma enquanto foi acordado. Ele verifica
%       se o agtAcordado nao esta' na listaNaoFezNada para inseri'-lo
%       caso nao esteja.

  naoFezNada( AgtAcordado ) :-

    listaNaoFezNada( ListaNFN ) ,
    (   not( inList( AgtAcordado , ListaNFN ) ) ,
            insert( AgtAcordado , ListaNFN , ListaNFN2 ) ,
            retractall( listaNaoFezNada( ListaNFN ) ) ,
            assert( listaNaoFezNada( ListaNFN2 ) )
    ;   /* se o agente ja' esta' na lista, nao faz nada */
            true
    ) ,
    !
```

```
% fim naoFezNada
  .

% ------------------------------------------------------------------------
%    PREDICADOS DE TRATAMENTO DE MENSAGEM DE TERMINO
% ------------------------------------------------------------------------


% ------------------------------------------------------------------------
% TratamentoDeMsgTermino :- todo agente tem que verificar se alguem ja'
%        sinalizou com uma mensagem de termino. Caso ja' , ele deve re-
%        tirar seu nome da lista dos que nao viram a mensagem , sinali-
%        zando , assim , que a viu.


  tratamentoDeMsgTermino( AgtAcordado ) :-

%         caso nao haja mensagem de termino
      (   not( terminei( _ , _ ) ) ,

%         nao faz nada
                true

    ;     /* caso haja alguma mensagem de termino */
                foreach( terminei( AgtTermino , ListaTermino ) ,

                     (   /*  se o agente ainda nao viu esta mensagem ... */
                         (   AgtAcordado \= AgtTermino ,
                          not( inList( AgtAcordado , ListaTermino ) )
                         ) ,
%                        ... sinaliza que agora o AgtAcordado viu a mensagem ,
%                        inserindo-o na lista de termino
                         insert( AgtAcordado , ListaTermino , NovaListaTermino ) ,
%                        pega o tamanho dessa nova lista e ...
                         length( NovaListaTermino , TamanhoNLT ) ,
%                        ... o numero de agentes ...
                         numAgts( NumAgts ) ,

                         (   /* ... para verificar se sao iguais */
                          TamanhoNLT = NumAgts ,
                          retract( todosViramTermino( fail ) ) ,
                          assert( todosViramTermino( true ) )

                         ;    retract( terminei( AgtTermino , ListaTermino ) ) ,
                          assert( terminei( AgtTermino , NovaListaTermino ) )
                         )

                     ;   /* se o agente acordado ja' tinha visto a msg de termino,
                         apenas continua */
                                true
                         )
                        ) /* fim do foreach */
      )

% fim tratamentoDeMsgTermino
  .



% ------------------------------------------------------------------------
%    PREDICADOS DE ESTABELECIMENTO DE LACOS .
% ------------------------------------------------------------------------


% +-----------------------------------------+
% | pedido por um AG. VERBAL para um AG. NOMINAL |
% +-----------------------------------------+

  estabLacos_Verb_Nom( AgtVerbal , AgtNominal , suj , Ok ) :-

     (   /* o agente nominal nao possui um superior */
                superior( AgtNominal , _ , 0 ) ,

                /* registra que agora ele possui um superior */
                retractall( superior( AgtNominal , _ , _ ) ) ,
                assert( superior( AgtNominal , suj , AgtVerbal ) ) ,

                iguala( Ok , true )

     ;  /* o agente nominal ja' possui um superior */
                iguala( Ok , fail )
     )

% fim estabLacos_Verb_Nom
  .

  estabLacos_Verb_Nom( AgtVerbal , AgtNominal , obj , Ok ) :-

     (   /* o agente nominal nao possui um superior */
                superior( AgtNominal , obj , 0 ) ,

                /* registra que agora ele possui um superior */
                retractall( superior( AgtNominal , obj , _ ) ) ,
                assert( superior( AgtNominal , obj , AgtVerbal ) ) ,

                iguala( Ok , true )

     ;  /* o agente nominal ja' possui um superior */
                iguala( Ok , fail )
```

```
    )

% fim estabLacos_Verb_Nom
  .


  estabLacos_Verb_Nom( AgtVerbal , AgtNominal , obj_ind , Ok ) :-

    (   /* o agente nominal nao possui um superior */
            superior( AgtNominal , obj_ind , 0 ) ,

            /* registra que agora ele possui um superior */
            retractall( superior( AgtNominal , obj_ind , _ ) ) ,
            assert( superior( AgtNominal , obj_ind , AgtVerbal ) ) ,

            iguala( Ok , true )

    ;   /* o agente nominal ja' possui um superior */
            iguala( Ok , fail )
    )

% fim estabLacos_Verb_Nom
  .

% +-------------------------------------------------+
% | pedido por um AG. DETERMINANTE para um AG. NOMINAL |
% +-------------------------------------------------+

  estabLacos_Det_Nom( AgtDeterm , AgtNominal , det , Ok ) :-

    (   /* o agente nominal nao possui um subordinado */
            subordinado( AgtNominal , det , 0 ) ,

            /* registra que agora ele possui um subordinado */
            retractall( subordinado( AgtNominal , det , _ ) ) ,
            assert( subordinado( AgtNominal , det , AgtDeterm ) ) ,

            iguala( Ok , true )

    ;   /* o agente nominal ja' possui um subordinado */
            iguala( Ok , fail )
    )

% fim estabLacos_Det_Nom
  .

% +-------------------------------------------------+
% | pedido por um AG. PRONOME RELATIVO para um AG. NOMINAL |
% +-------------------------------------------------+

  estabLacos_PrRel_Nom( AgtPrRel , AgtNom , Ok ) :-

    assert( superior_que( AgtNom , AgtPrRel ) ) ,
    iguala( Ok , true )

% fim estabLacos_PrRel_Nom
  .


% +-------------------------------------------------+
% | pedido por um AG. PRONOME RELATIVO para um AG. VERBAL |
% +-------------------------------------------------+

  estabLacos_PrRel_Verb( AgtPrRel , AgtVerb , AgtNom ,
                         FuncSint , SugestFuncSint , Ok ) :-

%       verifica se o AgtNom ja' e' subordinado do AgtVerb
    (   subordinado( AgtVerb , TipSint , AgtNom ) ,
%       o AgtVerb pede para o AgtNom apagar suas ligacoes com ele
            quebraLaco_Verb_Nom( AgtVerb , AgtNom ) ,
%       o proprio AgtVerb apaga suas ligacoes do tipo "TipSint" com o AgtNom
            retractall( subordinado( AgtVerb , TipSint , AgtNom ) ) ,
%       faz as ligacoes com o agente pronome relativo
            assert( subordinado( AgtVerb , TipSint , AgtPrRel ) ) ,
            assert( subordinado_que( AgtVerb ) ) ,

            iguala( FuncSint , TipSint ) ,
            iguala( Ok , true )

    ;   not( subordinado( AgtVerb , _ , AgtNom ) ) ,

            (   /* o agente verbal ainda nao estava saturado */
                pendencia( AgtVerb , SugestFuncSint ) ,
                retractall( subordinado( AgtVerb , SugestFuncSint , 0 ) ) ,
                assert( subordinado( AgtVerb , SugestFuncSint , AgtPrRel ) ) ,
                assert( subordinado_que( AgtVerb ) ) ,
                retractall( pendencia( AgtVerb , SugestFuncSint ) ) ,
                iguala( FuncSint , SugestFuncSint ) ,
                iguala( Ok , true )

            ;   /* o agente verbal ja' estava saturado */
%           pega o agente que ja' estava ligado na posicao que se quer
                subordinado( AgtVerb , SugestFuncSint , SubAgtVerb ) ,
%           o subordinado quebra seus lacos com o agente verbal
                quebraLaco_Verb_Nom( AgtVerb , SubAgtVerb ) ,
%           o verbal apaga seus lacos com seu antigo subordinado
                retractall( subordinado( AgtVerb , SugestFuncSint , SubAgtVerb )
) ,
                assert( subordinado( AgtVerb , SugestFuncSint , AgtPrRel ) ) ,
                assert( subordinado_que( AgtVerb ) ) ,
```

```
                igual( FuncSint , SugestFuncSint ) ,
                igual( Ok , true )

            )
    ) .

%   esse "flag" sera' utilizado para um agente
%   verbal poder mudar seu comportamento no "acorda"
    assert( flag_que ) .
    !

% fim estabLacos_PrRel_Verb
  .


% +-----------------------------------------------+
% | pedido por um AG. COORDENACAO para um AG. NOMINAL |
% +-----------------------------------------------+

% A principio, este predicado deveria estar separado em dois,
% visto que deveria ser enviado para dois agentes diferentes,
% mas, para facilitar,


  estabLacos_Coord_Nom( AgtCoord , AgtNomEsq , AgtNomDir , Ok ) :-

    (   /* os agentes nominais nao possuem superiores */
            superior( AgtNomEsq , _ , 0 ) ,
            superior( AgtNomDir , _ , 0 ) ,

%       pega as informacoes de numero e pessoa e de significado para
%       ver como elas podem "unificar". Se nao houver problemas, essa
%       informacao unificada sera' inserida no novo agente

            agente( AgtNomEsq , _ , _ , InfoAgtEsq , _ ) ,
            agente( AgtNomDir , _ , _ , InfoAgtDir , _ ) ,

            unificaNominais( InfoAgtEsq , InfoAgtDir , InfoUnificada ) ,

            agente( AgtCoord , X , Y , Z , W ) ,
            retractall( agente( AgtCoord , X , Y , Z , W ) ) ,
            assert( agente( AgtCoord , X , Y , InfoUnificada , W ) ) ,

            retractall( superior( AgtNomEsq , _ , 0 ) ) ,
            assert( superior( AgtNomEsq , coord , AgtCoord ) ) ,
            retractall( superior( AgtNomDir , _ , 0 ) ) ,
            assert( superior( AgtNomDir , coord , AgtCoord ) ) ,

            igual( Ok , true )

    ;   /* o agente nominal ja' possui superior */
            (   not( superior( AgtNomEsq , _ , 0 ) )
            ;   not( superior( AgtNomDir , _ , 0 ) )
            ) ,
            igual( Ok , fail )
    ) .

    !

% fim estabLacos_Coord_Nom
  .


% ---------------
  unificaNominais( InfoAgtEsq , InfoAgtDir , InfoUnificada ) :-

%   aqui teria quer ser feito um mapeamento, mas nao irei me preocupar
%   muito com isso porque, nos exemplos/testes, isso nao e' necessario.
%
%   Exemplo de um mapeamento:
%   eu[1sing] e voce[3sing] ===> nos[1plur]

%   mapeamento de genero
    retornaValorAtributo( gen , InfoAgtDir , GenDir ) ,
    retornaValorAtributo( gen , InfoAgtEsq , GenEsq ) ,
    (   (   GenDir = masc
            ;   GenEsq = masc
            ) ,
            igual( GenUnif , masc )
    ;   igual( GenUnif , fem )
    ) .

    retornaValorAtributo( significado , InfoAgtDir , SigDir ) ,
    retornaValorAtributo( significado , InfoAgtEsq , SigEsq ) ,

% => Atencao!
%   O predicado abaixo "unificaSignificado" nao foi implementado!
%   unificaSignificado( SigEsq , SigDir , SigUnif ) ,
%
%   para os exemplos das frases de teste, as coordenacoes estao
%   sendo feitas apenas com objetos semanticos do tipo "humano"
%   que esta' sendo inserido diretamente no campo "significado"
%   da "InfoUnificada"
%
%   a mesma coisa foi feita com relacao ao campo "pessoa" que
%   nas coordenacoes das frases exemplo sempre serao "terceira pessoa"

    igual( InfoUnificada , [ pessoa=[3] ,
                             num=[plur] ) ,
```

```
                        gen={GenUnif} ,
                        significado={[ ([ tip_sem={humano} ]} ]]
                       ] ) .


    !

% fim unificaNominais
  .


% +-----------------------------------------------+
% | pedido por um AG. COORDENACAO para um AG. VERBAL |
% +-----------------------------------------------+

  estabLacos_Coord_Verb( AgtCoord , AgtVerbEsq , AgtVerbDir , Ok ) :-

    (   /* o agente verbal nao possui superior */
            superior( AgtVerbEsq , _ , 0 ) ,
            superior( AgtVerbDir , _ , 0 ) ,
            retractall( superior( AgtVerbEsq , _ , 0 ) ) ,
            retractall( superior( AgtVerbDir , _ , 0 ) ) ,
            assert( superior( AgtVerbEsq , coord , AgtCoord ) ) ,
            assert( superior( AgtVerbDir , coord , AgtCoord ) ) ,
            iguala( Ok , true )

    ;   /* o agente verbal ja' possui superior */
            (   not( superior( AgtVerbEsq , _ , 0 ) )
            ;   not( superior( AgtVerbDir , _ , 0 ) )
            ) ,
            iguala( Ok , fail )
    ) ,

    !

% fim estabLacos_Coord_Verb
  .


% +-----------------------------------------------+
% | pedido por um AG. PREPOSICAO para um AG. NOMINAL |
% +-----------------------------------------------+

% ---------------------------------------------------------------------
% estabLacos_1_Prep_Nom( AgtPrep , RestrSemAP , AgtNom , RestrSemAN , Ok ) :-
%       esse predicado, alem de fazer o estabelecimento de lacos do agente
%       preposicional com o agente nominal, "casa" as restricoes semanticas
%       que o agente preposicional tem para complemento com as possibilida-
%       des do agente nominal e retorna em "RestrSemAN" (AN de agente nomi-
%       nal), pois o agente nominal e' que tem o papel semantico principal

  estabLacos_1_Prep_Nom( AgtPrep , ListaRestrSemAP ,
                         AgtNom , RestrSemAN , Ok ) :-


%   pega as restricoes semanticas do agente nominal
    agente( AgtNom , _ , _ , InfoDic , _Signif ) ,
    retornaValorAtributo( significado , InfoDic , Significados ) ,
    retornaTodosValores( tip_sem , Significados , ListaRestrSemAN ) ,

%   verifica a que melhor se adequa com as do agente preposicional
    verificaAdequacaoSem( ListaRestrSemAP , ListaRestrSemAN , RestrSem ) ,

    (   % se eles forem "compativeis", entao estabelece os lacos
            not( iguala( RestrSem , [] ) ) ,
            retractall( superior( AgtNom , _ , _ ) ) ,
            assert( superior( AgtNom , prep , AgtPrep ) ) ,

%       instancia a restricao semantica do agente nominal para ser retornada
            iguala( RestrSem , RestrSemAN ) ,

            iguala( Ok , true )

    ;   % senao atesta a incompatibilidade dos dois agentes
            iguala( RestrSem , [] ) ,
            iguala( Ok , fail )
    ) ,

    !

% fim estabLacos_1_Prep_Nom
  .


% +-----------------------------------------------+
% | pedido por um AG. PREPOSICAO para um AG. NOMINAL |
% +-----------------------------------------------+

% ---------------------------------------------------------------------
  estabLacos_2_Prep_Nom( AgtPrep , AgtNom , Ok ) :-


    assert( subordinado( AgtNom , prep , AgtPrep ) ) ,
    iguala( Ok , true ) ,

    !

% fim estabLacos_2_Prep_Nom
  .

% +-----------------------------------------------+
```

```
% | pedido por um AG. PREPOSICAO para um AG. VERBAL |
% +-----------------------------------------------+

% ---------------------------------------------------------------------
  estabLacos_Prep_Verb( AgtPrep , PalavraPrep , RestrSemAP ,
                        AgtVerb , RestrSemRetornada , Ok ) :-

%    o agente verbal verifica se ele possui uma opcao de objeto indireto
     agente( AgtVerb , _ , _ , InfoDic , _ ) ,
     retornaValorAtributo( significado , InfoDic , ListaSignif ) ,
     retornaTodosValores( opcion , ListaSignif, [ListaOpcion] ) ,
     retornaValorAtributo( obj_ind, ListaOpcion, ListaObjInd ) ,


%    pega os valor semantico associado `a preposicao "PalavraPrep"
%    nos dados do agente verbal. Isto e', se o agente possui, nas
%    informacoes do dicionario, um atributo preposicao ("prep") com
%    valor igual a "PalavraChave" (ex.: 'de', 'com', 'para', etc.),
%    entao insere o valor do atributo "tip_sem" em uma lista que
%    sera' retornada como "ListaRestrSemVerb"
     retornaAtributoDadoChave( PalavraPrep , prep , tip_sem ,
                               ListaObjInd , ListaRestrSemVerb ) ,

     verificaAdequacaoSem( ListaRestrSemVerb , RestrSemAP , Melhor ) ,

     (   Melhor \= [] ,
             retractall( subordinado( AgtVerb , _ , 0 ) ) ,
             assert( subordinado( AgtVerb , obj_ind , AgtPrep ) ) ,

%        retorna o melhor tipo semantico que foi encontrado/"casado"
             iguala( RestrSemRetornada , Melhor ) ,
             iguala( Ok , true )

     ;   Melhor = [] ,
             iguala( Ok , fail )
     ) ,

     !

% fim estabLacos_Prep_Verb
  .



% ---------------------------------------------------------------------
%    PREDICADOS DE QUEBRA DE LACOS
% ---------------------------------------------------------------------

% ---------------------------------------------------------------------
% QuebraLaco_Verb_Nom( AgtVerb , AgtNom ) :- o AgtVerb pede para o
%        AgtNom apagar suas ligacoes com ele

  quebraLaco_Verb_Nom( AgtVerb , AgtNom ) :-

     retractall( superior( AgtNom , _ , AgtVerb ) ) ,

%    como e' o agente verbal que esta' pedindo a quebra de lacos,
%    e o agente nominal so' pode ter um superior, entao sinaliza
%    que nao ha' mais superior para esse agente nominal
     assert( superior( AgtNom , _ , 0 ) )

% fim quebraLaco_Verb_Nom
  .

% ---------------------------------------------------------------------
% PedidoQuebraLacos( Ag1 , Ag2 ) :- pedido para o Ag1 quebrar seus lacos
%        com o Ag2, resultando em o Ag1 nao possuir o Ag2 como subordina-
%        do. O Ag1 ira' entao lancar mensagem "preciso" (flexibilizacao)
%        para tentar obter um agente para substituir a posicao sintatica
%        ocupada pelo Ag2.

  pedidoQuebraLacos( Ag1 , Ag2 ) :-

%    pega a funcao sintatica entre o Ag2 e o Ag1
     subordinado( Ag1 , FuncSint , Ag2 ) ,
%    pede ao Ag2 para apagar suas ligacoes com o Ag1
     quebraLaco_Verb_Nom( Ag1 , Ag2 ) ,
     retractall( subordinado( Ag1 , _ , Ag2 ) ) ,

%        verifica o tipo de funcao sintatica (suj, objb, obj_ind)
%        para pedir sua flexibilizacao
     (   iguala( FuncSint , suj ) ,
             flexibilizaSuj( Ag1 )
     ;   iguala( FuncSint , obj ) ,
             flexibilizaObj( Ag1 )
     ;   iguala( FuncSint , obj_ind ) ,
             flexibilizaObjInd( Ag1 )
     ) ,

     (   /* se NAO havia uma mensagem de termino do Ag1 */
             not( terminei( Ag1 , _ ) ) ,
             /* continua */
             true
     ;   /* se havia */
             terminei( Ag1 , _ ) ,
             /* retira a mensagem da memoria */
             retractall( terminei( Ag1 , _ ) )
     )

% fim pedidoQuebraLacos
```

```
% ------------------------------------------------------------------
%    PREDICADOS DE MAPEAMENTO/ENVIO DOS "PRECISO"s
% ------------------------------------------------------------------


% ------------------------------------------------------------------
% mapeamentoNecessidades( IdAg , Signif , Info ) :- de acordo com os
%       "Signif"icados e as "Info" do verbo em questao, insere mensa-
%       gens "Preciso" na memoria global.


  mapeamentoNecessidades( IdAg , [] , _ ) .

  mapeamentoNecessidades( IdAg , [ISignif|RestSignif] , Info ) :-

%    pega o primeiro significado e obtem os valores obrigatorios
     retornaValorAtributo( obrig , Signif , ValoresObrig ) ,

%    pega os valores para concordancia com o sujeito (pessoa e numero) e ...
     retornaValorAtributo( pessoa , Info , ValorPessoa ) ,
     retornaValorAtributo( num , Info , ValorNum ) ,
%    ...pega os valores para "sujeito" no primeiro significado e ...
     retornaValorAtributo( suj , ValoresObrig , ValorSuj ) ,
%    ... os mapeia
     mapeaSuj( IdAg , menor , ValorSuj , ValorPessoa , ValorNum ) ,

%    faz o mesmo para "objeto direto"
     retornaValorAtributo( obj , ValoresObrig , ValorObj ) ,
     mapeaObj( IdAg , maior , ValorObj ) ,

%    faz o mesmo para "objeto indireto"
     retornaValorAtributo( obj_ind , ValoresObrig , ValorObjInd ) ,
     mapeaObjInd( IdAg , maior , ValorObjInd ) ,

%    chamada recursiva para mapear o restante dos significados
     mapeamentoNecessidades( IdAg , RestSignif , Info )

% fim mapeamentoNecessidades
  .


% ------------------------------------------------------------------
% mapeaSuj( IdAg , Local , Info , Pessoa , Num ) :- de acordo com "Info",
%       "Pessoa" e "Num", insere "Precisos" na memoria global relativos
%       aos possiveis "sujeitos" de um verbo


  mapeaSuj( IdAg , Local , [] , _ , _ ) :- ! .

  mapeaSuj( IdAg , Local , [] , _ , _ ) :- ! .

  mapeaSuj( IdAg , Local , [[X]|Xs] , Pessoa , Num ) :-

     retornaValorAtributo( tip_sem , X , TipoSem ) ,
     assert( mem([ preciso=[[ local=[Local] ,
                              emissor=[IdAg] ,
                              tip_sint=[suj] ,
                              pessoa=[Pessoa] ,
                              num=[Num] ,
                              tip_sem=[TipoSem] ]] ,
                   lista=[[]] ]) ) ,

%    sinaliza que ha' uma pendencia de sujeito para esse agente
     assert( pendencia( IdAg , suj ) ) ,

     mapeaSuj( IdAg , Local , Xs , Pessoa , Num )

% fim mapeaSuj
  .


% ------------------------------------------------------------------
% mapeaObj( IdAg , Local , Info ) :- de acordo com "Info" , insere "Precisos" na
%       memoria global relativos aos possiveis "objetos" de um verbo


  mapeaObj( IdAg , Local , [] ) :- ! .

  mapeaObj( IdAg , Local , [[]] ) :- ! .

  mapeaObj( IdAg , Local , [[X]|Xs] ) :-

     retornaValorAtributo( tip_sem , X , TipoSem ) ,
     assert( mem([ preciso=[[ local=[Local] ,
                              emissor=[IdAg] ,
                              tip_sint=[obj] ,
                              tip_sem=[TipoSem] ]] ,
                   lista=[[]] ]) ) ,

%    sinaliza que ha' uma pendencia de objeto direto para esse agente
     assert( pendencia( IdAg , obj ) ) ,

     mapeaObj( IdAg , Local , Xs )

% fim mapeaObj
  .


% ------------------------------------------------------------------
% mapeaObjInd( IdAg , Local , Info ) :- de acordo com "Info" , insere
%       "Precisos" na memoria global relativos aos possiveis "objetos
```

```
%       indiretos" do verbo


  mapeaObjInd( IdAg , Local , [] ) :- ! .

  mapeaObjInd( IdAg , Local , [[]] ) :- ! .

  mapeaObjInd( IdAg , Local , [[X]|Xs] ) :-

     retornaValorAtributo( tip_sem , X , TipoSem ) ,
     retornaValorAtributo( prep , X , Prep ) ,
     assert( mem([ preciso=[[ local=[Local] ,
                              emissor=[IdAg] ,
                              tip_sint=[obj_ind] ,
                              tip_sem=[TipoSem] ,
                              prep=[Prep] ]] ,
                   lista=[[]] ]) ) ,

%    sinaliza que ha' uma pendencia de objeto indireto para esse agente
     assert( pendencia( IdAg , obj_ind ) ) ,

     mapeaObjInd( IdAg , Local , Xs )

% fim mapeaObjInd
  .



% ------------------------------------------------------------------------
%    PREDICADOS DE FLEXIBILIZACAO DE "PRECISO"s PARA POSICOES SINTATICAS
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% flexibilizaSuj( IdAg ) :- pega as informacoe do agente verbal "IdAg"
%        e insere "Precisos" na memoria global relativos aos possiveis
%        "sujeitos" de um verbo com POSICAO flexibilizada


% ------- flexibilizaSuj /1 --------

  flexibilizaSuj( IdAg ) :-

%    pega as informacoes do agente verbal
     agente( IdAg , verbal , _ , InfoDic , _ ) ,

%    pega os possiveis significados
     retornaValorAtributo( significado , InfoDic , Significados ) ,

%    de acordo com os significados, ira' inserir "precisos" na memoria
     flexibilizaSuj( IdAg , Significados , InfoDic )
  .

% ------- flexibilizaSuj /3 --------

  flexibilizaSuj( IdAg , [] , _ ) :- ! .

  flexibilizaSuj( IdAg , [[Signif]|RestSignif] , InfoDic ) :-

%    pega os valores obrigatorios do primeiro significado
     retornaValorAtributo( obrig , Signif , ValoresObrig ) ,

%    pega os valores para concordancia com o sujeito (pessoa e numero) e ...
     retornaValorAtributo( pessoa , InfoDic , ValorPessoa ) ,
     retornaValorAtributo( num , InfoDic , ValorNum ) ,
%    ...pega os valores para "sujeito" no primeiro significado e ...
     retornaValorAtributo( suj , ValoresObrig , ValorSuj ) ,
%    ... os mapeia

%    A FLEXIBILIZACAO ESTA' NO VALOR "QUALQUER"
     mapeaSuj( IdAg , qualquer , ValorSuj , ValorPessoa , ValorNum )


% fim flexibilizaSuj /3
  .

% ------------------------------------------------------------------------
% flexibilizaObj( IdAg ) :- pega as informacoe do agente verbal "IdAg"
%        e insere "Precisos" na memoria global relativos aos possiveis
%        "objetos" de um verbo com POSICAO flexibilizada


% ------- flexibilizaObj /1 --------

  flexibilizaObj( IdAg ) :-

%    pega as informacoes do agente verbal
     agente( IdAg , verbal , _ , InfoDic , _ ) ,

%    pega os possiveis significados
     retornaValorAtributo( significado , InfoDic , Significados ) ,

%    de acordo com os significados, ira' inserir "precisos" na memoria
     flexibilizaObj( IdAg , Significados )

  .

% ------- flexibilizaObj /2 --------

  flexibilizaObj( IdAg , [] ) :- ! .
```

```
  flexibilizaObj( IdAg , [(Signif)|RestSignif] ) :-

%    pega os valores obrigatorios do primeiro significado
     retornaValorAtributo( obrig , Signif , ValoresObrig ) ,

%    ...pega os valores para "objeto" no primeiro significado e ...
     retornaValorAtributo( obj , ValoresObrig , ValorObj ) ,
%    ... os mapeia

%    A FLEXIBILIZACAO ESTA' NO VALOR "QUALQUER"
     mapeaObj( IdAg , qualquer , ValorObj )


% fim flexibilizaObj /2
  .


% ------------------------------------------------------------------
% flexibilizaObjInd( IdAg ) :- pega as informacoe do agente verbal "IdAg"
%       e insere "Precisos" na memoria global relativos aos possiveis
%       "objetos indiretos" de um verbo com POSICAO flexibilizada


% ------- flexibilizaObjInd /1 --------

  flexibilizaObjInd( IdAg ) :-

%    pega as informacoes do agente verbal
     agente( IdAg , verbal , _ , InfoDic , _ ) ,

%    pega os possiveis significados
     retornaValorAtributo( significado , InfoDic , Significados ) ,

%    de acordo com os significados, ira' inserir "precisos" na memoria
     flexibilizaObjInd( IdAg , Significados )

  .

% ------- flexibilizaObjInd /2 --------

  flexibilizaObjInd( IdAg , [] ) :- ! .

  flexibilizaObjInd( IdAg , [(Signif)|RestSignif] ) :-

%    pega os valores obrigatorios do primeiro significado
     retornaValorAtributo( obrig , Signif , ValoresObrig ) ,

%    ...pega os valores para "objeto direto" no primeiro significado e ...
     retornaValorAtributo( obj_ind , ValoresObrig , ValorObjInd ) ,
%    ... os mapeia

%    A FLEXIBILIZACAO ESTA' NO VALOR "QUALQUER"
     mapeaObjInd( IdAg , qualquer , ValorObjInd )


% fim flexibilizaObjInd /2
  .



% ------------------------------------------------------------------
%    PREDICADOS DE TRATAMENTO DOS "PRECISO"s / ENVIO DE "VOTO"s
% ------------------------------------------------------------------


% ------------------------------------------------------------------
% TrataMsgsPrecisos: para cada mensagem que satisfaz as restricoes
%    de posicao (i.e., sintaticas) e as semanticas, o agente deve
%    enviar uma mensagem de "voto" com suas informacoes dizendo que
%    ele se candidata a preencher a posicao necessitada/precisada

% OBS.: e' nesse predicado que se faz a verificacao de quem
%       ja' leu um preciso (simulacao do 'broadcast')


% ----------------
  trataMsgsPrecisos( AgtAcordado , ListaTiposSemAA , I0 , In ) :-

%    servira' como um contador
     I1 is I0 + 1 ,

%    pega o tipo do agente acordado
     tipoAgente( AgtAcordado , TipoAgtAcordado ) ,

%    pega uma mensagem na memoria centralizada
     mem([ preciso=(ListaPreciso) , lista=(ListaDosQueViram) ]) ,

%    pega o tipo sintatico precisado
     retornaValorAtributo( tip_sint , ListaPreciso , TipSintPrec ) ,

     (   iguala( TipoAgtAcordado , prep ) ,
               iguala( TipSintPrec , obj_ind )

     ;   iguala( TipoAgtAcordado , nominal ) ,
               (  iguala( TipSintPrec , suj )
               ;  iguala( TipSintPrec , obj )
               )
     ;   iguala( TipoAgtAcordado , coordenacao ) ,
```

```
            (   iguala( TipSintPrec , suj )
            ;   iguala( TipSintPrec , obj )
            )
    ) ,

%   pega o identificador do agente emissor do "preciso"
    retornaValorAtributo( emissor , ListaPreciso , AgtPreciso ) ,

%   verifica se nao e' uma msg que o proprio AgtAcordado colocou
    not( AgtAcordado = AgtPreciso ) ,

%   verifica se o AgtAcordado ainda nao viu a msg
    not( inList( AgtAcordado , ListaDosQueViram ) ) ,

%   como o AgtAcordado ainda nao viu a msg, entao
%   insere-o na lista dos que ja' viram a mensagem ,
%   formando uma nova lista, indicando que agora ele viu a msg
    insert( AgtAcordado , ListaDosQueViram , NovaListaDQV ) ,

%   retira a mensagem para atualiza'-la com a nova lista dos que
%   nao a viram , se ela nao for vazia (como testado logo abaixo)
    retract( mem([ preciso=(ListaPreciso) , lista=(ListaDosQueViram) ]) ) ,

%   pega a lista de todos os agentes desse tipo
    listaAgentes( TipoAgtAcordado , ListaAgtTipo ) ,

%   pega o numero de elementos dessa lista
    length( ListaAgtTipo , TamanhoListaTipo ) ,
    length( NovaListaDQV , TamNovaLDQV ) ,

    (   /* se o tamanho da lista e' igual ao tamanho da lista de todos os
                agentes do tipo do agtAcordado, e' porque todos os agentes desse
                tipo viram essa mensagem. Entao ela nao precisa se recolocada na
                memoria */
            TamanhoListaTipo = TamNovaLDQV

    ;   TamanhoListaTipo \= TamNovaLDQV ,
            assert( mem([ preciso=(ListaPreciso) , lista=(NovaListaDQV) ]) )
    ) ,

%   verifica a localizacao
    retornaValorAtributo( local , ListaPreciso , RelLocalMsg ) ,

%   pega as posicoes dos dois agentes
    posicao( PosAgtAcordado , AgtAcordado ) ,
    posicao( PosAgtPreciso , AgtPreciso ) ,

%   confere se "RelLocalMsg" esta' de acordo com o posicionamento dos 2 agts
    conferePosicionamento( RelLocalMsg , PosAgtPreciso , PosAgtAcordado ) ,

%   se o tip_sint do preciso for "suj" entao
%   verifica as restricoes de concordancia
    (   iguala( TipSintPrec , suj ) ,
            /* pega os valores de pessoa e numero do "preciso" */
            retornaValorAtributo( pessoa , ListaPreciso , PessoaPrec ) ,
            retornaValorAtributo( num , ListaPreciso , NumPrec ) ,
            /* pega os valores de pessoa e numero do agtAcordado */
            agente( AgtAcordado , _ , _ , InfoDic , _ ) ,
            retornaValorAtributo( pessoa , InfoDic , PessoaAgt ) ,
            retornaValorAtributo( num , InfoDic , NumAgt ) ,
            (   (   iguala( PessoaAgt , PessoaPrec ) ,
                    iguala( NumAgt , NumPrec )
                ) ,
                true

            ;   fail
            )
    ;   /* nao ha' checagem de concordancia se nao for sujeito */
            TipSintPrec \= suj ,
            true
    ) ,

%   pega a informacao do tipo semantico do AgtPreciso
    retornaValorAtributo( tip_sem , ListaPreciso , RestrTipoSemAP ) ,

%   verifica qual tipo semantico do "agente acordado"(AA) se aproxima
%   melhor da restricao pedida pelo "agente preciso"(AP)
    melhorTipoSem( RestrTipoSemAP , ListaTiposSemAA , MelhorTipoSemAA ) ,

    (   /* so' havera' votacao, se passou pelo teste semantico */
            MelhorTipoSemAA \= [] ,

%       obtem o tipo sintatico no qual o agtAcordado vai votar
            retornaValorAtributo( tip_sint , ListaPreciso , Tip_Sint ) ,

%       coloca o voto na memoria
            assert( voto( AgtPreciso , Tip_Sint , MelhorTipoSemAA , AgtAcordado
) )
    ;   /* senao, continuara' pegando uma nova mensagem para ser tratada */
            true
    ) ,

%   chamada recursiva para tratar mais mensagens
    trataMsgsPrecisos( AgtAcordado , ListaTiposSemAA , I1 , In )

% fim trataMsgsPrecisos

% -------------------
```

```
%
  trataMsgsPrecisos( AgtAcordado , TipoAgtAcordado , I , I ) :- ! .



% ------------------------------------------------------------------
%    PREDICADOS DE TRATAMENTO DOS "VOTO"s
% ------------------------------------------------------------------


% ------------------------------------------------------------------
% ApuraVotos( AgtDest , TipoSint , MelhorCandidato ) :- retorna em "Melhor-
%       Candidato" o identificador do agente que fez a melhor proposta com
%       relacao `a posicao sintatica "TipoSint" para o "AgtDest" (que foi
%       o que pos os "preciso"s na memoria

  apuraVotos( AgtDest , TipoSint , MelhorCandidato ) :-

     (   setof(  IdCandidato,
                    TSem^voto( AgtDest , TipoSint , TSem  , IdCandidato ) ,
                    ListaCandidatos ) ,

               length( ListaCandidatos , NumCandidatos ) ,

               (   /* apenas um candidato */
                   NumCandidatos = 1 ,
                   [MelhorCandidato] = ListaCandidatos

               ;   /* ha' mais de um candidato */
                   NumCandidatos > 1 ,
                   posicao( PosAgtDest , AgtDest ) ,
                   melhorProposta( PosAgtDest , TipoSint ,
                            ListaCandidatos , MelhorCandidato )
               )

     ;   /* nao houve votos, pois "setof" falhou */
               MelhorCandidato = 0
     ) ,
     !

% fim apuraVotos
  .

% ------------------------------------------------------------------

  melhorProposta( PosAgtDest , TipoSint , ListaCandidatos , MelhorCandidato ) :-

      (   TipoSint = suj ,
              melhorPropSuj( PosAgtDest , ListaCandidatos , MelhorCandidato )
      ;   TipoSint = obj ,
              melhorPropObj( PosAgtDest , ListaCandidatos , MelhorCandidato )
      )

% fim melhorProposta /4
  .

% ---------
% Melhor Proposta para SUJEITO

  melhorPropSuj( PosAgtDest , [X,Y|[]] , MelhorCandidato ) :-

%    obtem posicao dos agentes X e Y que votaram
     posicao( PosX , X ) ,
     posicao( PosY , Y ) ,

%    o objetivo e' pegar o mais proximo pela esquerda
     (   PosX > PosY ,
               (   PosX > PosAgtDest ,
                   iguala( Y , MelhorCandidato )
               ;   PosX < PosAgtDest ,
                   iguala( X , MelhorCandidato )
               )

     ;   PosX < PosY ,
               (   PosY > PosAgtDest ,
                   iguala( X , MelhorCandidato )
               ;   PosY < PosAgtDest ,
                   iguala( Y , MelhorCandidato )
               )
     )

% fim melhorPropSuj
  .


  melhorPropSuj( PosAgtDest , [X,Y|Resto] , MelhorCandidato ) :-

%    obtem posicao dos agentes X e Y que votaram
     posicao( PosX , X ) ,
     posicao( PosY , Y ) ,

%    o objetivo e' pegar o mais proximo pela esquerda
     (   PosX > PosY ,
               (   PosX > PosAgtDest ,
                   melhorPropSuj( PosAgtDest , [Y|Resto] , MelhorCandidato )
               ;   PosX < PosAgtDest ,
                   melhorPropSuj( PosAgtDest , [X|Resto] , MelhorCandidato )
               )
```

```
    ;   PosX < PosY ,
            (   PosY > PosAgtDest ,
                melhorPropSuj( PosAgtDest , [X|Resto] , MelhorCandidato )
            ;   PosY < PosAgtDest ,
                melhorPropSuj( PosAgtDest , [Y|Resto] , MelhorCandidato )
            )
    )

% fim melhorPropSuj
  .

% ---------
% Melhor Proposta para OBJETO

  melhorPropObj( PosAgtDest , [X,Y|[]] , MelhorCandidato ) :-

%   obtem posicao dos agentes X e Y que votaram
    posicao( PosX , X ) ,
    posicao( PosY , Y ) ,

%   o objetivo e' pegar o mais proximo pela direita
    (   PosX > PosY ,
            (   PosY > PosAgtDest ,
                igual a( Y , MelhorCandidato )
            ;   PosY < PosAgtDest ,
                igual a( X , MelhorCandidato )
            )

    ;   PosX < PosY ,
            (   PosX > PosAgtDest ,
                igual a( X , MelhorCandidato )
            ;   PosX < PosAgtDest ,
                igual a( Y , MelhorCandidato )
            )
    )

% fim melhorPropObj
  .


  melhorPropObj( PosAgtDest , [X,Y|Resto] , MelhorCandidato ) :-

%   obtem posicao dos agentes X e Y que votaram
    posicao( PosX , X ) ,
    posicao( PosY , Y ) ,

%   o objetivo e' pegar o mais proximo pela direita
    (   PosX > PosY ,
            (   PosY > PosAgtDest ,
                melhorPropObj( PosAgtDest , [Y|Resto] , MelhorCandidato )
            ;   PosY < PosAgtDest ,
                melhorPropObj( PosAgtDest , [X|Resto] , MelhorCandidato )
            )

    ;   PosX < PosY ,
            (   PosX > PosAgtDest ,
                melhorPropObj( PosAgtDest , [X|Resto] , MelhorCandidato )
            ;   PosX < PosAgtDest ,
                melhorPropObj( PosAgtDest , [Y|Resto] , MelhorCandidato )
            )
    )

% fim melhorPropObj
  .



% ------------------------------------------------------------------
% VerificaAdequacaoSem( ListaRestrSemAP , ListaRestrSemAN , RestrSem ) :-
%       verifica qual tipo semantico do agente nominal se adequa melhor
%       aos tipos possiveis especificados no dicionario para o comple-
%       mento do agente preposicao.

%       Obs.: AP = agente preposicao, AN = agente nominal

  verificaAdequacaoSem( ListaRestrSemAP , ListaRestrSemAN , Melhor ) :-

    verificaAdequacaoSem( ListaRestrSemAP , ListaRestrSemAN , [] , Melhor ) .


% -------------------
  verificaAdequacaoSem( [] , ListaRestrSemAN , Melhor , Melhor ) .


% -------------------
  verificaAdequacaoSem( [TipSem1|Rest] , ListaRestrSemAN , M1 , Melhor ) :-

    melhorTipoSem( TipSem1 , ListaRestrSemAN , M2 ) ,
    (   not( igual a( M2 , [] ) ) ,
            not( inList( M2 , M1 ) ) ,
            insert( M2 , M1 , M3 )
    ;   (  igual a( M2 , [] )
            ;  inList( M2 , M1 )
            ) ,
            igual a( M1 , M3 )
    ) ,
    verificaAdequacaoSem( Rest , ListaRestrSemAN , M3 , Melhor )

% fim verificaAdequacaoSem
```

```
.

% ----------------------------------------------------------------------
% MelhorTipoSem( TipoSemDesejado , ListaTiposSem , Melhor ) :-
%       retorna em "Melhor" o tipo semantico da lista de tipos semanticos
%       "ListaTiposSem" que mais se aproxima do tipo semantico deseja-
%       do "TipoSemDesejado"

 melhorTipoSem( TipoSemDesejado , ListaTiposSem , Melhor ) :-

    length( ListaTiposSem , TamLTS ) ,
            /* ha' mais de uma possibilidade */
    (   TamLTS > 1 ,
%       verifica qual e' melhor
            melhorCandidatoSem( TipoSemDesejado, ListaTiposSem , Melhor )

            /* ha' apenas uma possibilidade */
    :   TamLTS = 1 ,
            (   (   /* operacao para retirar o tipo candidato da lista */
                 select( CandidatoUnico , ListaTiposSem , _ ) ,
                 /* teste semantico */
                 isa( CandidatoUnico , TipoSemDesejado )
                ) ,
                iguala( Melhor , CandidatoUnico )

            :   /* se nao passou no teste ISA */
                Melhor = []
            )
    )

% fim melhorTipoSem
 .

% ----------------
 melhorCandidatoSem( TipoSemDesejado , [X,Y|[]] , Melhor ) :-

                                      % Possibilidades em
                                      % uma hierarquia (arvore):
%   acha o melhor entre os dois que restaram       %
                                      % (chute)   TipoDesejado
%   verifica se os dois satisfazem o teste ISA     %         / \
    (   (   isa( X , TipoSemDesejado ) ,           %        0  ...
                isa( X , Y )                          %        / \
            ) ,                                       %       X   Y
            iguala( Melhor , X )                      %
                                      % (Y melhor)  TipoDesejado
    :   (   isa( Y , TipoSemDesejado ) ,           %         / \
                isa( Y , X )                          %          X  ...
            ) ,                                       %         / \
            iguala( Melhor , Y )                      %        Y  ...
    :   /* aqui e' um chute */                        %
            iguala( Melhor , X )                      % (X melhor)  TipoDesejado
    )                                                 %         / \
                                                      %        Y  ...
% fim melhorCandidatoSem                              %         / \
 .                                                    %        X  ...


 melhorCandidatoSem( TipoSemDesejado, [X,Y|Resto] , Melhor ) :-

%   verifica se os dois satisfazem o teste ISA .
    (   (   isa( X , TipoSemDesejado ) ,
                isa( X , Y )
            ) ,
            /* melhor e' X */
            melhorCandidatoSem( TipoSemDesejado , [X|Resto] , Melhor )

    :   (   isa( Y , TipoSemDesejado ) ,
                isa( Y , X )
            ) ,
            /* melhor e' Y */
            melhorCandidatoSem( TipoSemDesejado , [Y|Resto] , Melhor )

    :   /* aqui e' um "chute": no caso, esta' indo X */
            melhorCandidatoSem( TipoSemDesejado , [X|Resto] , Melhor )
    )

% fim melhorCandidatoSem
 .

% ----------------------------------------------------------------------
% RetornaTodosValores( Atributo , ListaSignificados , ListaValoresAtrib ) :-
%       dado um determinado "Atributo", retorna uma lista com todos os
%       valores encontrados referentes a esse atributo

 retornaTodosValores( Atributo , ListaSignif , ListaValoresAtrib ) :-
    retornaTodosValores( Atributo , ListaSignif , [] , ListaValoresAtrib )
 .

% --------------
 retornaTodosValores( Atributo , [] , ListaValoresAtrib , ListaValoresAtrib ) .

 retornaTodosValores( Atributo , [(Signif)|Rest]  , L1 , LN ) :-

%   pega o valor do atributo passado no primeiro significado
    retornaValorAtributo( Atributo , Signif , ValorAtrib ) ,

%   insere na lista
    insert( ValorAtrib , L1 , L2 ) ,
```

```
    retornaTodosValores( Atributo , Rest , L2 , LN )

% fim retornaTodosValores
  .


% ------------------------------------------------------------------
% Subordinado( Ag1 , Ag2 ) :- no caso de Ag2 ser subordinado (direta ou
%       indiretamente) do Ag1, esse predicado e' satisfeito. Caso con-
%       trario, ele falha

% --

  subordinado( Ag1 , Ag2 ) :-

     subordinado( Ag1 , _ , Ag2 ) ,
     !  /* encontrou */

% fim subordinado
  .

% --

  subordinado( Ag1 , Ag2 ) :-

    subordinado( Ag1 , _ , AgX ) ,
    AgX \= 0 ,
    subordinado( AgX , Ag2 )

% fim subordinado
  .


% ------------------------------------------------------------------
% TipoAgente( IDAgente , Tipo ) :- retorna em Tipo o tipo do agente
%       identificado por "IDAgente"


  tipoAgente( Agente , Tipo ) :-  agente( Agente , Tipo , _ , _ , _ ) .




% ------------------------------------------------------------------
%    PREDICADOS DE VIZINHANCA
% ------------------------------------------------------------------

% -----
% posicao( Pos , idAgente ) :- dado o idAgente, retorna Pos (e vice-versa)
%
% obs: isso e' armazenado como fato na base de dados

% ------------------------------------------------------------------
% ObtemVizinho( IdAgCham , Relacao , IdAgViz ) :-
%       retorna o "id" do agente vizinho dependendo da relacao
%       ser "maior" ou "menor"

  obtemVizinho( IdAgCham , maior , IdAgViz ) :-

     posicao( PosAgCham , IdAgCham ) .  % obtem a posicao do agente chamador
     PosProxAg is PosAgCham + 1 ,       % incrementa a posicao
     numAgts( NumAgs ) ,                % obtem o numero de agentes
     (   PosProxAg > NumAgs ,           % se nao existe o proximo,
               IdAgViz is 0                     %    sinaliza com zero
     :   posicao( PosProxAg , IdAgViz ) % senao, retorna "id" do vizinho
     )
% fim obtemVizinho /caso maior
  .

% ----
  obtemVizinho( IdAgCham , menor , IdAgViz ) :-

     posicao( PosAgCham , IdAgCham ) .  % obtem a posicao do agente chamador
     PosAgAnt is PosAgCham - 1 ,        % decrementa a posicao
     (   PosAgAnt < 1 ,                 % se nao existe o anterior,
               IdAgViz is 0                     %    sinaliza com zero
     :   posicao( PosAgAnt , IdAgViz )  % senao, retorna "id" do vizinho
     )
% fim obtemVizinho /caso menor
  .


% ------------------------------------------------------------------
% ProcuraVizinho( AgChamador , Direcao , Tipo , IdVizinho ) :- procura por
%       um vizinho (que, se encontrado, sera' retornado em IdVizinho) de
%       um determinado "Tipo" fornecido (verbal, nominal, coordenacao, etc.)
%       em uma determinada "Direcao" com relacao ao "AgtChamador" deste
%       predicado


  procuraVizinho( AgChamador , Direcao , TipoDesejado , IdVizProc ) :-

     obtemVizinho( AgChamador , Direcao , IdViz ) ,
     (   % se idViz = 0 e' porque chegou a um dos extremos da sentenca.
               % ( -> Ver obtemVizinho )
               IdViz = 0 ,
               /* retorna 0, sinalizando que nao encontrou */
               iguala( IdVizProc , 0 )

     :   /* senao e' zero, pega o tipo para ... */
```

```
            agente( IdViz , TipoViz , _ , _ , _ , _ ) ,
            /* ... testar se e' do tipo procurado */
            (   TipoViz = TipoDesejado ,
                /* caso seja, retorna seu identificador */
                igual( IdVizProc , IdViz )

            ;   /* senao, continua a procura a partir de "IdViz" */
                procuraVizinho( IdViz , Direcao , TipoDesejado , IdVizProc )
            )
    ) ,
    !

% fim procuraVizinho
  .


% ------------------------------------------------------------------
% conferePosicionamento( relacao , agt1 , agt2 ) :- verifica se
%       ( agt2 relacao agt1 ) e' satisfeita , onde relacao pode
%       ser: maior , menor ou diferente.

%       obs.: esse tipo de pedido de local e conferencia nao
%             trata de intervalos fechados

  conferePosicionamento( maior , Agt1 , Agt2 ) :-

    Agt2 > Agt1 , !
  .

%
  conferePosicionamento( menor , Agt1 , Agt2 ) :-

    Agt2 < Agt1 , !
  .

%
  conferePosicionamento( qualquer , Agt1 , Agt2 ) :-

    not( Agt1 = Agt2 )
  .

% ------------------------------------------------------------------
% ObtemSuperior( IdAg , IdSupAg ) :- retorna em IdSupAg o superior do
%       agente IdAg. Esse predicado e' so' para separar o que e' in-
%       formacao dentro do escopo do agente e o que deve ser obtido
%       via outro agente. Assim, se um agente, em seu "acorda", preci-
%       sa ter seu superior eu estou usando "superior". Se ele precisa
%       saber o superior de um outro agente, estou usando "obtemSuperi-
%       or".

  obtemSuperior( IdAg , IdSupAg ) :-
    superior( IdAg , _ , IdSupAg ) ,
    !
  .

% ------------------------------------------------------------------
% ObtemSujeito( IdAgtVerb , IdSujAgtVerb ) :- retorna em IdSujAgtVerb o
%       sujeito do agente verbal "IdAgtVerb". Retornara' zero caso nao
%       exista o sujeito (conforme a inicializacao de todos os agentes.

  obtemSujeito( IdAgtVerb , IdSujAgtVerb ) :-

    (   subordinado( IdAgtVerb , suj , IdSujAgtVerb )
    ;   /* no caso de algum erro */
            IdSujAgtVerb is 0
    ) ,
    !
% fim obtemSujeito
  .

% ------------------------------------------------------------------
% ObtemObjeto( IdAgtVerb , IdObjAgtVerb ) :- ver comentario para o pre-
%       dicado ObtemSujeito logo acima

  obtemObjeto( IdAgtVerb , IdObjAgtVerb ) :-

    (   subordinado( IdAgtVerb , obj , IdObjAgtVerb )
    ;   /* no caso de algum erro */
            IdObjAgtVerb is 0
    ) ,
    !
% fim obtemObjeto
  .

% ------------------------------------------------------------------
%   PREDICADOS PARA A FUNCAO DE ALEATORIEDADE
% ------------------------------------------------------------------

% ------------
% Random( X , NumAgentes ) :- retorna em "X" um identificador de um
%       agente que estava em uma lista que foi formada de maneira
%       aleatoria e colocada na base de dados. Para o numero de agentes da
%       frase sera' formada uma lista inicial onde estarao os identificadores
%       dos agentes formando blocos de "todos" -- cada bloco possui "id" de
%       todos os agentes -- com a aleatoriedade introduzida em cada
%       bloco (devem ser mais ou menos 3 ou 4 blocos). Cada vez que e'
%       chamado esta funcao retorna o primeiro da lista e o retira da
%       lista -- que e' entao salva. Quando a lista esta' vazia, pega-se
%       uma nova lista que estara' na base de dados (formando-se assim
```

```
%       uma circularidade).

  random( X , NumAgentes ) :-

     listaAleatoria( NumAgentes , Lista1 ) ,
     retornaLista( Lista1 , Lista2 , NumAgentes ) ,
     select( X1 , Lista2 , ListaN ) ,
     retract( listaAleatoria( NumAgentes , Lista1 ) ) ,
     assert( listaAleatoria( NumAgentes , ListaN ) ) ,
     igual( X , X1 ) ,
     !

% fim random
  .

% ------------
% RetornaLista( Lista1 , Lista2 , NumAgentes ) :- funcao de inicializacao e
%       reinicializacao da "lista aleatoria". Quando a lista fica vazia,
%       que e' no inicio do programa ou quando ela ja' foi toda "rodada",
%       forma-se "Lista2" a partir de uma lista inicial. Quando nao esta'
%       vazia, "Lista2" e' igual a "Lista1".

  retornaLista( [] , Lista , NumAgentes ) :-

     listaInicial( NumAgentes , Lista )

  .


  retornaLista( Lista , Lista , _NumAgentes ) .

% -------------
% InicializaListasAleatorias :- salva as listas iniciais e
%       cria a lista aleatoria que inicialmente sera' vazia

  inicializaListasAleatorias :-

     numAgts( NumAgentes ) ,
     assert( listaAleatoria( NumAgentes , [] ) ) ,

     (   NumAgentes = 1 ,
            assert( listaInicial( 1 , [1,1,1,1,1,1,1,1,1,1] ) )

     ;   NumAgentes = 2 ,
            assert( listaInicial( 2 , [2,1,2,1,2,1] ) )

     ;   NumAgentes = 3 ,
            assert( listaInicial( 3 , [2,1,3,1,3,2,3,2,1,2,3,1,3,1] ) )

     ;   NumAgentes = 4 ,
            assert( listaInicial( 4 , [2,4,3,1,4,2,3,1,2,3,4,1,2,1,4,3,2] ) )

     ;   NumAgentes = 5 ,
            assert( listaInicial( 5 , [2,4,5,3,1,2,3,1,5,4,2,3,1,4,5] ) )

     ;   NumAgentes = 6 ,
            assert( listaInicial( 6 , [6,1,3,4,5,2,1,6,3,5,4,1,2,3,4,5,6] ) )

     ;   NumAgentes = 7 ,
            assert( listaInicial( 7 , [1,2,6,4,3,7,5,1,6,3,5,7,4,1,2,7,3,4,5,6]
) )

     ;   NumAgentes = 8 ,
                        assert(         listaInicial(         8         ,
[8,4,6,5,1,7,2,3,1,8,6,3,5,7,4,1,2,7,3,8,4,5,6] ) )

     ;   NumAgentes = 9 ,
                        assert(         listaInicial(         9         ,
            [2,8,4,6,5,1,7,3,9,2,1,8,6,9,3,5,7,4,1,2,7,9,3,8,4,5,6] ) )

     ;   NumAgentes = 10 ,
                        assert(         listaInicial(         10        ,
            [2,4,6,1,7,9,8,3,10,5,2,1,8,10,6,9,3,5,7,4,1,2,7,9,10,3,8,4,5,6] ) )

     ;   NumAgentes = 11 ,
                        assert(         listaInicial(         11        ,
            [2,11,6,5,1,10,7,9,8,4,3,2,1,8,10,6,9,11,3,5,7,4,1,2,11,7,9,10,3,8,4,
5,6] ) )

%    nao houve frases com mais de onze agentes

     )

% fim inicializaListasAleatorias
  .



% ---------------------------------------------------------------------
% inicializaListasAgentes :- para cada tipo de agente, cria uma lista e
%       insere todos os agentes desse tipo (ex.: nominal, verbal, etc.).


  inicializaListasAgentes :-

     foreach( setof( Agente,
                        P^D^S^agente( Agente , TipoAgente , P , D , S ) ,
                        ListaAgentes ) ,
                   /* do */
                   assert( listaAgentes( TipoAgente , ListaAgentes ) )
                 )
```

```
% fim inicializaListasAgentes
  .

% ------------------------------------------------------------------------
%    PREDICADOS GERAIS
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% FOREACH :- for each X do Y

  foreach( X , Y ) :-
     X ,
     do( Y ) ,
     fail
  .


  foreach( X , Y ) :-
     true
  .

%
  do( Y ) :- Y , ! .

% ------------------------------------------------------------------------
% Retractall( X ) :-
%       remove all clauses that match X , C&M pg.172

  retractall( Clause ) :-
     retract( Clause ) ,
     fail
  .
  retractall( _ ) .


% ------------------------------------------------------------------------
% InList( X , Xs ) :-
%       verifica se o termo X (atomico) esta' na lista Xs.

  inList( X , [X|_] ) :- atomic( X ) , ! .

  inList( X , [Z | Y] ) :- inList( X , Y ) .

% ------------------------------------------------------------------------
% Delete( Lista , X , SemX ) :-
%       a lista SemX e' o resultado de remover
%       todas as ocorrencias de X da lista Lista

  delete( [X|Xs] , X , Ys ) :- delete( Xs , X , Ys ).

  delete( [X|Xs] , Z , [X|Ys] ) :- X \= Z , delete( Xs , Z , Ys ) .

  delete( [ ] , X , [ ] ) .


% ------------------------------------------------------------------------
% Insert( X , Ys , Zs ) :- insere X no inicio da lista Ys retornando em Zs

  insert( X , Ys , [X|Ys] ) .


% ------------------------------------------------------------------------
% Select( X , HasXs , OneLessXs ) :-  A lista "OneLessXs" e' o resultado
%       da remocao de "X" da lista "HasXs"

  select( X , [X|Xs] , Xs ) .

  select( X , [Y|Ys] , [Y ,Zs] ) :- select( X , Ys , Zs ) .


% ------------------------------------------------------------------------
% ordena( Xs , Ys ) :- Ys e' a lista Xs ordenada

  ordena( [X|Xs] , Ys ) :-

     particao( Xs , X , Littles , Bigs ) ,
     ordena( Littles , Ls ) ,
     ordena( Bigs , Bs ) ,
     append( Ls , [X|Bs] , Ys )
  .

  ordena( [] , [] ) .

  particao( [X|Xs] , Y , [X|Ls] , Bs ) :-

     X =< Y ,
     particao( Xs , Y , Ls , Bs )
  .

  particao( [X|Xs] , Y , Ls , [X|Bs] ) :-

     X > Y ,
     particao( Xs , Y , Ls , Bs )
  .

  particao( [] , Y , [] , [] ) .


% ------------------------------------------------------------------------
% append( Xs , Ys , XsYs )
```

```
  append( [] , Ys , Ys ).
  append( [X|Xs] , Ys , [X|Zs] ) :- append( Xs , Ys , Zs ).

% ------------------------------------------------------------------
% Iguala dois termos , um dos quais deve ser "unground"

  iguala( X , X ) .

% ------------------------------------------------------------------
% RetornaValorAtributo( Atributo , ListaAtributo , Valor ) :- esse predicado
%       ira' manipular a lista de atributos das palavras do dicionario.
%       Dada uma lista de atributos (como a do dicionario) e especificado
%       um atributo qualquer , este predicado retorna em "Valor" o valor
%       que este atributo possui na lista. Se nao existir tal atributo ,
%       entao "Valor" sera' igualado 'a (lista vazia) (=[]).

  retornaValorAtributo( Atributo , [] , [[]] ) .

  retornaValorAtributo( Atributo , [X|Resto] , Valor ) :-

     (   not( iguala( Atributo=Y , X ) ) ,
              retornaValorAtributo( Atributo , Resto , Valor )

     ;   iguala( Atributo=[Valor] , X )

     ) ,

     !

% fim retornaValorAtributo
  .

% ------------------------------------------------------------------
% RetornaAtributoDadoChave( Chave , AtribChave , Atributo ,
%                           ListaSignificados , ListaAtributos ) :-
%
%       dado uma determinada palavra "Chave" , verifica se o atributo onde
%       ela pode aparecer "AtribChave" a contem e, isso ocorrendo, retorna
%       o valor contido no "Atributo" passado, formando uma lista com todos
%       os valores encontrados referentes a esse atributo

  retornaAtributoDadoChave( Chave , AtribChave , Atributo ,
                            ListaSignif , ListaAtributos ) :-

     retornaAtributoDadoChave( Chave , AtribChave , Atributo ,
                               ListaSignif , [] , ListaAtributos )
  .

% --------------

  retornaAtributoDadoChave( Chave , AtribChave , Atributo ,
                            [] , ListaAtributos , ListaAtributos ) .

  retornaAtributoDadoChave( Chave , AtribChave , Atributo ,
                            [[Signif]|Rest] , L1 , LN ) :-

%    pega o atributo semantico do primeiro significado
     retornaValorAtributo( Atributo , Signif , TipoSem ) ,
     retornaValorAtributo( AtribChave , Signif , Chave2 ) ,

     (   iguala( Chave , Chave2 ) ,
%        insere na lista
              insert( TipoSem , L1 , L2 )
     ;   /* continua */
              not( iguala( Chave , Chave2 ) ) ,
              true
     ) ,

     retornaAtributoDadoChave( Chave , AtribChave ,
                               Atributo , Rest , L2 , LN )

% fim retornaAtributoDadoChave
  .


% ------------------------------------------------------------------
%    DICIONARIO - contem informacoes semanticas
% ------------------------------------------------------------------

% DETERMINANTES

  dicionario([palavra=[o] , cat=[det] , tipo=[def] , num=[sing] , gen=[masc] ]).

  dicionario([palavra=[os] , cat=[det] , tipo=[def] , num=[plur] , gen=[masc] ]).

  dicionario([palavra=[a] , cat=[det] , tipo=[def] , num=[sing] , gen=[fem] ]).

  dicionario([palavra=[um] , cat=[det] , tipo=[indef] , num=[sing] , gen=[masc] ]).


% NOMES


  dicionario([ palavra=[joao] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[marcos] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[maria] , cat=[nome] , pessoa=[3] , num=[sing] ,
```

```
                    gen=[fem] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[malu] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[fem] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[menina] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[fem] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[menino] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[meninos] , cat=[nome] , pessoa=[3] , num=[plur] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[humano] ]] ]] ]).

  dicionario([ palavra=[presente] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[obj_fis] ]] ]] ]).

  dicionario([ palavra=[gato] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[felino] ]] ]] ]).

  dicionario([ palavra=[cachorro] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[canino] ]] ]] ]).

  dicionario([ palavra=[revolver] , cat=[nome] , pessoa=[3] , num=[sing] ,
                     gen=[masc] , significado=[[ [[ tip_sem=[arma_de_fogo] ]] ]]
]).

  dicionario([ palavra=[maquina] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[fem] , significado=[[ [[ tip_sem=[obj_fis] ]] ]] ]).

  dicionario([ palavra=[telescopio] , cat=[nome] , pessoa=[3] , num=[sing] ,
                    gen=[masc] , significado=[[ [[ tip_sem=[instr_otico] ]] ]] ]).


% PREPOSICOES

% obs.: "tip_sem" e' o tipo semantico do COMPLEMENTO que acompanha a preposicao

  dicionario([ palavra=[com] , cat=[prep] ,
                     significado=[[ [[ pap_tem=[acompanhante] , tip_sem=[obj_fis]
]] ,
                                    [[ pap_tem=[instrumento] , tip_sem=[instrumento] ]]
                                  ]]
                    ]).

  dicionario([ palavra=[para] , cat=[prep] ,
                    significado=[[ [[ pap_tem=[destino] , tip_sem=[local] ]] ,
                                    [[ pap_tem=[destinatario] , tip_sem=[humano] ]]
                                  ]]
                    ]).


% PRONOME RELATIVO

  dicionario([ palavra=[que] , cat=[pron_rel] ]).


% COORDENACAO

  dicionario([ palavra=[e] , cat=[conjuncao] ]).

% VERBOS

  dicionario([ palavra=[matou] ,
                    cat=[verbo] ,
                    pessoa=[3] ,
                    num=[sing] ,
                    tempo=[passado] ,
                    significado=[[ [[ tip_sem=[causar_a_morte] ,
                            obrig=[[ suj=[[ [[ pap_tem=[agente] ,
                                               tip_sem=[humano] ]] ,
                                          [[ pap_tem=[instrumento] ,
                                             tip_sem=[arma] ]]
                                        ]] ,
                                     obj=[[ [[ pap_tem=[tema] ,
                                               tip_sem=[animado] ]]
                                          ]]
                                   ]] ,
                            opcion=[[ obj_ind=[[ [[ pap_tem=[instrumento] ,
                                                    tip_sem=[arma] ,
                                                    prep=[com] ]]
                                                 ]]
                                    ]]
                                  ]]
                                ]]  % outros tipos viriam dentro deste colchete
                   ]).

  dicionario([ palavra=[viu] ,
                    cat=[verbo] ,
                    pessoa=[3] ,
                    num=[sing] ,
                    tempo=[passado] ,
                    significado=[[ [[ tip_sem=[ver_algo] ,
                            obrig=[[ suj=[[ [[ pap_tem=[agente] ,
                                               tip_sem=[humano] ]]
                                          ]] ,
                                     obj=[[ [[ pap_tem=[tema] ,
                                               tip_sem=[objeto] ]]
                                          ]]
                                   ]] ,
                            opcion=[[ obj_ind=[[ [[ pap_tem=[instrumento] ,
```

```
                                    tip_sem=[instr_otico] ,
                                    prep=[com] ]]
                                  ])
                            ]]
                        ]]
                    ]]  % outros tipos viriam dentro deste colchete
            ]).

dicionario([ palavra=[caiu] ,
            cat=[verbo] ,
            pessoa=[3] ,
            num=[sing] ,
            tempo=[passado] ,
            significado=[[ [[ tip_sem=[cair_no_chao] ,
                    obrig=[[ suj=[[ [[ pap_tem=[paciente] ,
                                tip_sem=[obj_fis] ]]
                            ]]
                        ]]
                    ]]  % outros tipos viriam dentro deste colchete
            ] ).

dicionario([ palavra=[cairam] ,
            cat=[verbo] ,
            pessoa=[3] ,
            num=[plur] ,
            tempo=[passado] ,
            significado=[[ [[ tip_sem=[cair_no_chao] ,
                    obrig=[[ suj=[[ [[ pap_tem=[paciente] ,
                                tip_sem=[obj_fis] ]]
                            ]]
                        ]]
                    ]]  % outros tipos viriam dentro deste colchete
            ] ).

dicionario([ palavra=[ama] ,
            cat=[verbo] ,
            pessoa=[3] ,
            num=[sing] ,
            tempo=[presente] ,
            significado=[[ [[ tip_sem=[amar] ,
                    obrig=[[ suj=[[ [[ pap_tem=[agente] ,
                                tip_sem=[humano] ]]
                            ]] ,
                        obj=[[ [[ pap_tem=[tema] ,
                                tip_sem=[objeto] ]]
                            ]]
                        ])
                    ]]
                ]]  % outros tipos viriam dentro deste colchete
            ]).


dicionario([ palavra=[operou] ,
            cat=[verbo] ,
            pessoa=[3] ,
            num=[sing] ,
            tempo=[passado] ,
            significado=[[ [[ tip_sem=[fazer_funcionar] ,
                    obrig=[[ suj=[[ [[ pap_tem=[agente] ,
                                tip_sem=[humano] ]]
                            ]] ,
                        obj=[[ [[ pap_tem=[tema] ,
                                tip_sem=[objeto] ]]
                            ]]
                        ])
                    ]]
                ]]  % outros tipos viriam dentro deste colchete
            ]).


dicionario([ palavra=[deu] ,
            cat=[verbo] ,
            pessoa=[3] ,
            num=[sing] ,
            tempo=[passado] ,
            significado=[[ [[ tip_sem=[dar_algo_para_alguem] ,
                    obrig=[[ suj=[[ [[ pap_tem=[agente] ,
                                tip_sem=[humano] ]]
                            ]] ,
                        obj=[[ [[ pap_tem=[tema] ,
                                tip_sem=[objeto] ]]
                            ]] ,
                        obj_ind=[[ [[ pap_tem=[paciente] ,
                                tip_sem=[humano] ,
                                prep=[para] ]]
                            ]]
                        ])
                    ]]  % outros tipos viriam dentro deste colchete
            ]).


% ------------------------------------------------------------------
%    BASE DE CONHECIMENTO
% ------------------------------------------------------------------

  isa( humano , animado ) :- ! .
  isa( arma_de_fogo , arma ) :- ! .
```

```
  isa( arma , instrumento ) :- ! .
  isa( felino , animal ) :- ! .
  isa( canino , animal ) :- ! .
  isa( animal , animado ) :- ! .
  isa( animado , obj_fis ) :- ! .
  isa( instr_otico , instrumento ) :- ! .
  isa( instrumento , obj_fis ) :- ! .
  isa( obj_fis , objeto ) :- ! .
  isa( objeto , objeto ) :- ! .
  isa( objeto , _ ) :- ! , fail .

% reflexao
  isa( T1 , T2 ) :- atomic( T1 ) , atomic( T2 ) , igual( T1 , T2 ) , ! .

% transitividade
  isa( A , C ) :- isa( A , B ) , isa( B , C ) .


% ------------------------------------------------------------------
%    FRASES DE TESTE
% ------------------------------------------------------------------

  test1 :-  test( [o, menino, matou, o, gato] ) .
  test2 :- test( [o, menino, matou, o, gato, com, o, revolver] ) .
%  test3 :- test( [joao, ama, maria, e, joao, deu, um, presente, para, maria] ) .
  test4 :- test( [o, menino, que, ama, a, menina, caiu] ) .
  test5 :- test( [joao, ama, maria, e, marcos, malu] ) .
  test6 :- test( [joao, matou, um, gato, e, maria, um, cachorro] ) .
  test7 :- test( [caiu, o, menino] ) .
%%  test8 :- test( [caimos] ) .
  test9 :- test( [operou, a, maquina, o, menino] ) .
  test10 :- test( [a, maquina, operou, o, menino] ) .
  test11 :- test( [o, menino, que, ama, a, menina, que, caiu, matou, o, gato] ) .
  test12 :- test( [o, menino, que, a, menina, que, caiu, ama, matou, o, gato] ) .
%%  test13 :- test( [vivemos, sem, preconceito] ) .
  test14 :- test( [os, meninos, cairam] ) .
  test15 :- test( [joao, que, ama, maria, e, malu, caiu] ) .
  test16 :- test( [joao, que, ama, maria, e, malu, cairam] ) .
  test17 :- test( [joao, que, ama, maria, e, marcos, que, ama, malu, cairam] ) .
  test18 :- test( [joao, que, ama, maria, e, marcos, que, ama, malu, caiu] ) .
  test19 :- test( [joao, ama, maria, e, marcos, ama, malu] ) .
  test20 :- test( [joao, e, marcos, cairam] ) .
  test21 :- test( [joao, ama, maria, marcos, ama, malu] ) .
  test22 :- test( [o, menino, ama] ) .
  test23 :- test( [o, menino, que, ama, maria] ) .
  test24 :- test( [o, menino, que, joao, ama, maria, caiu] ) .
  test25 :- test( [o, menino, caiu, e, maria] ) .
  test26 :- test( [joao, ama, e, maria, matou, marcos] ) .
  test27 :- test( [joao, caiu, e, ama, maria] ) .
  test28 :- test( [joao, cairam] ) .
  test29 :- test( [cairam, joao, e, maria] ) .
  test30 :- test( [joao, ama, maria, e, ama] ) .
  test31 :- test( [joao, ama, maria, e, ama, malu] ) .
  test32 :- test( [joao, ama, maria, e, ama, malu, joao] ) .
  test33 :- test( [o, menino, viu, a, menina, com, o, telescopio] ) .
  test34 :- test( [joao, marcos, ama, e, viu, a, menina, o, telescopio] ) .
  test35 :- test( [joao, marcos, ama, e, caiu, maria] ) .
  test36 :- test( [joao, marcos, e, ama, caiu, maria] ) .
  test37 :- test( [joao, que, caiu, e, que, marcos, matou, ama, maria] ) .
```

Referências Bibliográficas

A year working in Artificial Intelligence is enough to make one believe in God.[67]

[Adriaens86] ADRIAENS, G., WEP (Word Expert Parser) Revised and Applied to Dutch, In: *Proceedings of the 7th European Conference on Artificial Intelligence* – ECAI86, Brighton, p. 222-235, 1986.

[Allen95] ALLEN, J., *Natural Language Understanding*, *2nd ed.*, Benjamin Cummings, 1995.

[Berthet92] BERTHET, S.; DEMAZEAU, Y.; BOISSIER, O., Knowing Each Other Better. *11th International Workshop on Distributed Artificial Intelligence*, Glen Arbor, 1992.

[Bourgois91] BOURGOIS, M., *The Parallel Expert Parser Comes of Age*. Master Thesis, Katholieke Universiteit Leuven, 1991.

[Bussmann94] BUSSMANN, S.; DEMAZEAU, Y., An Agent Model Combining Reactive and Cognitive Capabilities. In: *IEEE International Conference on Intelligent Robots and Systems* – IROS94, Munique, 1994.

---

[67] Alan J. Perlis em "Epigrams on programming." *SIGPLAN notices*, 17(9), September 1982, 7-13. Citado em [Hirst87] à página 164.

[Campbell90] CAMPBELL, J.A.; D'INVERNO, M.P., Knowledge Interchange Protocols, In: [Demazeau90a], p. 63-80, 1990.

[Castelfranchi90] CASTELFRANCHI, C., Social Power: A Point Missed in Multi-Agent, DAI and HCI. In: [Demazeau90a], p. 49-62,1990.

[Chao87] CHAO, W., *On Ellipsis*, Ph.D. Thesis, Dept. of Linguistics, University of Massachusetts-Amherst, 1987.

[Charniak81] CHARNIAK, E., The case-slot identity theory. *Cognitive Science*, 5(3), p. 285-292, 1981.

[Cohen87] COHEN, P.R.; LEVESQUE, H.J., Intention = Choice + Commitment. In: *Proceedings of the 6th Conference of American Association for Artificial Intelligence - AAAI87*, Seattle, 1987.

[Davis83] DAVIS, R.; SMITH, R.G., Negotiation as a metaphor for distributed problem solving, *Artificial Intelligence* 20, p. 63-109, 1983.

[Demazeau90a] DEMAZEAU, Y.; MÜLLER, J-P. (eds.), *Decentralized Artificial Intelligence*. Elsevier Science Publishers, 1990.

[Demazeau90b] DEMAZEAU, Y.; MÜLLER, J-P., Decentralized Artificial Intelligence. In: [DEMAZEAU90a], 1990.

[Demazeau91] DEMAZEAU, Y.; MÜLLER, J-P., From Reactive to Intentional Agents. In: DEMAZEAU, Y.; MÜLLER, J-P. (eds.) *Decentralizes Artificial Intelligence 2*, Elsevier Science Publishers, 1991.

[Demazeau93a] DEMAZEAU, Y. La plate-forme PACO et ses applications, *2ème Journée Nationale du PRC-IA sur les Système Multi-Agents*, PRC-IA, Montpellier, 1993.

[Demazeau93b] DEMAZEAU, Y.; BOISSIER, O.; KONING, J-L., Using Interaction Protocols to Control Vision Systems. In: *IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, 1994.

[Devos88] DEVOS, M.; ADRIAENS, G.; WILLEMS, Y.D., 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* – COLING88, Budapest, p. 142-147, 1988.

[Devos94] DEVOS, M.; ADRIAENS, G., The Parallel Expert Parser. In: ADRIAENS, G.; HAHN, U. (eds.), *Parallel Natural Language Processing*, Ablex, p. 350-375, 1994.

[Drogoul92] DROGOUL, A.; DUBREUIL, C., Eco-Problem-Solving Model: Results of the N-Puzzle. In: WERNER, E.; DEMAZEAU, Y. (eds.), *Decentralized Artificial Intelligence 3*, Elsevier Science Publishers, p. 283-295, 1992.

[Drogoul94] DROGOUL, A.; FERBER, J., Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies. In: CASTELFRANCHI, C; WERNER, E. (eds.), *Artificial Social Systems*, Lectures Notes in Artificial Intelligence 830, Springer-Verlag, p. 3-23, 1994.

[Drogoul95] DROGOUL, A., When Ants Play Chess (Or Can Strategies Emerge From Tactical Behaviours?). In: CASTELFRANCHI, C; MÜLLER, J-P. (eds.), *From Reaction to Cognition*, Lectures Notes in Artificial Intelligence 957, Springer-Verlag, p. 13-27, 1995.

[Durfee94] DURFEE, E.H.; ROSENSCHEIN, J.S., Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples. In: *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*. Seattle, WA, 1994.

[Engelmore88] Engelmore, R.S.; Morgan, A.J. (eds.), *Blackboard Systems*, Addison-Wesley, 1988.

[Ferber91] FERBER, J.; JACOPIN, E., The Framework of Eco-Problem Solving. In: DEMAZEAU, Y.; MÜLLER, J-P. (eds.), *Decentralized Artificial Intelligence 2*, Elsevier Science Publishers, p. 181-193, 1991.

[Ferber94] FERBER, J., Un modèle de l'action pour les systèmes multi-agents. *2ème Journées Francophones IAD&SMA*, Voiron, 1994.

[Hahn94] HAHN, U.; ADRIAENS, G., Parallel Natural Language Processing: Background and Overview. In: In: ADRIAENS, G.; HAHN, U. (eds.), *Parallel Natural Language Processing*, Ablex, p. 1-134, 1994.

[Hays64] HAYS, D.G., Dependency Theory: A Formalism and some Observations, *Language* 40, p. 511-525, 1964.

[Hirst87] HIRST, G., *Semantic interpretation and the resolutions of ambiguity*. Cambridge University Press, 1987.

[Hirst88] HIRST, G., Semantic Interpretation and Ambiguity, *Artificial Intelligence* 34, p. 131-177, 1988.

[Marslen-Wilson87] MARSLEN-WILSON, W.; TYLER, L.K., Against Modularity, In: GARFIELD, J.L. (ed.), *Modularity in Knowledge Representation and Natural-Language Understanding*, MIT Press, p. 37-62, 1987.

[Martial90] von MARTIAL, F., Interactions Among Autonomous Planning Agents. In: [Demazeau90a], p. 105-119, 1990.

[Milward94] MILWARD, D., Dynamic Dependency Grammar, Linguistics and Philosophy 17, p. 561-605, 1994.

[Palmer93] PALMER, M.S.; PASSONEAU, R.J.; WEIR, C.; FININ, T., The KERNEL text understanding system, *Artificial Intelligence* 63, 1993.

[Perini96] PERINI, Mário A., *Gramática descritiva do português.* Editora Ática, 1996.

[Rao91] RAO, A.S.; GEORGEFF, M.P., Modeling Rational Agents within a BDI-Architecture. In: ALLEN, J.; FIKES, R.; SANDEWALL, E. (eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning* – KR91, Morgan Kaufmann, San Mateo, CA, 1991.

[Rao95] RAO, A.S.; GEORGEFF, M.P., BDI Agents: From Theory to Practice. In: *Proceedings of the 1st International Conference on Multi-Agent Systems* – ICMAS95, San Francisco, CA, 1995.

[Sag84] SAG, I.; GAZDAR, G.; WASOW, T.; WEISLER, S., *Coordination and How to Distinguish Categories.* Tech. Rep. 84-3, CSLI, 1984.

[Sichman92] SICHMAN, J.S.; DEMAZEAU, Y.; BOISSIER, O., When can knowledge-based systems be called agents? In: *Anais do IX SBIA*, Rio de Janeiro, p. 172-185, 1992.

[Sichman94] SICHMAN, J.S.; CONTE, R.; DEMAZEAU, Y.; CASTELFRANCHI, C., A Social Reasoning Mechanism Based on Dependence Networks. *12th European Conference on Artificial Intelligence* – ECAI94, Amsterdã, 1994.

[Sichman95] SICHMAN, J.S., *Du raisonnement Social Chez less Agents: Une Approche Fondée sur la Théorie de la Dépendance.* Ph.D. Thesis, Institut National Polytechnique de Grenoble, France, Capítulo 2, 1995.

[Small80] SMALL, S., *Word Expert Parsing: A Theory of Distributed Word-Based Natural Language Understanding.* Ph.D. Thesis, Dept. Comp. Science, University of Maryland, 1980.

[Steels90] STEELS, L., Cooperation between Distributed Agents through Self-Organization. In: [Demazeau90a], p. 175-196, 1990.

[Stefanini93] STEFANINI, M.H., *Talisman: Une Architecture Multi-Agents pour l'Analyse du Français Écrit.* Thèse de Doctorat, Université Pierre Mendès-France, 1993.

[Stefanini95] STEFANINI, M.H.; DEMAZEAU, Y., TALISMAN: A Multi-Agent System for Natural Language Processing, In: WAINER, J., CARVALHO, A., (eds.) *Lectures Notes in Artificial Intelligence 991*, Springer-Verlag, p. 312-322, 1995.

[Wavish95] WAVISH, P.; GRAHAM, M., Roles, Skills and Behaviour. In: WOOLDRIDGE, M.J; JENNINGS, N.R. (eds.), *Intelligent Agents*, p. 371-385, Berlin, Springer-Verlag, 1995.

[Werner90] WERNER, E., Distributed Cooperation Algorithms, In: [Demazeau90a], 1990.