JOÃO BATISTA CAMARGO JUNIOR

ASPECTOS DO PROCESSAMENTO DE INTERFACES EM LINGUAGEM NATURAL

Dissertação apresentada à Escola Politécnica da USP para obtenção do Título de Mestre em Engenharia Elétrica.

São Paulo
1989

160P

F.D. 1031

JOÃO BATISTA CAMARGO JUNIOR

Engenheiro Eletrônico, Escola Politécnica da USP, 1981

ASPECTOS DO PROCESSAMENTO DE INTERFACES EM LINGUAGEM NATURAL

Dissertação apresentada à Escola Politécnica da USP para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador:
Prof. Dr. João José Neto

São Paulo
1989

Aos

meus pais

AGRADECIMENTOS

Ao Professor Doutor João José Neto, pela orientação precisa, grande incentivo e confiança.

A todos os companheiros da FDTE/EPUSP, que de alguma forma tenham contribuido para a realização deste trabalho.

À Meire Inevaso, pelo cuidadoso trabalho de datilografia.

À minha tia Maria Antonieta, pela inestimável colaboração na revisão do texto deste trabalho.

Ao meu irmão Paulo de Tarso de Almeida Camargo, pela importante colaboração na execução dos desenhos.

Aos meus pais e irmãos, pelo grande incentivo e apoio familiar.

RESUMO

Esta dissertação apresenta alguns formalismos usados no tratamento computacional de linguagens naturais, bem como uma proposta de método de processamento para as mesmas, envolvendo as fases de tradução, planejamento e execução.

A etapa de tradução consiste da análise, interpretação e determinação do escopo de sentenças interrogativas. Esta etapa traduz sentenças em linguagem natural para uma forma lógica que representa sua semântica.

Na etapa de planejamento, a forma lógica, obtida na etapa de tradução, é convertida em uma regra Prolog a ser interpretada durante a etapa de execução.

A principal etapa no processamento de linguagem natural é a etapa de tradução. Alguns formalismos, tais como a Gramática de Cláusulas Definidas - DCG, e a Gramática de Extraposição - XG, são discutidos em detalhe, para ilustrar os processos usados durante a tradução.

Em seguida é apresentado um protótipo que implementa o interfaceamento de uma base de dados em linguagem natural, no caso um sub-conjunto restrito da lingua portuguêsa.

Finalmente são feitos alguns comentários sobre a perspectiva da utilização da linguagem natural em diversos campos da computação, tais como entendimento de texto, programação automática e engenharia de software.

ABSTRACT

This work presents a methodology and some formalisms to be used in natural language processing.

The present proposal manipulates natural languages by appling three processing steps: translation, planning and execution

The translation step consists of parsing, interpreting, and determining the scope of the sentences. This step maps natural language sentences into some logical form that represents its semantics.

In the planning step the logical form, obtained in the translation step, is converted into a Prolog rule to be interpreted during the execution step.

The most important phase of natural language processing is the translation step. Some formalisms, like Definitive Clause Grammar - DCG and Extraposition Grammar - XG are discussed in detail to illustrate the methods used by the translation step.

Next, is presented a prototype that implements a natural language interface to a database, by using a restrict subset of portuguese language.

Finally, some comments are made about the perspectives of using natural language in some fields of computation, such as text understanding, automatic programming and software engineering.

SUMÁRIO

Página

1. Introdução 001
   1.1. Apresentação 001
   1.2. Visão Geral 002

2. Linguagem Natural na Computação 005
   2.1. Entendimento de Texto-Análise 011
   2.2. Síntese de Texto 014
   2.3. Programação Automática 015

3. Formalismos para o Processamento de Linguagem Natural . 021
   3.1. A Evolução da Gramática Livre de Contexto para a Gramática Transformacional 021
   3.2. Rede de Transição Estendida-ATN ("Augmented Transition Networks") 023
   3.3. Gramática de Cláusulas Definidas - DGC 025
      3.3.1. Descrição Geral do Formalismo - DCG 026
      3.3.2. A Função da Variável Lógica - DCG 032
      3.3.3. Razões pela Opção do DCG 033

4. Descrição Detalhada do Formalismo Adotado 036
   4.1. Descrição Informal do XG 036
   4.2. Descrição Formal do XG 037
      4.2.1. Grafos de Derivação 039
      4.2.2. A Gramática XG e a Extraposição à Esquerda 041
   4.3. A Gramática XG como Ferramenta de Programação Lógica 047

5. Implementação de um Protótipo 051
   5.1. Metodologia de Projeto 051
      5.1.1. O Processo de Tradução 053
      5.1.2. O Processo de Planejamento e Execução 056

5.2. Especificação dos Requisitos do Software - ERS .. 057
5.2.1. Introdução 058
5.2.2. Descrição Geral 058
5.2.3. Especificação dos Requisitos 064
5.3. Descrição do Projeto de Software - DPS 077
5.3.1. Introdução 077
5.3.2. Descrição de Decomposição 077
5.3.3. Descrição de Dependência 091
5.3.4. Descrição da Interface do Software- Produto 093
5.3.5. Descrição do TRADUTOR-AUTOMÁTICO 096

6. Avaliação 100
6.1. Interface Homem-Máquina e Aprendizado 100
6.2. Pontuação 101
6.3. Otimização de Regras 101
6.4. Ambiguidades 103
6.5. Transcrição de Discurso 105
6.6. Revisão de Texto 108
6.7. Aplicação à Engenharia de Software 109
6.8. Interface com Banco de Dados Reais 112

7. Conclusão 114

1. INTRODUÇÃO

1.1. Apresentação

Esta dissertação é composta por sete capítulos. O primeiro capítulo tem a função de apresentar a estrutura do trabalho como também dar uma rápida visão dos seus principais objetivos.

No Capítulo 2 será apresentado um panorama geral das aplicações da linguagem natural no campo da computação, como por exemplo, "query-language", entendimento de texto e programação automática. Ainda neste capítulo, serão feitas algumas considerações relativas à perspectiva futura da linguagem natural, na Engenharia de Software.

No Capítulo 3 será apresentada uma evolução dos formalismos existentes que são utilizados no processamento de linguagem natural. Será apresentada também uma breve descrição de cada formalismo, mostrando suas vantagens e desvantagens relativas.

No Capítulo 4 será apresentada uma descrição detalhada do formalismo adotado para o processamento de linguagem natural, como interface com bases de dados.

No Capítulo 5 será apresentada a implementação de um protótipo de teste, ou seja, um Software-Produto denominado "INTERFACE COM BANCO DE DADOS EM LINGUAGEM NATURAL". Basicamente neste capítulo, são apresentadas a Metodologia de Projeto adotada, a Especificação de Requisitos do Projeto e a Descrição do Projeto de Software do protótipo de teste.

No Capítulo 6 será feita uma avaliação do protótipo apresentado no capítulo anterior. São descritas suas principais vantagens, limitações, restrições, e capacidades a futuras expansões; também é feita uma avaliação da facilidade de uso desta

interface, por um usuário leigo em computação, e em linguística; sendo ainda analisada neste capítulo, a capacidade de formalização computacional de um sub-conjunto da língua portuguesa, com alguns comentários sobre a perspectiva futura de aplicação na engenharia de Software.

Finalizando, serão apresentadas no Capítulo 7 as conclusões deste trabalho e considerações para futuras ampliações do projeto. Será feita também uma análise deste projeto com relação ao atual estado da arte.

1.2. Visão Geral

De uma maneira geral, as pessoas se comunicam falando ou escrevendo em "linguagens naturais", tais como Inglês, Francês, Alemão ou Protuguês. Por outro lado, as linguagens dos computadores são estranhas à maioria das pessoas. Estas "linguagens artificiais" foram projetadas de tal maneira que, apresentando um formato rígido quanto à sintaxe, facilitarão a tradução de seu significado nas instruções adequadas para o computador, mas tornando-as, em geral, menos acessíveis às pessoas.

Ora, se o computador entendesse o que o ser humano quer dizer ou escrever, os sistemas operacionais de modo geral, seriam muito mais fáceis de serem utilizados, e poderiam fazer parte, naturalmente, da vida de todas as pessoas.

O principal objetivo desta dissertação não é o de encontrar a chave deste problema, mas de conduzir em direção a uma capacidade maior de entendimento, por parte do computador, de nossa linguagem natural, em particular, o português. Dentro deste contexto será conduzido este trabalho.

Assim sendo, o principal objetivo do projeto é o estudo de um determinado método de compreensão formal, por parte do computador,

de um subconjunto da língua portuguesa. Para tanto, embora seja muito difícil abordar o tema como um todo, por ser demasiadamente amplo, escolheu-se um assunto mais restrito, em torno do qual este trabalho será elaborado.

O assunto em questão, é o interfaceamento em linguagem natural com um programa de aplicação, em particular, com uma base de dados, para o qual será desenvolvido um protótipo de teste, com o objetivo de exercitar os conceitos apresentados.

Convém também destacar que, no trabalho de processamento de linguagem natural devem ser destacados dois aspectos importantes: a **análise** e a **síntese** de textos.

No processo de análise a entrada do sistema, é um texto em linguagem natural e a sua principal saída, é uma representação que procura apresentar formalmente o significado do texto.

Ao contrário da análise, no processo de síntese, a principal entrada do sistema é uma representação da semântica de um determinado texto, tendo como principal saída um texto em alguma linguagem natural ou artificial.

Mais uma vez o tema é excessivamente amplo, e uma restrição se torna necessária, para viabilizar a pesquisa proposta dentro dos recursos disponíveis. Com este raciocínio, optou-se por escolher o campo da análise como tema central do trabalho sem no entanto, deixar de abordar alguns reflexos importantes do método desenvolvido no campo da síntese. Além disso, como a compreensão de textos gerais é um problema, que ocorre menos freqüentemente na comunicação entre o ser humano e o computador, quando do tratamento com base de dados, os textos particulares que serão estudados são frases isoladas.

Desta forma, existe a possibilidade de explorar toda formalização desenvolvida em diversos campos de aplicação da

computação, tais como a Engenharia de Software (Prototipação e Especificação de Requisitos).

Concluindo, este trabalho estará mais dedicado à formalização básica computacional da linguagem natural do que a problemas específicos de abrangência de seu dicionário. Dada uma pergunta em português, esta deverá ser analisada (léxica, sintática e semanticamente) e mapeada em uma forma lógica; forma esta que deverá ser, em seguida, transformada em um conjunto de comandos intelegíveis pelo computador. é importante também, que este sistema posssibilite ao usuário ampliar o dicionário de palavras para que o sistema possa entender.

2. LINGUAGEM NATURAL NA COMPUTAÇÃO

Recentemente tem-se discutido a utilização de linguagem natural em sistemas computacionais. Com isso torna-se importante discutir os diversos campos de aplicação e análise da real necessidade da linguagem natural na computação.

Primeiramente, é apresentado um levantamento das razões a favor e contra a utilização da linguagem natural na computação, como resultado de diversas pesquisas realizadas por muitos especialistas na área. O objetivo desta introdução, é mostrar a polêmica que ainda existe em diversos campos quanto ao uso da linguagem natural.

Dentre os fatores favoráveis ao uso da linguagem natural na computação, pode-se dizer, que um grande número de pessoas não é capaz de entender e usar uma linguagem artificial. Assim sendo, para algumas aplicações a linguagem natural constitui um meio ideal de comunicação com o ser humano.

Dentre estas diversas aplicações pode-se citar:

- aplicações com um grande número de usuários em potencial, e possivelmente, usuários com um bom conhecimento da aplicação, porém imaturos com relação à programação.

- sistemas com um grande número de entradas variadas e utilizados por um grande número de usuários. No caso de se ter um pequeno número de usuários poderia ser mais econômico treinar esse grupo em uma determinada linguagem artificial, do que desenvolver um sistema em linguagem natural. No entanto, em algumas aplicações, este pequeno grupo pode ainda relutar em utilizar uma linguagem artificial, motivando portanto o uso da linguagem natural.

. no atual estágio da tecnologia parece bastante viável a utilização de sistemas em linguagem natural. Neste aspecto convém apresentar uma breve descrição de sistemas existentes já desenvolvidos na área.

L SNLIS - Lunar Sciences Natural Language Information System [13], [29].

Este sistema foi desenvolvido por Willian Woods com o objetivo de pesquisar os problemas envolvidos na construção de uma interface, homem-máquina, que permitisse a comunicação em inglês. O LSNLIS contém informações sobre as rochas que foram coletadas durante a Missão Apolo à Lua e permite a geólogos acessar, comparar e avaliar os dados convenientemente.

Neste trabalho uma pergunta em inglês é traduzida numa expressão formal - "query language". Esta tradução é realizada através de um analisador ATN, explicado no capítulo 3 desta dissertação, acoplado a um interpretador semântico dirigido por regra. A forma "query-language" constitui-se numa generalização da lógica de predicados.

As três etapas em que este sistema foi desenvolvido são:

. análise sintática, utilizando o formalismo ATN

. interpretação semântica, com a finalidade de produzir uma representação do significado da sentença em uma linguagem formal

. execução da expressão formal sobre uma base de dados.

O dicionário deste sistema contém aproximadamente 3500 palavras.

Registrou-se, em uma conferência em 1971, que 78% das perguntas, feitas por geólogos, foram entendidas corretamente, 12% poderiam ser respondidas com algumas modificações no sistema, e 10% só poderiam ser respondidas com grandes modificações.

SHRDLU [29], [29]

Este sistema constitui-se na tese de doutorado de Terry Winograd no MIT e foi implementado na linguagem LISP.

Este programa simula a operação de um braço de um robô que manipula objetos tais como blocos, pirâmides, bolas e caixas, todos colocados sobre uma mesa. Este sistema trouxe uma importante contribuição ao desenvolvimento de sistemas de pergunta-resposta, demonstrando ser possível tratar simultaneamente a capacidade sintática, semântica, inferencial e gráfica em um único programa. Ele mantém um diálogo com o usuário, aceitando comandos e declarações, e respondendo a perguntas sobre o estado dos objetos na mesa. Sua implementação contém quatro elementos básicos: um analisador, um reconhecedor da gramática, um programa para análise semântica e um resolvedor de problemas que implementa as tarefas sobre os objetos. Uma das limitações do SHRDLU, também presente na maioria dos sistemas de processamento de linguagem natural, é a de não conseguir manipular as características mais complexas da lingua inglesa. Entre os pontos mais problemáticos citam-se: as concordâncias nominal e verbal, a manipulação de infomações condicionais e o tratamento de palavras tais como "the" e "and", as quais aumentam a complexidade semântica das sentenças a que pertencem.

REQUEST [13], [14]

É um sistema de pergunta-resposta que utiliza um suybconjunto restrito de lingua inglesa, podendo analisar e responder a uma diversidade de perguntas com grande complexidade sintática. Foi desenvolvido com base em gramáticas transformacionais.

É apresentada, a seguir, uma representação gráfica deste sistema:

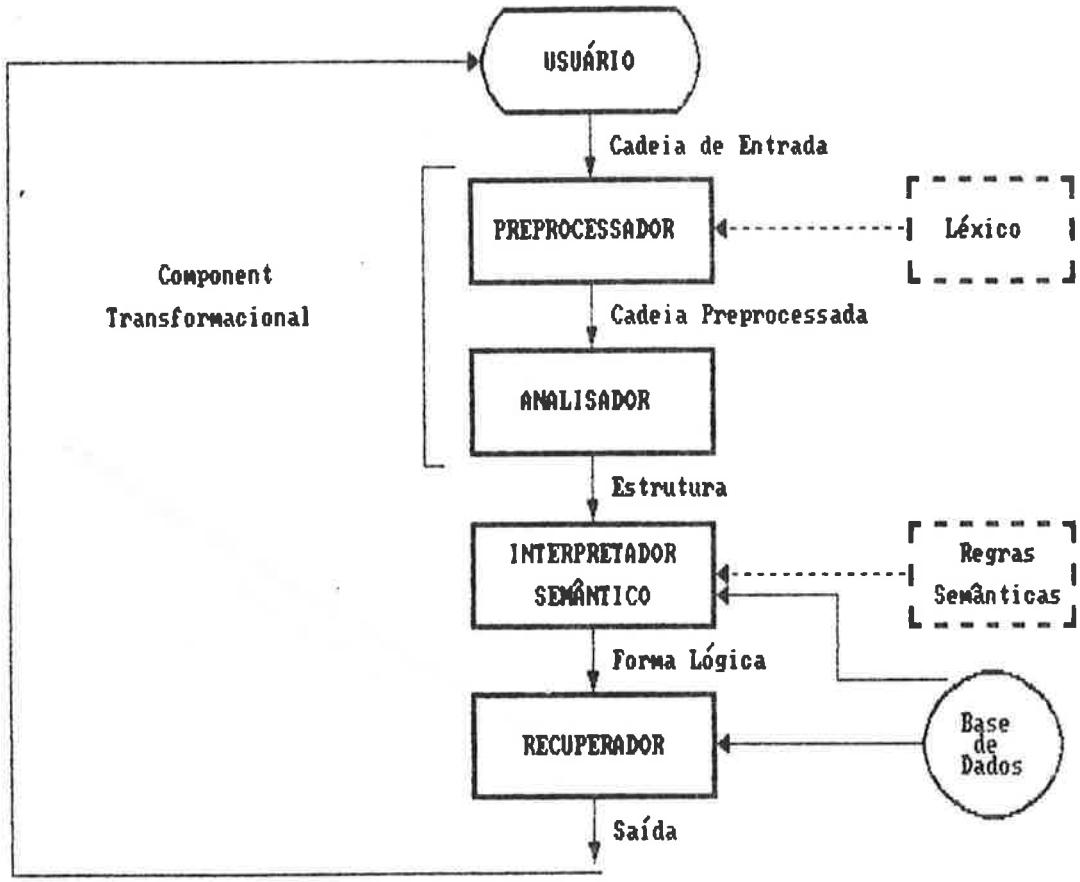


FIGURA 2.1 - DIAGRAMA DO REQUEST

Sempre que for detectada alguma existência de erros na cadeia de entrada, devido à presença de palavras escritas incorretamente ou não reconhecidas pelo sistema, ou então à existência de ambigüidades, o preprocessador interage com o usuário, solicitando-lhe esclarecimentos adicionais.

O sistema REQUEST, nesta versão, é capaz de manipular sentenças dos tipos interrogativa, negativa e com expressões com conotação temporal, não exibindo, entretanto, recursos para assimilar constuções lingüisticas tais como: comparações, uso de conjunções e de quantificadores.

A referência [13] cita ainda os sistemas REL e NLP.

O sistema REL - Rapidly Extensible Language foi o primeiro de pergunta-respostas que apresentava características de extensibilidade, quando novos termos poderiam ser definidos em inglês e subseqüentemente processados com sucesso pelo sistema. Devido a algumas restrições, parece utilizar um dialeto artificial do inglês.

O sistema NLP - Natural Language Processing foi usado para implementar o NLPQ, que foi aplicado num sistema de programação auotmática, descrito mais adiante neste capítulo. O sistema NLP não aceita frases subordinadas.

De maneira contrária, existem fatores que não favorecem o uso da linguagem natural em sistemas computacionais. Uma das características mais difícies de um problema é sua formulação precisa pois a linguagem natural é inerentemente vaga e ambígua. Assim sendo, seu uso pode conduzir a um processamento ineficiente, e provavelmente incorreto, devido a um mau entendimento. Este argumento é mais justificável quando a linguagem natural, for utilizada como linguagem de programação, e não como linguagem de interface (pergunta-resposta).

De maneira geral, uma interface em liunguagem natural é considerada mais adequada quando for orientada por problema, ou seja, que se apresente ao usuários como uma forma de exprimir os problemas a serem resolvidos, ao invés de indicar a forma como tal solução deve ser efetuada. A especificação do problema deve ser feita em uma linguagem aderente à particular aplicação a que se destina a interface.

Além deste fator convém destacar que o uso irrestrito da linguagem natural não é tecnicamente viável no momento. Conseqüentemente, um subconjunto da linguagem natural deve ser usado na comunicação com os computadores. Deve-se tomar máximo cuidado para que este subconjunto não se torne mais difícil de ser entendido do que uma linguagem artificial. Isto pode acontecer, devido aos hábitos e vícios de uso da linguagem natural.

2.1. Entendimento de Texto-Análise

Outro campo bastante importante da utilização computacional da linguagem natural, além da interface pergunta-resposta, constitue-se no Entendimento de Texto [18], [19]. No processo de Análise de um texto, em linguagem natural, é de grande importância o seu mapeamento numa representação interna adequada. Uma das representações internas da semântica de um texto mais conhecidas é a Dependência Conceitual. A Dependência Conceitual constitui-se numa estrutura que procura organizar o conhecimento humano e foi desenvolvida por Roger Schank [20], [29]. As primeiras pesquisas na área falam em direção à busca de um conjunto de primitivas, que pudessem ser utilizadas na representação estrutural de conceitos.

A título de curiosidade, convém citar que Aristóteles foi o primeiro a fazer um estudo sistemático dos conceitos, procurando descobrir suas propriedades, como podem ser unidos e separados, divididos e definidos, e como é possível extrair conceitos novos a partir de conceitos previamente conhecidos. Os resultados dessas

pesquisas de acordo com a referência [33] se encontram na obra de Aristóteles denominada "Organon".

Para Aristóteles todas as idéias podem ser reduzidas a dez grandes grupos, chamados categorias. As dez categorias são: substância, qualidade, quantidade, ação, paixão, relação, tempo, lugar, posição e hábito.

Existe uma grande similaridade entre a teoria desenvolvida por Aristóteles e os princípios da Dependencia Conceitual de Schank. Nesta última teoria foi desonvolvido um conjunto de categorias de AÇÃO para representar alguns conceitos:

- Categoria Instrumental
- Categoria Física
- Categoria Mental
- Categoria Global

Dentro de cada uma dessas categorias foi formalizado um conjunto básico de primitivas, que procura representar as ações humanas. A seguir é apresentado um resumo dessas primitivas. No Apêndice A deste trabalho é detalhada a Teoria da Dependência Conceitual.

Para a Categoria Instrumental tem-se as seguintes primitivas:

- ATTEND : ação que tem os orgãos dos sentidos como objeto e a localização física como direção.

- SPEAK : ação que produz sons e seus objetos são sempre sons.

A Categoria Física compõe-se das seguintes primitivas:

- PROPEL : ação que significa "aplicar uma força a" e seu objeto deve estar sob certas características de tamanho e peso.

- MOVE : ação que significa "mover uma parte do corpo" e os únicos objetos são as partes do corpo.

- INGEST : ação que significa "ingerir alguma coisa" e seu objeto deve ser menor do que a boca do ator, ou deve ser dividido em pedaços menores, que caibam na boca.

- EXPEL : ação que significa "tirar alguma coisa de dentro de voce e forçar para fora".

- GRASP : ação que significa "agarrar" e seu objeto deve estar dentro de um limite de tamanho.

Dentro da Categoria Mental devem estar as ações que representam os caminhos, através dos quais os pensamentos chegam à nossa mente. Nesta categoria tem-se as seguintes primitivas:

MTRANS : ação que representa uma mudança no controle mental, de uma conceituação e engloba verbos tais como relembrar, perceber e comunicar.

MBUILD: ação que significa combinação de pensamento e não inclue nenhuma informação do sucesso ou falha da operação.

PLAN: ação que tem como entrada um objetivo e um conjunto de planos possíveis que podem ser requeridos para atingir o determinado objetivo. Esta primitiva decide entre os planos possíveis, e produz como saída uma sequência de ações, que o ator tenciona realizar para conseguir o seu objetivo.

A Categoria Global exprime as ações que representam uma mudança de estado consequência de uma variável de ação física. Suas primitivas são:

PTRANS: ação que exprime a mudança na localização física de um objeto.

ATRANS: ação que opera sobre relações abstratas e seu instrumento físico raramente é especificado. Tem como objetivo a relação abstrata entre dois objetos reais. Esta relação pode ser dos tipos: Propriedade, Posse, Controle e outras.

Esta breve explicação da Dependência Conceitual, teve como principal objetivo mostrar a complexidade envolvida no processo de entendimento de texto. O número de variáveis e conceitos se multiplica de maneira bastante grande, para englobar um conjunto razoável de idéias.

2.2. Síntese de Texto

O processo de mapeamento de um texto em linguagem natural numa representação interna adequada, constitue-se no processo de Análise, como foi mostrado. No entanto, o processo de geração de sentenças em linguagem natural a partir de uma representação interna do significado, relacionando caminhos apropriados através da gramática, constitue-se na Síntese, também bastante importante como um dos campos da linguagem natural na computação.

O processo de Síntese em linguagem natural começa a partir de uma "deep structure", que representa a semântica do conhecimento, e continua através da aplicação de uma série de transformações organizadas, até que seja conseguida uma sentença numa forma, "surface structure", que corresponde à forma em que esta sentença se apresenta ao usuário.

Woods tem estudado a formalização destas transformações, representando a gramática inglesa na forma de uma Rede de Transição de Estados. Esta rede é estendida com subrotinas, através de subredes, e uma série de condições e operações

associadas com cada caminho possível. Este formalismo é discutido em mais detalhes no capítulo seguinte.

O algoritmo de Síntese é um processo simples, que modifica sucessivamente uma estrutura semântica, aplicando uma série de transformações funcionais, em uma determinada ordem estabelecida pela gramática.

A seguir são apresentadas algumas características obtidas de experiências na Síntese de sentenças em inglês controladas por semântica.

- Uma gramática isolada gera sentenças sintaticamente corretas, mas geralmente sem significado semântico;
- Redes Semânticas associadas ao dicionário léxico, geram sentenças inteligíveis, mas sintaticamente mal construídas;
- A Síntese de sentenças controladas na semântica requer que, a cada relação desta, corresponda uma ou mais relações sintáticas e as transformações associadas podem gerar, desta forma, diversas "surface structures";
- A gramática impõe uma ordem na aplicação das transformações;
- O primeiro passo no processo de Síntese, parece ser a seleção de um conjunto de determinantes em termos de valores, de forma, aspecto, tempo, etc...

2.3. Programação Automática

Outro campo de grande aplicação da linguagem natural além de interface com base de dados, entendimento de texto e geração de sentenças, é a Programação Automática.

A habilidade de programar em linguagem natural, em vez de usar uma linguagem de programação tradicional, possibilitaria o

uso de construções familiares na expressão de requisitos, tornando as máquinas acessíveis a um grupo maior de pessoas [10].

Existe um sistema desenvolvido na Universidade de Duke, Carolina do Norte, chamado "Natural Language Computer - NLC" que torna possível desempenhar uma programação limitada, em linguagem natural [2]. Este sistema possibilita uma pessoa observar suas estruturas de dados, e o processo computacional, à medida que os comandos são digitados. Esta forma de programação em linguagem natural, permite ao usuário examinar o desempenho do sistema à medida que os comandos são digitados, e detectar os erros imediatamente.

Este sistema apresenta algumas restrições para contornar os problemas já citados anteriormente, com relação à utilização da linguagem natural na computação. Estas restrições são:

- O usuário pode referenciar apenas estruturas de dados que aparecem na tela e especificar operações simples sobre elas - restrição semântica.

- O usuário deve começar cada sentença com um verbo imperativo - restrição sintática.

A programação em linguagem natural apresente, neste últimos anos, uma grande possibilidade de desenvolvimento, devido à evolução da teoria da representação do conhecimento, da teoria da inferência e da linguística computacional. O que os sistemas em programação automática procuram desenvolver, é um ambiente baseado em conhecimento, que possa entender uma declaração do usuário ou um procedimento, e convertê-los em um programa de computador. Como disse Balzer:

" A principal diferença entre a programação convencional e a automática, é que a última utiliza um modelo semântico do domínio, para estruturar o diálogo entre o sistema e o usuário, para entender as respostas do usuário e para traduzir as respostas deste em ações. "

A filosofia que está por trás destes sistemas, é a de suportar um diálogo em linguagem natural com o usuário, sobre seus requisitos e com isso produzir um programa apropriado para ele.

Existem alguns projetos na área de programaçào automática, que são citados a seguir: [17]

NPGS - Naval PostGraduate School (Monterey - Califórnia)

O principal objetivo deste grupo, foi desenvolver um sistema, que iria gerar um programa de simulação após um diálogo, em inglês, com o usuário sobre problemas de fila. Foi desenvolvida uma linguagem NLPQ, apoiada em uma linguagem já citada anteriormente, a NLP.

A medida que o diálogo se desenvolve, o sistema utiliza a informação obtida, para construir uma sub-rede chamada de "IPD - Internal Problem Description". O diálogo em ingles deve preencher todas as informações necessárias, para que o sistema possa construir uma IPD completa.

MIT

O objetivo deste projeto no MIT, que iniciou em 1972, era construir um sistema de programação automática para aplicações comerciais. A interação deste sistema com o usuário começava com um questionário, que lhe permitia responder com sentenças e com múltipla escolha. Após adquirir a descrição da aplicação do usuário, o sistema o conduz, a construir um diagrama em bloco

adequado ao problema, podendo testar este diagrama através de uma simulação. Finalmente, o diagrama em bloco é transformado num programa otimizado em PL/1.

IBM

O projeto de programação automática começou na IBM em 1974. O objetivo deste projeto, é permitir ao usuário, criar programas de aplicação comercial, através de um diálogo informal e interativo com o computador. Neste sistema ambos fazem declarações e respondem a perguntas. Pretende-se também, que o usuário possa requisitar uma execução simulada do programa, para explorar seu comportamento sob diversas condições. A estrutura de dados deste sistema, é uma rede semântica constituída por três partes: modelo de programa, modelo de aplicação e modelo lingüístico.

O modelo de programa fornece uma descrição abstrata dos programas parametrizados, incluindo informações sobre várias opções e usando conceitos definidos no banco de informações determinado BDS - "Business Definition System".

O modelo de aplicação fornece informações sobre conceitos relevantes de negócios. O modelo lingüístico tem informações sobre palavras em inglês e como são usadas. Cada objeto no modelo de programa, é ligado a um objeto de aplicação que se constitue numa explicação conceitual e cada objeto de aplicação é ligado a um objeto lingüístico, que é usado na comunicação em linguagem natural. O processamento da linguagem natural neste sistema foi feito numa versão expandida da NLP e implementado em Lisp.

Finalizando, uma das grandes dificuldades da linguagem natural na Engenharia de Software, é suportar um diálogo com o usuário sobre seus requisitos, e com isso produzir uma programa apropriado para ele. Na opinião do professor Turski, um dos

grandes pesquisadores na área, a Lógica constitue-se muito mais num processo de comunicação do que num processo de pensamento. Assim sendo, a lógica não tem o poder de evitar inconsistências na formação de uma Base de Conhecimento, mas constitue-se, juntamente com a lingüística, numa Teoria Formal de Especificação, ou seja:

Formalização de um Sistema =	Axiomas Lógicos	+	Nível Lingüistico

Dentro deste processo de comunicação, o grande problema ainda é a transformação da Visão Descritiva de um Sistema para a Visão Prescritiva ou Normativa. (figura 2.2)

A Visão Descritiva de uma Especificação de um Sistema é abstrata, livre e portanto "permissiva". Já a Visão Normativa é formal, concreta e matemática. A partir do ponto em que uma especificação se encontra dentro de uma Visão Normativa, é bastante possível a geração automática de seu programa, através de transformações lógicas.

No entanto, não se pode afirmar o mesmo com relação a Visão Descritiva de uma Especificação. Convém acrescentar ainda que esta Visão Descritiva constitue-se no primeiro passo natural na especificação de um Sistema, e a grande barreira da aplicação da Linguagem Natural na Engenharia de Software.

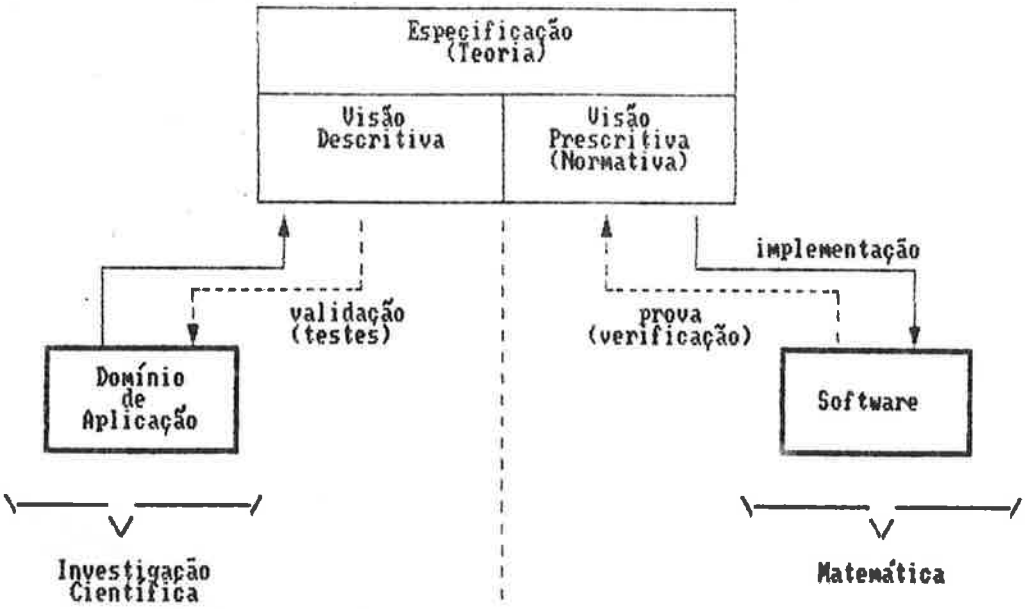


FIGURA 2.2 - VISÕES DA ESPECIFICAÇÃO DE UM SISTEMA (TURSKI)

3. FORMALISMOS PARA PROCESSAMENTO DE LINGUAGEM NATURAL

O caminho mais comum para se formalizar uma linguagem, seja natural ou artificial, é através de uma coleção de regras chamada gramática. Regras estas, que determinam cadeias de símbolos ou palavras, consideradas sentenças válidas de uma linguagem. Além disso, a gramática geralmente proporciona um tipo de análise de sentença, dentro de uma estrutura que torna mais explícito seu significado.

A seguir são descritos alguns formalismos básicos que podem ser utilizados na especificação formal de um subconjunto (gramática) de uma linguagem natural. Um dos objetivos lingüísticos de uma gramática de linguagem natural, é capturar as regularidades da linguagem.

3.1. A Evolução da Gramática Livre de Contexto para a Gramática Transformacional

Chomsty classificou as gramáticas em quatro classes, em função de restrições impostas às leis de formação que a compõem (produções). A primeira classe é uma gramática cujas regras são irrestritas, conhecida como Gramática Tipo 0. A Gramática Tipo 1 corresponde à Gramática dependente de Contexto. A Gramática Tipo 2 é mais conhecida como Gramática Livre de Contexto. A última classe é a Gramática Regular - Tipo 3 - onde as produções têm uma forma bastante restritiva [37].

A definição de sintaxe das linguagens de programação, é geralmente dividida em duas partes. A primeira parte da sintaxe é especificada na forma de uma Gramática Livre de Contexto, que pode ser eficientemente analisada pelo computador. A outra parte, consiste de restrições dependentes de contexto. Essas duas partes descrevem em conjunto a estrutura de compiladores práticos.

A Gramática Livre de Contexto é importantíssima na descrição e tradução de linguagem de programação, pois constitue-se no formalismo mais poderoso para o qual existem algorítmos de análise bastante eficientes. Este tipo de gramática, pode especificar a maioria das sintaxes de linguagem de computador, visto que em sua maioria são simples do ponto de vista estrutural. As Gramáticas Livres de Contexto geram linguagens livres de contexto.

O modelo da Gramática Transformacional [31] consiste de uma Gramática Livre de Contexto, (base) associada a um conjunto de regras de transformação, que mapeiam árvores sintáticas em novas árvores sintáticas (derivadas) semânticamente equivalentes. A geração de uma sentença através do uso de tal tipo de gramáticas, consiste primeiramente, em se construir, a partir de uma representação semântica completa da sentença ("deep structure"), e por meio da aplicação de transformações sucessivas, uma nova representação ("deep structure") semânticamente equivalente à primeira, e cujos nós terminais (folhas) representam a denotação final da sentença gerada [21].

Este modelo da transformacional, é mais adequado à geração de sentenças (síntese) do que à sua análise, embora seja possível obter um algorítmo próprio para análisar sentenças com base nesta gramática. Este algorítmo consiste em aplicar transformações inversas, com a finalidade de buscar a sentença em análise entre todas as possíveis sentenças da linguagem, geradas pelo algoritmo.

As transformações inversas não são seletivas como as transformações diretas e muitas aplicações espúrias de transformações inversas são possíveis. Enquanto a maioria das transformações diretas, irá conduzir a estruturas ("surface structure") verdadeiras, a maioria das trasnformações inversas não irá conduzir a estruturas ("deep structures") legítimas, e um grande esforço adicional será gasto em caminhos perdidos. Existem pesquisadores que evitam esse não-determinismo nas transformações inversas, construindo um conjunto determinítico de

transformações inversas a uma determinada gramática. Este método, entretanto, não garante a produção de "deep structures" legítimas da sentença.

3.2. Rede de Transição Estendida - ATN ("Augumented Transition Networks"

Um dos primeiros modelos para a gramática de linguagem natural, foi o grafo de transição de estados finitos. No entanto, tal modelo mostra-se inadequado para representar estas gramáticas devido à não capacidade de representar muitas de suas características. Uma de suas maiores falhas é a ausência do mecanismo de pilha, que permite suspender o processamento em um determinado nível, enquanto utiliza a mesma gramática para processar um outro nível.

Considere que se tenha uma coleção de grafos de transição em que se permita rótular os arcos não apenas com símbolos terminais mas também símbolos não-terminais, representando construções complexas, que se façam presentes para que a transição aconteça. A determinação se tais construções estão presentes em uma sentença, será feita por uma "chamada de subrotina" a um outro grafo de transição (ou o mesmo). O modelo resultante é uma "Rede de Transição Recursiva" (RTR). Seu poder de geração é equivalente à gramática livre de contexto, ou a um autômato de pilha. Mas ele apresenta também, maior eficiência de expressão do algorítmo de análise, e uma extensão natural que permite vários graus de dependência de contexto e a construção de estruturas flexíveis durante a análise.

A RTR é um grafo com rótulos nos estados e nos arcos, um estado distinto chamado estado inicial e um conjunto de estados chamados estados finais. Assemelha-se a uma diagrama de transição de estados finito não determinístico, exceto que os rótulos nos arcos podem ser nomes de estado como também símbolos terminais.

Descreva-se a seguir um modelo de gramática baseado na notação da RTR e que supera algumas das dificuldades exibidas pelo modelo transformacional.

Pode-se adicionar recursos ao modelo de rede de transição, tais como: a possibilidade de mover fragmentos da estrutura da sentença, copiar e remover fragmentos da estrutura da sentença e realizar ações em seus constituintes, dependendo do contexto em que ocorrem. Tais ações são implementadas, adicionando a cada arco da transição uma condição arbitrária, que deve ser satisfeita para que o arco possa ser percorrido. Este novo modelo é conhecido como Rede de Transição Estendida - ATN.

A rede de transição estendida - ATN - [27], [28], constitui-se numa rede composta por um ou mais "conjunto de arcos". Cada "conjunto de arcos" é formado por um "estado" e alguns "arcos" (se existirem) que saem deste "estado". Cada "arco" pode apresentar quatro tipos: CAT, PUSH, TST e POP.

Quando o tipo for CAT, o arco pode ser percorrido se o símbolo da entrada for um membro da categoria léxica indicada no arco e se um determinado "teste" for satisfeito.

No caso do tipo PUSH, o arco causa um desvio para o estado indicado desde que um "teste" seja satisfeito. Neste caso é retornada à rede principal a forma da estrutura manipulada.

Quando o tipo for TST, o arco permite um teste arbitrário para determinar se ele pode ser percorrido.

Nos três casos anteriores em que os arcos são do tipo CAT, PUSH ou TST, depois que foram satisfeitas as condições para eles serem percorridos, é executado um conjunto de "ações" que corresponde à construção de estruturas.

No caso de um arco do tipo POP, indica sob quais condições o estado pode ser considerado um estado final e qual a estrutura que deve ser retornada.

É apresentado no Apêndice B uma descrição detalhada do formalismo ATN.

3.3. Gramática de Cláusulas Definidas - DCG [12], [30]

O formalismo de uma gramática livre de contexto (CFG) expresso em predicados lógicos ou "Cláusulas de Horn" e executado como um programa em linguagem Prolog, constitue-se na chamada gramática de cláusulas definidas, mais conhecida por DCG. Portanto, o DCG constitue-se numa extensão natural da CFG.

É conhecido que CFG não é totalmente adequada para descrever a linguagem natural. O formalismo DCG supera essas dificuldades estendendo a CFG em três importantes aspectos:

a) O DCG possibilita definir dependência de contexto na gramática, de maneira que, dependendo da sua situação no texto, uma frase possa assumir diferentes interpretações;

b) O DCG permite que estruturas em árvore, sejam construídas durante a análise, não se limitando pela estrutura recursiva da gramática; estas estruturas em árvore podem fornecer uma representação do significado da cadeia;

c) O DCG permite que decisões extras sejam incluídas nas regras da gramática.

Desta forma, o formalismo DCG estende a CFG incrementando os não-terminais com argumentos.

3.3.1 Descrição Geral do Formalismo - DCG

A idéia básica deste formalismo, é como expressar gramáticas em lógica, ou seja, em cláusulas definidas.

Cada regra em CFG é expressa da seguinte forma:

nt -----> corpo;

onde nt corresponde a um não-terminal que se deseja definir, e o corpo é uma seqüência de um ou mais ítens separados por vírgula. Cada ítem, ou é um não terminal ou uma seqüência de símbolos terminais. O significado da regra corresponde a dizer que o corpo é uma das formas possíveis para uma frase do tipo nt.

Para mostrar a tradução do CFG para o formalismo DCG, é apresentado um exemplo em CFG, que consegue analisar sentenças do tipo "John loves Mary" e "Every man that lives loves a woman".

```
sentence -----> noun-phrase.
noun-phrase --> determinar, noun, rel-clause.
noun-phrase --> name.
verb-phrase --> trans-verb, noun-phrase.
verb-phrase --> intrans-verb.
rel-clause ---> [that], verb-phrase.
rel-clause ---> [ ].
determinar ---> [every].
determinar ---> [a].
noun ---------> [man].
noun ---------> [woman].
name ---------> [John].
name ---------> [Mary].
trans-verb ---> [loves].
intrans-verb -> [lives].
```

Para realizar a tradução para o DCG, é associado a cada não terminal um predicado, com o mesmo nome do não-terminal, dentro da cadeia.

Com isso tem-se:

```
sentence (S0, S)    : - noun-phrase (S0, S1), verb-phrase (S1, S).
noun-phrase (S0,  S): -  determinar (S0,  S1), noun (S1, S2), rel-
                         clause (S2, S).
noun-phrase (S0, S): - name (S0, S).
verb-phrase (S0, S): - trans-verb (S0, S1), noun-phrase (S1, S).
verb-phrase (S0, S): - intrans-verb (S0, S).
rel-clause   (S0, S  : - connects (S0, that, S1), verb-phrase (S1,
                         S).
rel-clause  (S0, S).
determiner (S0, S) : - connects (S0, every, S).
determiner (S0, S) : - connects (S0, a, S).
noun (S0, S)       : - connects (S0, man, S).
noun (S0, S)       : - connects (S0, woman, S).
name (S0, S)       : - connects (S0, John, S).
name (S0, S)       : - connects (S0, Mary, S).
trans-verb (S0, S) : - connects (S0, loves, S).
intrans-verb (S0,S): - connects (S0, lives, S).
```

O predicado "connects (S0, John, S)" indica que a constante 'John' une o ponto S0 ao ponto S.

No entanto, pode-se otimizar este formalismo. No lugar de um inteiro indicando a posição na cadeia, colocamos uma lista de símbolos que ocorrem após esta posição.

Assim sendo temos, por exemplo:

```
rel-clause ([that|S1],S): - verb-phrase (S1,S) e
determinar ([every|S],S).
```

Para questionar se a frase "Every man that lives loves Mary" é reconhecida por este formalismo, deve-se utilizar o seguinte predicado:

? - sentence ([every, man, that, lives, loves, Mary], [ ]).

A resposta será: true.

Para se obter este formalismo DCG, a gramática livre de contexto é generalizada de maneira a manter a correspondência com as cláusulas definidas.

A notação para o DCG é uma extensão da notação da CFG nos seguintes aspectos:

(a) não-terminais podem ser termos compostos:
sentence (S), np (X, S)

(b) Do lado direito de uma regra pode existir uma sequência de chamadas de procedimento, escritas entre chaves ({ }) para exprimir algumas condições extras, que devem ser satisfeitas para que a regra ser válida.

Por exemplo:

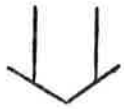
noun (N) --> [W], {rootform (W, N), is-noun (N)}

Ou seja, W é um substantivo se estiver sob uma raiz N que é substantivo.

Em cláusulas definidas, não-terminais, terminais e chamadas de procedimento do lado direito da regra são referenciadas como "goals".

Com isso em DCG teríamos a seguinte correspondência:

```
noun (N) --> [W], {rootform (W, N), is-noun (N)} <--> CFG
                                                     estendida
                                                       (DCG)
```



```
noun (N,S0,S): --> connects (S0,W,S), rootform (W,N), is-noun (N)

            ou                                                  DCG em
                                                                Prolog

noun (N, [W|S], S): --> rootform (W, N), is-noun (N)
```

Apresentado o formalismo do DCG, é discutido agora como este formalismo manipula três características importantes para a análise da linguagem natural:

- construção de estruturas (tais como árvores de análise);
- condições extras para os constituintes de uma frase;
- dependência de contexto.

a) Construção de Estruturas

A gramática livre de contexto é modificada para produzir explicitamente, para cada frase, uma interpretação que é simplesmente sua árvore de análise.

Em geral, em vez de se ter uma regra na forma:

```
category (argumento) --> [word].
```

para cada palavra dentro da classe **categoria**, é reescrita:

```
category (argumento) --> [W], {cat (W, argumento)};
```

Obs.: Os argumentos podem ser utilizados para análise léxica (por exemplo, singular e plural).

Observou-se também que definiu-se um "procedimento de dicionário" cat que consiste de cláusulas da forma:

cat (word, argumento), para cada palavra dentro da categoria cat.

Com estas medidas, o conjunto de regras se modifica para:

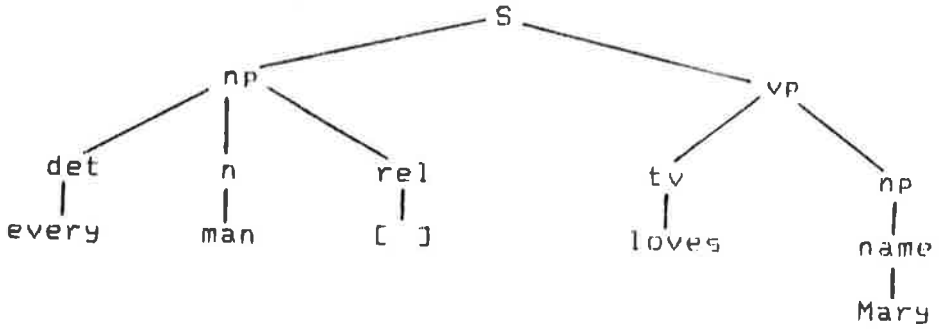
```
sentence (s (NP, VP)) --> noun-phrase(NP), verb-phrase (VP). (I)

noun-phrase (np  (Det, Noun, Rel)) --> determiner  (Det),  noun
                                        (Noun), rel-clause(Rel).
noun-phrase (np (Name)) --> name (Name).
verb-phrase (vp (TV, NP)) --> trans-verb (TV), noun-phrase(NP).
verb-phrase (vp(IV)) --> intrans-verb(IV).
rel-clause (rel(that, VP)) --> [that], verb-phrase (VP).
rel-clause (rel (nil)) --> [ ].
determinar (det (W)) --> [W], {is-determiner (W)}.
noun (n(W)) --> [W], {is-noun (W)}.
name (name(W)) --> [W], {is-name(W)}.
trans-verb (tv(W)) --> [W], {is-trans(W)}.
intrans-verb (iv(W)) --> [W], {is-intrans (W)}.
```

O significado da regra (I) é: "Uma sentença com interpretação s(NP, VP) pode consistir de uma frase substantiva com interpretação NP seguida de uma frase verbal com interpretação VP".

No dicionário teríamos, por exemplo:

```
is-determiner (every).
is-noun (man).
is-intrans (lives);
```

Assim, a árvore de análise da sentença "Every man loves Mary" seria:



b) Condições Extras para os Constituintes de uma Frase

Estas condições extras constituem-se em chamadas explícitas a procedimentos dentro do corpo da regra.

Por exemplo:

```
det (D,M) --> month(M), [D], {interger(D), 0<D<32}.
```

Uma frase representando o dia D do Mês M pode ser como uma frase representando o mês M, seguido de um símbolo D, onde D é um inteiro maior do que zero e menor do que 32.

c) Dependência de Contexto

Os argumentos dos não-terminais no formalismo DCG podem ser utilizados não só para construir estruturas (já foi mostrado anteriormente), como também para representar e testar alguma informação contextual.

Por exemplo, pode-se modificar o conjunto de regras, anteriormente apresentado, de forma a permitir a manipulação da concordância com relação ao número (singular ou plural) entre os componentes de uma frase.

As regras modificadas seriam:

```
sentence (s(NP, VP) --> noun-phrase (N,NP), verb-phrase(N,VP).
noun-phrase (N, np(Det, Noun, Rel)) -->
determiner (N, Det), noun (N,Noun), rel-clause (N, Rel).
noun-phrase (singular, np(Name)) --> name (Name).
verb-phrase (N,  vp(TV, NP))  --> trans-verb  (N,TV),  noun-phrase
                                  (N1,NP).
verb-phrase (N, vp(IV)) --> intrans-verb (N, IV).
rel-clause (N, rel(that, VP)) --> [that], verb-phrase (N, VP).
rel-clause (N, rel(nil)) --> [ ].
determiner (N, det (W)) --> [W], {is-determiner (W,N)}
determiner (plural, det(nil)) --> [ ].
noun (N, n(Root)) --> [W], {is-noun (W, N, Root)}.
name (name(W)) --> [W], {is-name (W)}.
trans-verb (N, tv (Root)) --> [W], {is-trans (W, N, Root)}.
intrans-verb (N, iv (Root)) --> [W], {is-intrans (W, N, Root)}
```

Abaixo são apresentados alguns exemplos de cláusulas associadas ao dicionário:

```
is-determiner (every, singular).
is-determiner (all, plural).
is-noun (man, singular, man).
is-noun (men, plural, man).
is-name (Mary).
is-trans (likes, singular, like).
is-trans (like, plural, like).
is-intrans (live, plural, live).
```

3.3.2. A função da variável Lógica no DCG

A variável lógica torna o formalismo DCG bastante potente na análise de linguagens naturais. Com este recurso, estruturas podem ser construídas deixando algumas partes, ainda não especificadas, como variaveis. Quando as informações necessárias

tornarem-se disponíveis, então estas lacunas nas estruturas, representados por variáveis lógicas, são preenchidos pelo processo de **unificação**. O processo de unificação consiste numa aderência do "pattern-matching" com a "variável lógica".

3.3.3 Razões pela opção do DCG

O formalismo adotado para desenvolver este trabalho foi o DCG, devido a diversos fatores que incluem, evidentemente, o fato de o autor ter uma razoável familiaridade com a linguagem PROLOG, na qual este formalismo é expresso mais naturalmente. Alguns destes fatores são a seguir descritos [25], [12].

Em geral, sistemas práticos para a análise de linguagens naturais são necessariamente grandes e complexos, e a tarefa de escrevê-los é também uma atividade experimental. A flexibilidade, característica desejável em qualquer formalismo, é particularmente importante aqui, ou seja, quanto tempo é gasto para se modificar e estender um sistema.

Dentro deste aspecto, acredita-se que o formalismo DCG apresenta grande vantagem sobre o ATN. Como o DCG pode ser entendido como uma máquina à análise de uma linguagem, em particular. Entrentanto, diferentemente do ATN, o DCG pode ser entendido como uma **descrição** da linguagem. O DCG partilha desta propriedade com o formalismo das Gramáticas Livres de Contexto - CFG.

A proximidade do relacionamento entre o DCG e a linguagem que ele descreve, pode ser formalmente explicitada em termos de semântica declarativa das cláusulas definidas. A semântica declarativa, permite a identificação de um conjunto de cláusulas definidas com um conjunto de "termos-verdadeiros". Cada um destes termos especifica que uma certa frase da liguagem em questão ocorre entre certos pontos em uma determinada cadeia. O conjunto de termos verdadeiros enumera todas as ocorrências possíveis de

frases da linguagem. Observa-se que em nenhum momento, a explicação envolve a execução do DCG. No entanto, para explicar formalmente como o ATN define uma linguagem, é necessário envolver a noção de como um ATN é executado, conforme foi mostrado no item 3.2 e detalhado no Apêndice B.

Conceitualmente, o ATN não é mais do que um mecanismo particular, para analisar uma linguagem de maneira "top-down", da esquerda para a direita e a seqüência imposta por esta estratégia de análise, está implicita na maneira pela qual os registros são operados. No entanto, o DCG pode ser executado por diversos caminhos.

Outra razão bastante importante pela opção do DCG é sua modularidade. O mecanismo do DCG é composto por pequenos componentes (cláusulas), que se comunicam apenas através de passagem de argumentos explícitos. Não há variáveis globais, pois o escopo de cada variável é limitado a uma única cláusula. Como resultado, o comportamento de cada cláusula no DCG pode ser entendido inpendentemente da outra. Por sua vez, no ATN a menor unidade que pode ser isolada é uma subrede, ou seja, uma parte de uma rede não conectada ao restante exceto via operações tipo "PUSH" ou "POP". Nenhuma unidade menor pode ser isolada, pois o escopo do registro é uma subrede.

Outro fator importante do DCG é a utilização do "pattern-marching" em vez de testes explícitos e comandos especificos de construção de estruturas.

Um critério de comparação bastante adotado, é a concisão, ou seja, a melhor gramática é aquela que caracteriza a linguagem com o menor número de símbolos. Comparando as formas textuais equivalentes do DCG e do ATN, conclue-se geralmente, que o DCG é significativamente menor. Assim sendo, o tamanho de um programa em DCG corresponde aproximadamente à metade de um programa equivalente em ATN.

Outra vantagem bastante característica do DCG em relação ao ATN, é quanto à eficiência. Enquanto o DCG é um programa em linguagem PROLOG, não necessitando um interpretador ou compilador especial, o mesmo não ocorre com o ATN.

4. DESCRIÇÃO DETALHADA DO FORMALISMO ADOTADO [11], [36]

Neste capítulo é descrito o formalismo XG e também é realizada uma execução simbólica de tal formalismo, visando mostrar seu poder de análise.

A gramática de extraposição (XG) é uma extensão da gramática DCG, e é definida de modo semelhante em termos de cláusulas lógicas. A extensão do DCG, através da inclusão deste novo formalismo, torna fácil descrever a extraposição à esquerda dos componentes da frase. Isto é muito conveniente para que seja possível descrever com precisão e clareza certos aspectos de uma linguagem natural, tais como as sentenças relativas e interrogativas [11], em linguagens como o Inglês, Francês, Espanhol e Português.

A extraposição à esquerda constitue-se numa lista onde são colocados componentes da frase para serem analisados posteriormente. Na realidade, estes componentes formam as frases relativas, que devem sempre estar aninhadas para manter a coerência sintática e semântica da frase. À medida que as frases relativas são analisadas, no sentido da mais interna para a mais externa, seus elementos são retirados desta lista de extraposição, conforme exemplificado no ítem 4.2.2 e no Apêndice C desta dissertação.

4.1 Descrição Informal do XG

Inicialmente é analisado um exemplo de XG, no qual se evita, por razões didáticas, a complicação adicional do uso de argumento nos não-terminais.

Na discussão que se segue o XG é visto como uma extensão da gramática livre de contexto e aplicado a um subconjunto da gramática inglêsa.

```
sentence     --> noun-phrase, verb-phrase.
noun-phrase  --> determiner, noun, relative.
noun-phrase  --> trace.
verb-phrase  --> trans-verb, noun-phrase.
verb-phrase  --> intrans-verb.
relative     --> [ ].
relative     --> rel-marker, sentence
rel-marker...trace --> rel-pronoun.
```

Todas as regras acima, exceto a última, são livres de contexto. A última regra expressa a extraposição em sentenças relativas. Ela esclarece que um pronome relativo deve ser analisado como sendo um "rel-marker", seguido por elementos que são irrelevantes para esta regra (denotados por '...'), seguidos por um "trace".

Por exemplo:

The boy rel-marker the girl kissed trace laugh.

The boy rel-pronoun [the girl kissed] laugh.

Neste caso o "trace" é utilizado como parte da análise do "rel-marker", ou seja, o elemento "rel-marker", que representa o pronome relativo na frase, indica que deve existir um outro componente à sua direita, representado pelo "trace", de forma que o elemento em análise possa ser interpretado como sendo um pronome relativo dentro de uma frase relativa corretamente aninhada na sentença principal.

4.2 Descrição Formal do XG

Uma regra tem a seguinte forma:

S1...S2 etc. SK-1...SK --> r. (1)

onde cada segmento Si é uma seqüência de terminais e não-terminais, denotados em DCG.

O primeiro símbolo em S1, o **símbolo-líder**, está restrito a ser um não-terminal.

O significado de regra (1) é que qualquer sequência de símbolos da forma:

S1 X1 S2 X2 etc... SK-1 XK-1 SK, com Xis arbitrários, pode ser reescrita como sendo:

r X1 X2... XK-1.

Desta forma, pode-se dizer que um não-terminal pode ser expandido através da regra (1), desde que haja concordância com o símbolo-líder, do lado esquerdo da regra, e concordância em seqüência com os demais simbolos Si.

Esta seqüência de símbolos pode ser interrompida por uma cadeia arbitrária, representada pela ocorrência do símbolo "..." no lado esquerdo da regra.

Voltando a analisar a seguinte regra:

rel-marker... trace --> rel-pronoun. (2)

e sua aplicação, tem-se:

rel-marker john likes trace, produz como resultando:

rel-pronoun [john likes].

↑ ↑

"limitação dos parênteses" - regras de aninhamento

Após a aplicação da regra, não é possível aplicar qualquer outra regra, onde haja uma intersecção de segmentos, dentro da região dos parênteses, com outro fora da região dos parênteses.

O uso da regra (2) dividiu a cadeia em duas porções isoladas, sendo que cada uma delas deve ser independentemente expandida. Este processo é conhecido como "bracketing constraint" ou "limitação dos parenteses".

4.2.1 Grafos de Derivação

Da mesma maneira que árvores de análise, definidas no item 3.3.1, são utilizadas para visualizar derivações livres de contexto, grafos de derivação são utilizados para representar derivações ou regras em XG.

A topologia dos grafos de derivação reflete claramente a característica da limitação dos parênteses conhecida como "bracketing constraint".

Na construção de um grafo de derivação devem ser assumidas algumas convenções importantes:

(a) As setas que chegam em um nó devem ser ordenadas no sentido horário, seguindo a seqüência correspondente de símbolos do lado esquerdo da regra, que este nó representa;

(b) As setas que partem, saindo de um nó devem ser ordenadas no sentido anti-horário, seguindo a seqüência correspondente de símbolos do lado direito da regra, que este nó representa.

Assim sendo, o grafo de derivação obedece à regra de aninhamento se e somente se ele pode ser construído, seguindo suas convenções e sem nenhum cruzamento de setas.

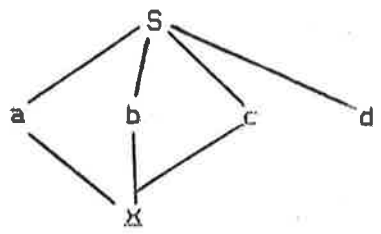
Por exemplo,

S --> a, b, c, d

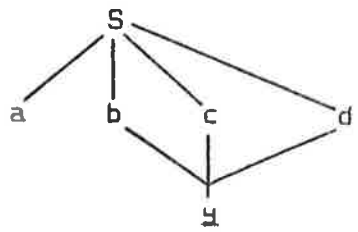
a...c --> [x]

b...d --> [y]

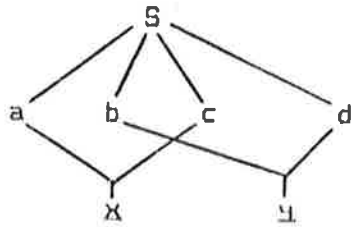
permite obter um dos grafos seguintes:



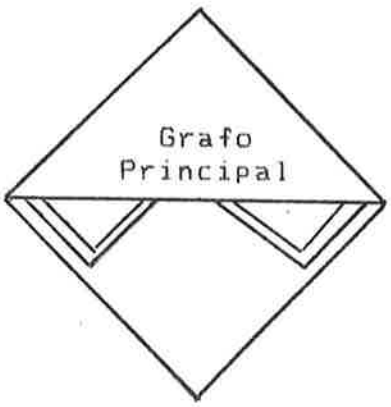
ou



Não é possível obter, neste caso, o seguinte grafo:



Pode-se observar que, no caso acima, não é possível a aplicação de regras de aninhamento, ou seja, respeitar a limitação dos parênteses. No caso em que estas regras forem observadas, o aspecto do grafo será o seguinte:



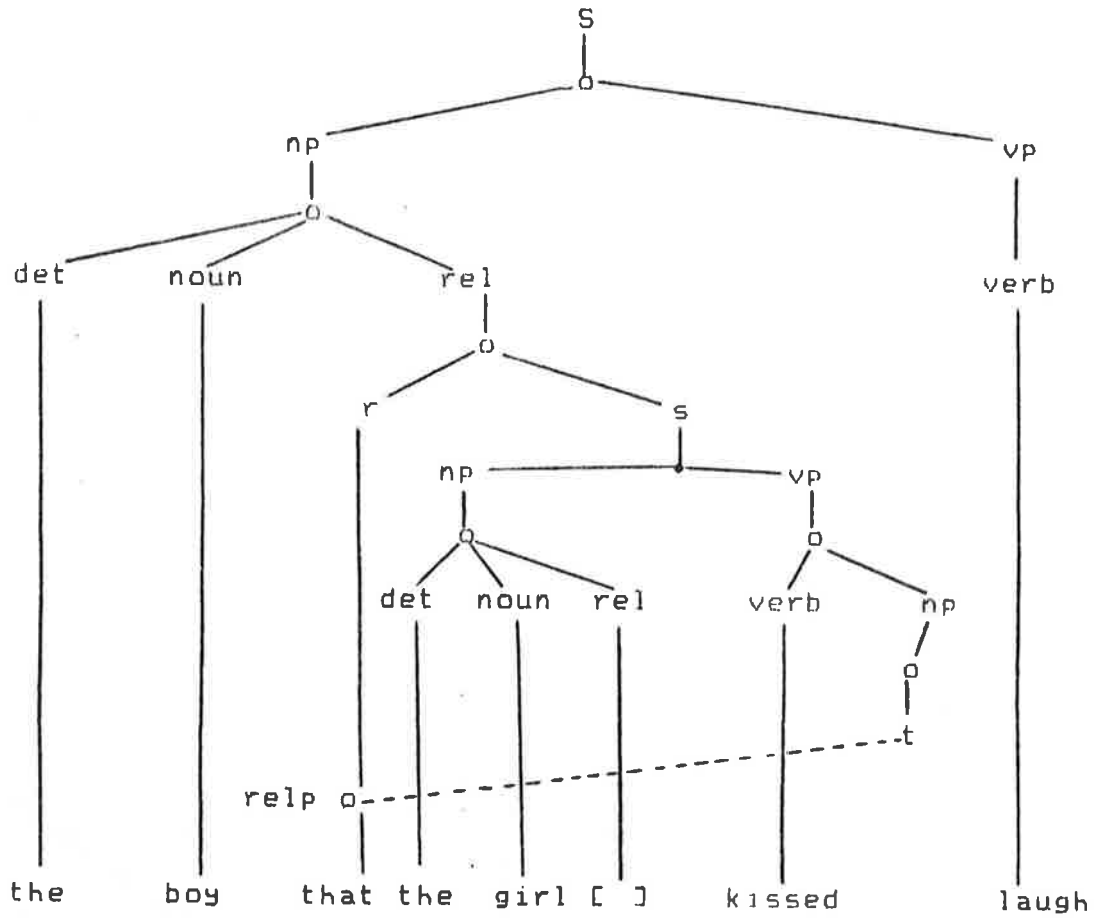
4.2.2 A Gramática XG e a Extraposição à Esquerda

A seguir é apresentado formalmente como a gramática XG manipula a extraposição à esquerda.

É apresentado a seguir um exemplo razoavelmente complexo para demonstrar tal propriedade:

```
sentence    --> noun-phrase, verb-phrase.
noun-phrase --> proper-noun.
noun-phrase --> determiner, noun, relative.
noun-phrase --> determiner, noun, prep-phrase.
noun-phrase --> trace.
verb-phrase --> verb, noun-phrase
verb-phrase --> verb.

relative --> [ ]
relative --> rel-marker, sentence.                                  (3)
rel-marker...trace --> rel-pronoun.

prep-phrase --> preposition, noun-phrase.
```

Aplicando tal formalismo à sentença "The boy that the girl kissed laugh" obtemos o seguinte grafo de derivação:



Legenda:

det : determiner
np : noun-phrase
r : rel-marker
rel : relative
relp: rel-pronoun
s : sentence
t : trace
vp : verb-phrase

Para incluir a característica de "limitação dos parênteses" e permitir apenas construções aninhadas em frases relativas precisa-se fazer algumas adaptações ao formalismo anteriormente apresentado.

A regra (3) "relative --> rel-marker, sentence". deve ser modificada para:

relative --> open, rel-marker, sentence, close. (4)

e deve ser adicionada a seguinte regra

open...close --> [ ]. (5)

Com esta modificação na gramática, não é mais possível violar as regras de aninhamento, pois nenhum constituinte da frase pode ser reposicionado de fora para dentro do 'gap'criado, pela aplicação da regra (5) ao resultado da aplicação da regra (4) em frases relativas.

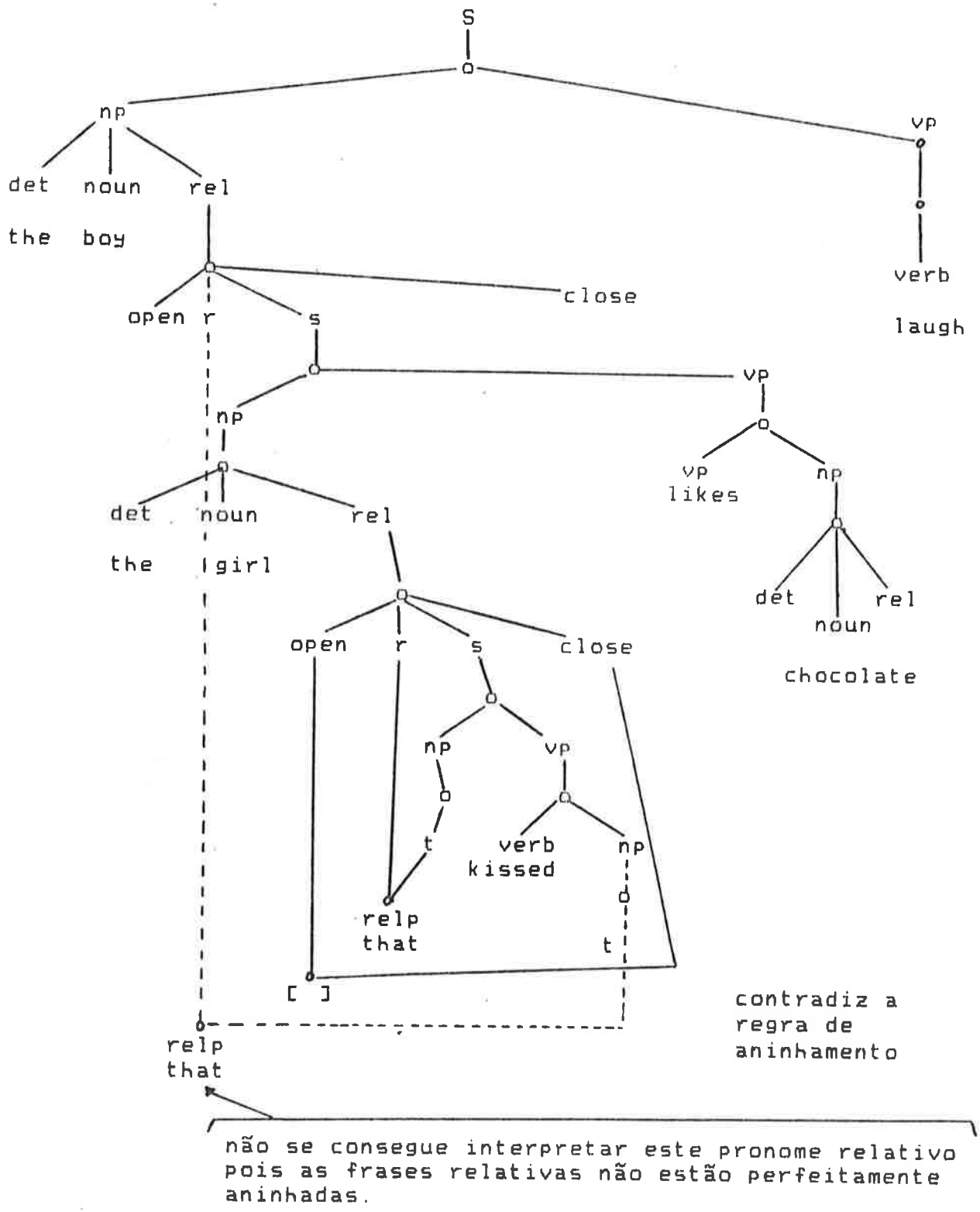
A seguir é apresentado um exemplo no qual a frase a ser analisada não está estruturada de uma forma correta, e tal característica é reconhecida por este formalismo ora apresentado.

Vamos considerar a seguinte frase incorretamente construida:

The boy that the girl that kissed likes chocolate laugh.

Se analisarmos com cuidado esta frase observamos que ela não apresenta uma estrutura correta quanto ao posicionamento das frase relativas. A posição correta do termo "kissed" está indicada pela seta pontilhada.

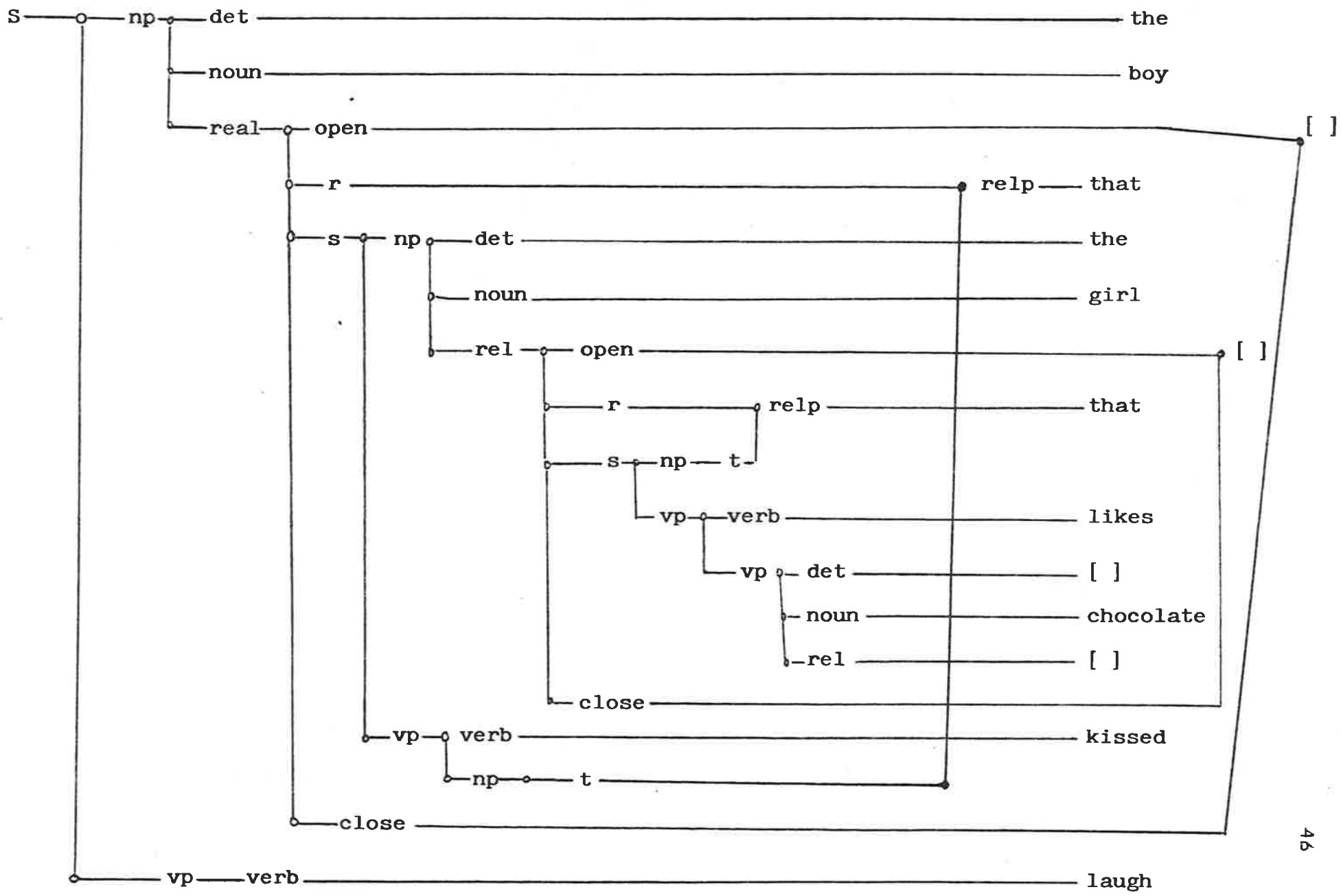
Seu grafo de derivação é apresentado de acordo com o formalismo explicado e verifica-se a inconsistência da frase devido ao não atendimento das regras de aninhamento.


S
np
vp
det
noun
rel
the
boy
verb
laugh
open
r
s
close
vp
np
vp
likes
np
det
noun
rel
the
girl
det
rel
noun
chocolate
open
r
s
close
np
vp
t
verb
kissed
np
relp
that
t
[ ]
contradiz a
regra de
aninhamento
relp
that
não se consegue interpretar este pronome relativo
pois as frases relativas não estão perfeitamente
aninhadas.

No entanto, a frase: "The boy that the girl that likes chocolate kissed laugh", está correta em relação à regra de aninhamento, ou seja, as frases relativas estão aninhadas.

Observe:
- The boy laugh
- The girl likes chocolate
- the girl kissed the boy

Mostra-se a seguir o grafo de derivação desta frase:


S
np
det
the
noun
boy
real
open
[ ]
r
relp
that
s
np
det
the
noun
girl
rel
open
[ ]
r
relp
that
s
np
t
vp
verb
likes
vp
det
[ ]
noun
chocolate
rel
[ ]
close
vp
verb
kissed
np
t
close
vp
verb
laugh

4.3 A Gramática XG como Ferramenta de Programação Lógica

Como a gramática DCG, a gramática XG não é mais do que uma notação conveniente para um conjunto de cláusulas definidas.

Neste ítem será apresentado o processo de implementação da gramática XG em uma linguagem de programação lógica.

Um não-terminal de grau n, ou seja, com n argumentos em XG, corresponde a um predicado de grau n+4 em programação lógica. Destes quatro argumentos adicionais dois são usados para representar a posição das idéias na sentença, como em DCG. Os outros dois argumentos são usados para representar elementos, de uma lista de extraposição, que, conforme explicado na apresentação deste capítulo, contém os componentes que formam as frases relativas, e que devem ser analisados posteriormente.

Ilustrando tem-se:

XG programação lógica

predicado(a,b,...,n) ---> predicado(a,b,...,n,n+1,n+2,n+3,n+4)

Cada elemento desta lista de extraposição representa um símbolo que pode ser substituido pelo seguinte predicado:

x (contexto, tipo, símbolo, xlist) onde,

contexto: . se o `símbolo' é precedido por '...' na regra original então o `contexto' é representado por **gap**
. se o `símbolo' é precedido por ',' na regra original então o `contexto' é representado por **nogap**

tipo: pode ser um terminal (t) ou um não-terminal (nt)
símbolo: é o símbolo propriamente dito

xlist: é o restante da lista de extraposição (no caso de uma lista vazia é representada por '[ ]')

Aplicando esta regra ao exemplo a seguir podemos entendê-la perfeitamente:

XG

rel-marker...trace ---> rel-pronoun

⇓

programação lógica

```
rel-marker (S0,S,X0,x(gap,nt,trace,X)):-rel_pronoun(S0,S,X0,X)
```

No caso em que há apenas um único símbolo no lado esquerdo da regra em XG, a tradução em programação lógica é muito similar às regras DCG.

A tradução de um símbolo terminal 't' que está no lado direito da regra em XG é feita através da utilização do predicado 'terminal'.

Veja o exemplo a seguir:

```
rel-pronoun -> [that] =>   .
rel-pronoun (S0,S,X0,X):- terminal (that,S0,S,X0,X).
```

O predicado auxiliar 'terminal' quando representa um terminal que está entre S0 e S assume a seguinte forma:

```
terminal (T,S0,S,X,X): - gap(X), connects (S0,T,S).
```

onde o predicado auxiliar 'gap' pode ser:

gap (x(gap,T,S,X)) no caso de existir uma lista de extraposição

ou

gap ([]) no caso de a lista de extraposição ser vazia.

No entanto, se o predicado 'terminal' representar um terminal que está na lista de extraposição então apresentará a seguinte forma:

terminal (T,S,S,x(C,t,T,X),X).

No entanto, para cada símbolo não-terminal de grau n, não lider, do lado esquerdo da regra em XG, a tradução em programação lógica deverá incluir a seguinte cláusula:

nt(V1,...,Vn,S,S,X0,X) :- virtual(nt(V1,...,Vn),X0,X)

onde o predicado 'virtual(C,X0,X)' pode ser lido como "C é o constituinte obtido da lista X0 menos os elementos da lista X de extraposição" e as variáveis Vi transferem os argumentos do símbolo na lista de extraposição para o predicado que traduz o símbolo.

Para que sejam satisfeitas estas condições é necessário que o predicado auxiliar 'virtual' seja definido como:

Virtual (NT,x(_,nt,NT,X),X); ou seja, NT é o elemento obtido na lista x(_,nt,NT,X) menos os elementos da lista X.

Para um melhor entendimento deste formalismo lógico, é apresentado no Apêndice C a implementação, em predicados lógicos, da gramática XG mostrada no ítem 4.2.2 deste capítulo, utilizando dos procedimentos lógicos explicados neste ítem.

Neste Apêndice C, é apresentado também uma execução simbólica deste formalismo (XG) visando ilustrar sua capacidade de análise no caso da seguinte frase:

The boy that the girl that likes chocolate kissed laugh.

5. IMPLEMENTAÇÃO DE UM PROTÓTIPO

Neste capítulo é apresentada a implementação de um protótipo de teste de um processador de linguagem natural como interface com base de dados. São apresentadas as principais características da implementação.

Basicamente é fornecida uma visão geral do protótipo implementado, conforme três visões fundamentais.

- Metodologia de Projeto
  Esboça a filosofia de projeto que foi seguida para implementar um protótipo de processamento de linguagem natural.

- Especificação de Requisitos do Projeto de Software
  Define com rigor os requisitos do software. Esta especificação delimita apenas o espectro de soluções válidas, sem tirar a flexibilidade do projeto.

- Descrição do Projeto de Software
  Este item mostra como o software será estruturado de forma a satisfazer os requisitos identificados em sua especificação. Esta descrição fornece um detalhamento pormenorizado do software, possibilitando a partir deste o desenvolvimento das atividades de implementação.

5.1. Metodologia de Projeto

Neste item é apresentado o método de projeto a ser utilizado no processamento de linguagem natural. É discutida também a possibilidade da utilização desta filosofia de trabalho, tanto para análise como para síntese de sentenças em linguagem natural.

O principal problema na análise de sentenças em linguagem natural é o da sua conversão para uma forma lógica que represente seu conteúdo semântico. Em seguida, a expressão lógica resultante deve ser transformada através de um algoritmo de planejamento, em

um código da linguagem Prolog eficiente, ou seja, é feita uma otimização da pergunta de acordo com a abrangência dos conceitos contidos na base de dados a ser considerada. Esse processo de planejamento é explicado no item 5.1.2. Finalmente esta regra em linguagem Prolog é executada e a resposta desejada é obtida

Um dos principais aspectos de sistemas em linguagem natural é que eles devem apresentar uma grande facilidade de adaptação a diversas aplicações. Este tipo de característica é conseguido, desde que a maior parte dos componentes deste sistema seja independente da aplicação.

Em face deste objetivo, o método de análise de sentenças em linguagem natural foi dividido em três fases, conforme mosta a figura 5.1 [8], [9], [24].

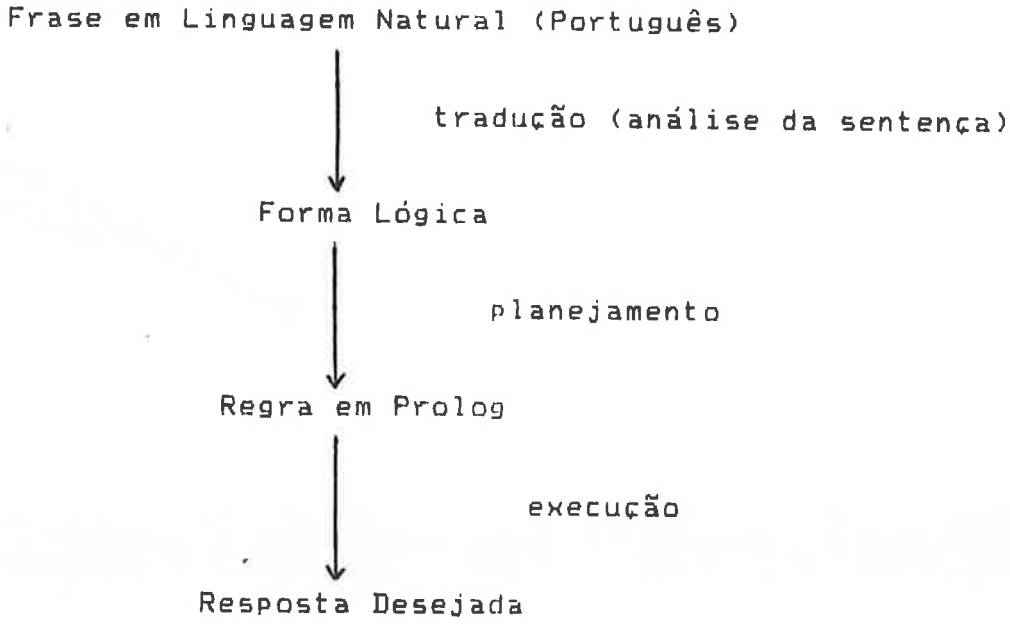


Figura 5.1 - Método de Análise

5.1.1. O Processo de Tradução

Uma maneira de compreender melhor uma frase em linguagem natural é representando-a em uma notação mais concisa. Isto corresponde essencialmente à denominada abordagem lógica e tal representação é dita "Forma Lógica".

Os nomes próprios são representados logicamente por constantes, que são cadeias iniciadas com letra minúscula.

Os verbos, substantivos e adjetivos, em sua grande maioria, são representados por "**predicados**" , contendo um ou mais "**argumentos**". Um predicado, juntamente com seus argumentos, é denominado "**predicação**" (ou algumas vezes "goal" que quer dizer "objetivo").

No exemplo a seguir é ilustrada tal característica.

Brasília é a capital do Brasil ==> capital (brasil, brasília)

Diversas sentenças mais complexas podem ser representadas por um conjunto de predicações, como por exemplo:

O Brasil é um país que é vizinho do Uruguai

$\Downarrow$

país (brasil) & é-vizinho (brasil, uruguai)

Após esta breve explicação, é apresentado a seguir o esquema de trabalho adotado na execução da tradução de sentenças em português para as "Formas Lógicas" equivalentes. Este processo é realizado em três etapas conforme ilustra a figura 5.2.

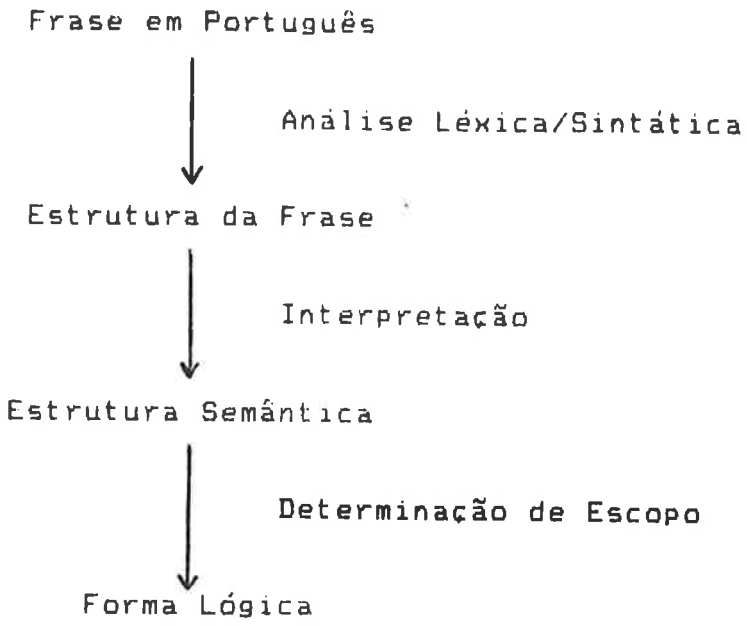


Figura 5.2 - Processo de Tradução

A cada uma das etapas do processo de tradução está associado um dicionário específico. Na etapa de análise existe o dicionário da gramática que contém as definições usuais de categoria sintática e léxica, bem como as palavras determinantes (pronomes interrogativos, advérbios de negação e de intensidade); conectivos lógicos ("e" e "ou") e palavras dependentes da aplicação (substantivos, verbos e adjetivos), correspondentes aos predicados da base de dados.

O formalismo que foi utilizado para a obtenção da estrutura da frase é a "Gramática de Extraposição - XG". Este formalismo é uma evolução da "Gramática de Cláusulas Definidas - DCG". Ambos os formalismos foram apresentados nos capítulos anteriores desta dissertação.

Na etapa de **interpretação** - ou preenchimento de "slots" - tem-se um dicionário de "**templates**", ou seja "**padrões**", que define a tradução de palavras para predicados, e argumentos-**padrão**

exigidos para tais traduções. Cada aplicação irá requerer um diferente conjunto de padrões.

Como exemplo de padrões tem-se:

verbo-transitivo (infinitivo,X,Y, amar(X,Y)) --> [amar]

Neste padrão "verbo-transitivo" é um predicado, um tipo de verbo. As variáveis ` X ' e ` Y ' representam as entidades envolvidas neste tipo de predicado. A constante ` infinitivo ' representa o tempo do verbo. A predicação do verbo amar corresponde a ` amar (X,Y) '.

Finalmente a etapa de "Determinação de Escopo" contém um pequeno dicionário definindo o relacionamento do escopo de atuação para determinantes e outras palavras "operadoras". Este dicionário é independente do domínio de aplicação.

A seguir, são apresentados alguns determinantes com sua respectiva quantificação, através de variáveis lógicas, e de predicações envolvendo tais variáveis. Estas variáveis são denominadas domínio (range) e escopo (scope), representadas pelas letras R e S, respectivamente.

a) alguns -----> exists (X,R & S)

Alguns homens cantam ---> exists (X, homem(X) & cantar(X))

b) não -----> \+ exists (X,R & S)
(obs.: ` \+P ' deve ser lido como, "não pode ser mostrado que P é verdade").

Não há ursos no zoológico ---> \+exists(X,urso(X)&in(X,zoológico)

c) todo -----> \+ exists (X,R & \+S)

O homem habita **todo** país

\+ exists (X, país(X) & \+ habita(homem, X))

d) Quais -----> answer (X) <===== R & S

Quais animais são mamíferos?

answer(X) <=== animal(X) & mamífero(X)

e) Quantos -----> answer(N) <===== number of (X,R & S,N)

Quantos países são desenvolvidos?

answer(N) <=== number of (X, país(X) & desenvolvido(X),N)

5.1.2. Os Processos de Planejamento e Execução

Neste ponto do processamento a frase em linguagem natural já foi mapeada para uma forma lógica equivalente, ou seja, a formalização semântica já está pronta. Apenas no sentido de tornar o protótipo de teste uma entidade relativamente prática e eficiente, que é dada continuidade ao processo de tratamento de sentenças em linguagem natural como interface com uma base de dados.

A principal questão neste aspecto é a de evitar uma explosão combinatória, quando do mapeamento da "forma lógica" em um programa prolog. O programa em prolog obtido deve ser eficiente, podendo ser então executado para produzir as respostas desejadas.

O **processo de planejamento** pode utilizar-se de certos dados estatísticos, relações de domínios, sendo análogo ao trabalho de otimização de perguntas, [23].

Há dois aspectos importantes no processo de planejamento:

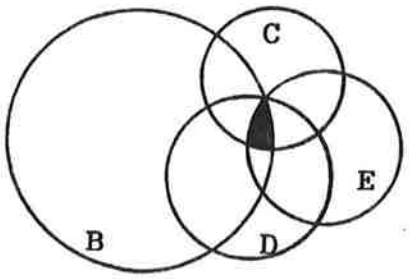
a) a ordem das predicações dentro de uma regra em prolog irá determinar, em geral, a ordem segundo a qual o programa será executado.

b) a notação em "problemas independentes" no sentido de limitar a quantidade de "back-tracking" na execução do programa em prolog.

Para ilustrar tais recomendações é apresentado a seguir um exemplo:

regra 1
```
A if B,
     C,
     D,
     E.
```

regra2
```
A if C,
     D,
     E,
     B.
```



Obs.: para se obter a resposta desejada a regra 2 é mais eficiente do que a regra 1.

- Domínio das predicações

5.2. Especificações dos Requisitos do Software - ERS [38]

Este item define todos os requisitos do protótipo de teste, que corresponde à interface em linguagem natural com uma base de dados.

5.2.1. Introdução

O propósito desta especificação é delinear algumas características funcionais que o protótipo de uma interface em linguagem natural com base de dados deve apresentar.

Este protótipo de teste apresenta algumas limitações, que são esclarecidas nos itens que se seguem.

O nome de software-produto é "INTERFACE PARA BANCO DE DADOS EM LINGUAGEM NATURAL". Este software tem a capacidade de, a partir de algumas perguntas fornecidas pelo usuário, fornecer a resposta desejada, de acordo com a base de dados consultada.

5.2.2. Descrição Geral

São descritos neste item os fatores gerais que afetam este software-produto e seus requisitos.

5.2.2.1. Perspectiva do Produto

Este software-produto tem como objetivo implementar um protótipo de teste, de uma interface em linguagem natural com uma base de dados, [1], [3].

A seguir é mostrado um diagrama geral do software-produto.

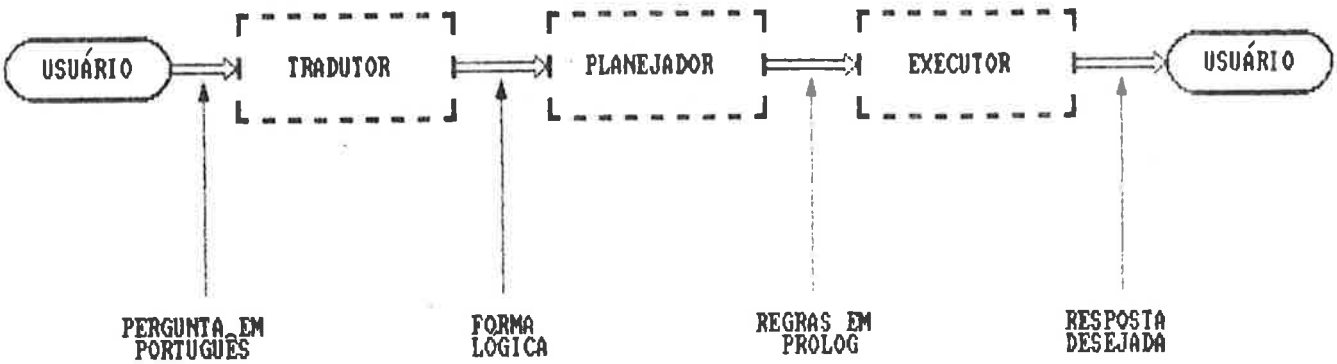


Figura 5.3 - Esquema Geral do Software-Produto.

Este sistema é composto de quatro elementos distintos:

- USUÁRIO;
- TRADUTOR;
- PLANEJADOR;
- EXECUTOR.

O USUÁRIO é um indivíduo que deseja obter uma resposta a respeito de um determinado assunto, contido nas informações armazenadas em uma base de dados.

O TRADUTOR tem a função de converter uma pergunta, feita em linguagem natural (português no caso) para uma forma lógica, que deve representar a semântica da frase interrogativa correspondente.

Basicamente o TRADUTOR é formado por um analisador semântico, dirigido por sintaxe. Isto é justificável pelo fato de a semântica de uma frase em linguagem natural, estar bastante ligada à sua estrutura sintática. Com isto evita-se também a obtenção de uma forma semântica que não corresponde à realidade da interrogação.

O PLANEJADOR tem a função de transformar a forma lógica em regras de linguagem prolog, com predicados compatíveis com a base de dados a ser consultada. Neste processo de planejamento deve ser utilizada também a técnica de otimização, mencionada no item 5.1 (Metodologia de Projeto).

O EXECUTOR simplesmente deve aplicar as regras em prolog à base de dados a ser consultada obtendo daí as respostas desejadas.

Uma perspectiva de utilização deste software-produto é que seja adaptado para funcionar como um TRADUTOR AUTOMÁTICO. Tal procedimento é explicado através do esquema da figura 5.4.

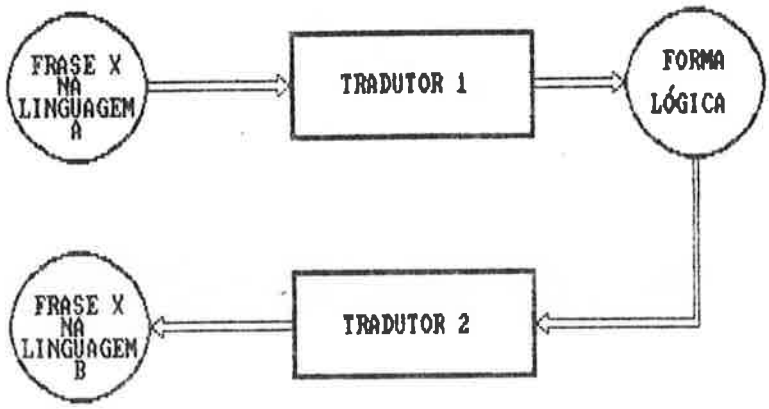


Figura 5.4 - Esquema Geral do Tradutor-Automático.

O TRADUTOR 1 utiliza discionários compatíveis com a linguagem natural A, enquanto que o TRADUTOR 2 utiliza os dicionários da linguagem natural B.

Esses tradutores incorporam também a sintaxe das linguagens envolvidas no processo. A FORMA LóGICA representa a semântica das frases, e pode ser considerada como uma FORMA INTERMEDIÁRIA independente da linguagem a ser utilizada.

5.2.2.2. Funções do Produto

A principal função do Software-Produto é gerar uma resposta correta à interrogação feita pelo usuário.

É apresentado na figura 5.5 um Diagrama de Dados - DFD do software-produto destacando suas principais funções.

Neste diagrama estão identificados os principais elementos:

- TRADUTOR;
- PLANEJADOR;
- EXECUTOR.

Além disso são destacados também a região do DFD que deve ser iterativa e aquela que é executada uma única vez ao inicio do processamento.

Com relação ao TRADUTOR-AUTOMÁTICO, seu diagrama de fluxo de dados - DFD é apresentado na figura 5.6. Neste caso não há a necessidade dos elementos PLANEJADOR e EXECUTOR, pois neste processo de tradução a principal característica é obter uma libnguagem intermediária da representação da semântica de uma frase em uma linguagem A,no caso a forma lógica,para posterior tradução em uma outra fase de mesmo significado em uma linguagem B.

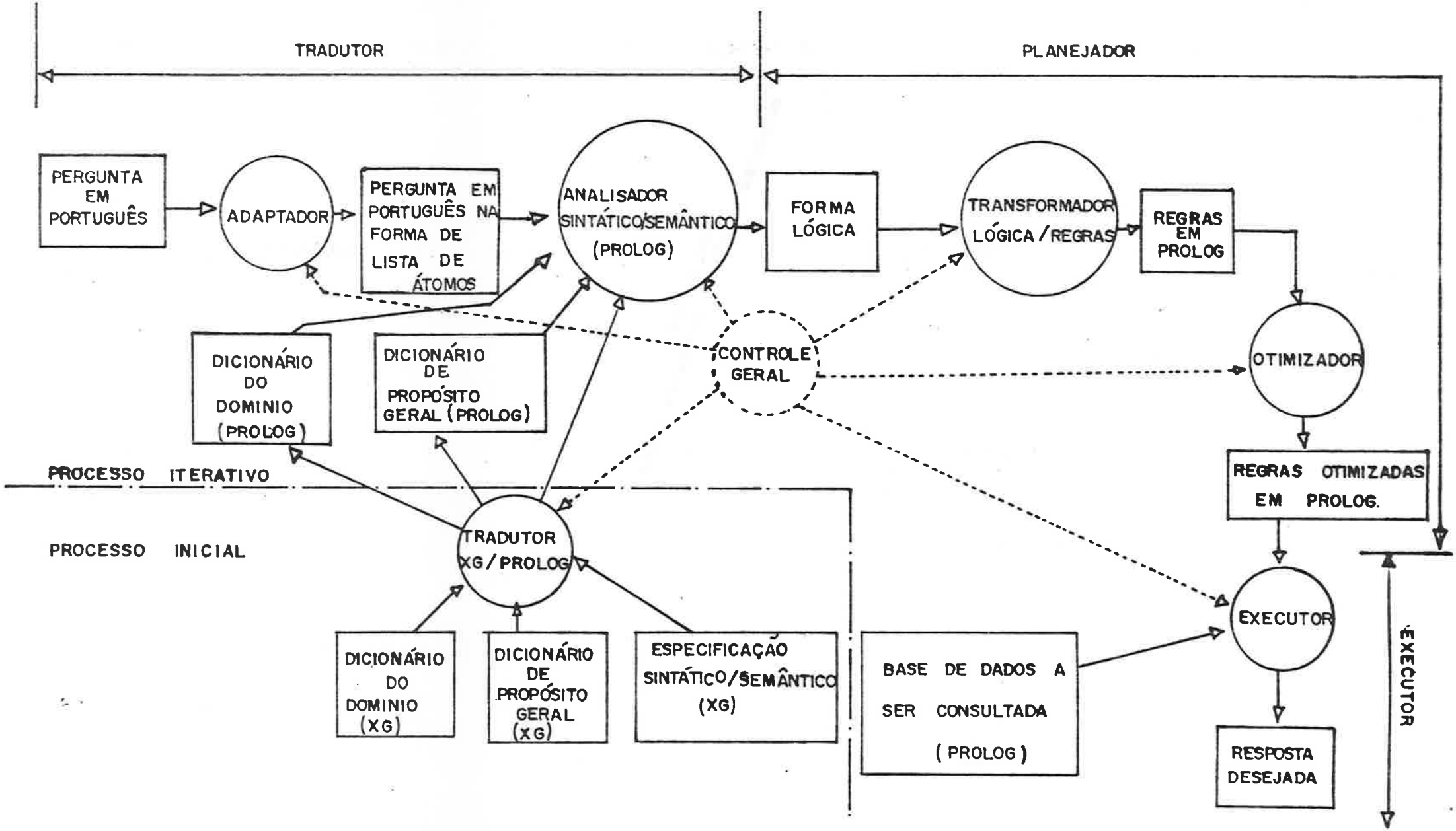


figura 5.5 DIAGRAMA DE FLUXO DE DADOS DO SOFTWARE_PRODUTO_ INTERFACE EM LINGUAGEM NATURAL

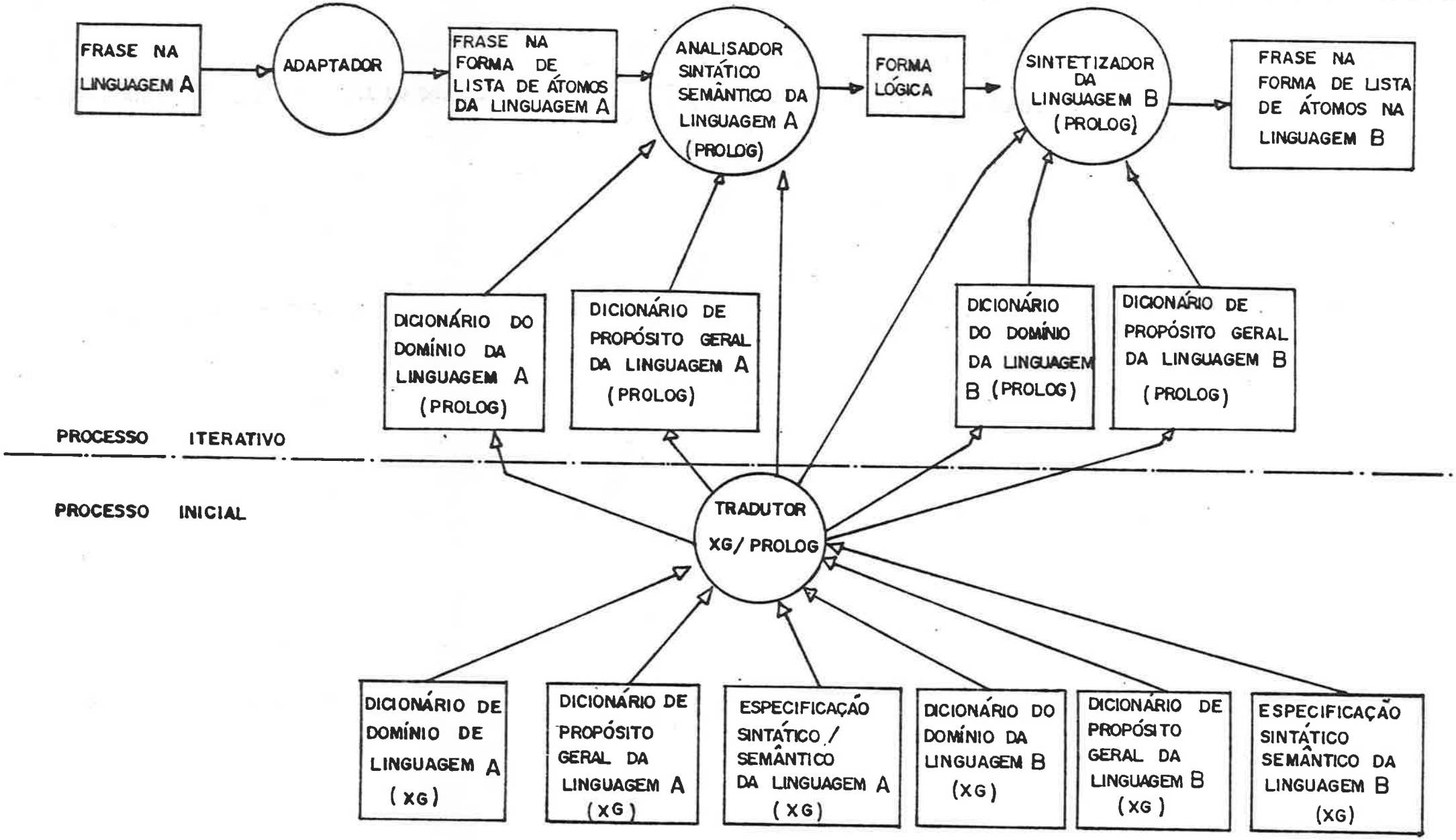


figura 5.6 – DIAGRAMA DE FLUXO DE DADOS DO TRADUTOR - AUTOMÁTICO

5.2.2.3. Limitações Gerais

Convém esclarecer que o objetivo deste software-produto não é de entender diversos tipos de frases interrogativas, mas sim demonstrar o poder do instrumental desenvolvido em função de alguns casos práticos. Todo o embasamento teórico e ferramental desenvolvido nesta dissertação, pode ser utilizado para a obtenção de uma maior capacidade deste software-produto, quer do ponto de vista semântico, sintático ou léxico.

No atual estágio este software-produto não reconhece o significado da pontuação.

5.2.3. Especificação dos Requisitos

Este é um dos ítens mais importantes deste capítulo. Contém os detalhes necessários para se implementar o software-produto.

5.2.3.1. Requisitos Funcionais

Aqui são descritas as ações fundamentais que estão presentes neste software-produto.

O software-produto em questão está dividido em 6 (seis) blocos funcionais:

- Adaptador
- Tradutor XG/Prolog
- Analisador Sintático/Semântico
- Transformador Lógica/Regras
- Otimizador
- Executor

Esta organização adotou o critério de funcionalidade.

5.2.3.1.1. Requisitos do Bloco Funcional: ADAPTADOR

a) Introdução

O objetivo deste bloco funcional é o de eliminar e eventualmente interpretar toda a pontuação da sentença, converter a sentença de entrada - vista como uma cadeia de caracteres - em uma lista de átomos onde cada átomo representa uma palavra (Conversor). Além destas funções, este bloco funcional deve localizar os átomos flexionados (gênero, grau e número) na sentença de entrada e colocar sua classificação no Dicionário do Domínio (Analisador Léxico). Este tipo de procedimento é de grande importância, caso contrário, tal trabalho deveria ser realizado toda vez que a classificação deste átomo, fosse requerida no bloco funcional "Analisador Sintático/Semântico".

Na figura 5.7 é apresentado um Diagrama de Fluxo de Dados - DFD mais detalhado deste bloco funcional.

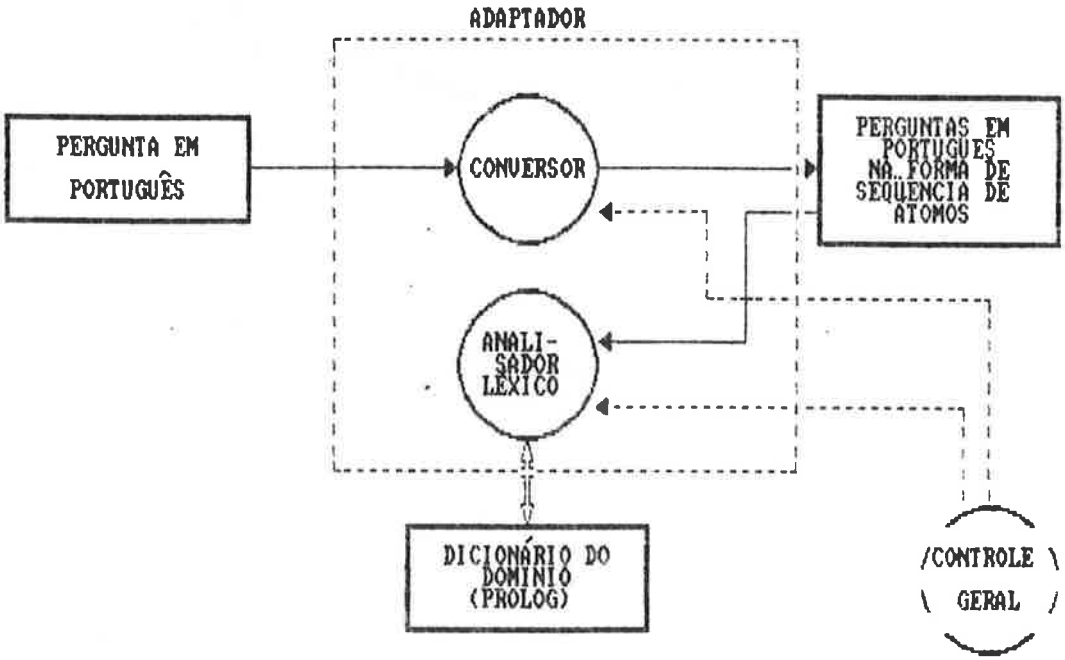


Figura 5.7 - Diagrama de Fluxo de Dados do ADAPTADOR.

b) Entradas

A entrada do Conversor é uma cadeia de caracteres representando uma pergunta em português, com pontuação.

A entrada do Analisador Léxico é uma sequência de átomos, onde cada átomo representa uma palavra da sentença de entrada.

c) Saídas

A saída do Conversor é uma sequência de átomos, onde cada átomo representa uma palavra da sentença de entrada. Os caracteres de pontuação não estão presentes, podendo eventualmente serem interpretados.

A saída do Analisador Léxico constitui-se na classificação léxico-semântico dos átomos flexionados encontrados na sentença de entrada em forma de sequência.

d) Processamento

O sub-bloco funcional Conversor tem por finalidade eliminar os espaços em branco e os caracteres de pontuação, além de transformar as cadeias de caracteres em "átomos" ou em "inteiros" conforme for mais adequado. Estas transformações são necessárias para preparar os dados convenientemente, para serem processados pelo bloco funcional Analisador Sintático/Semântico.

O sub-bloco funcional Analisador Léxico destina-se a localizar os átomos de entrada (sentença na forma de lista de átomos) flexionados e cuja forma primitiva está no Dicionário de Domínio, e inserir neste dicionário a classificação léxico-semântica deste novo átomo. Tal procedimento é importante, quando o bloco funcional Analisador Sintático/Semântico, necessitar consultar sua classificação, esta já estará introduzida no

dicionário do domínio, podendo ser diretamente extraida deste dicionário.

Os átomos flexionados que podem ser classificados automaticamente são os substantivos, os adjetivos, os verbos irregulares (alguns) e todos os verbos regulares [17].

Os substantivos que estão flexionados no plural irão assumir a mesma interpretação semântica do substantivo primitivo, ou seja, no singular. Convém esclarecer que substantivos flexionados em gênero e grau, podem também ser manipulados da mesma forma havendo, no entanto, alguma modificação na interpretação semântica do substantivo primitivo, através da inclusão de atributos adicionais específicos a este tratamento.

Os verbos irregulares são simplesmente consultados numa base de verbos irregulares e assumem a interpretação da primitiva - tempo infinitivo.

Os verbos regulares conjugados são localizados através da obtenção do radical e da terminação correspondente. Eles vão assumir a interpretação da primitiva - tempo infinitivo.

Para os verbos conjugados no futuro ou passado, a interpretação semântica poderia ser uma adaptação do tempo presente, ou seja, o atributo de tempo passaria a assumir o valor presente, passado ou futuro.

Convém esclarecer que a classificação das flexões verbais possuem também uma variável de número (singular ou plural), que auxilia na verificação da concordância verbal, pelo bloco funcional Analisador Sintático/Semântico.

5.2.3.1.2. Requisitos do Bloco Funcional: TRADUTOR XG/PROLOG

a) Introdução

O objetivo deste bloco funcional é o de traduzir para a linguagem Prolog toda a notação em XG, que define parte da gramática da língua portuguêsa, como também dos dicionários do domínio e o de propósito geral. Este bloco funcional foi adaptado daquele desenvolvido por Fernando Pereira na Universidade de Edinburgh para o Sistema DEC-10 PROLOG. [11]

b) Entradas

As entradas deste bloco funcional são o Dicionário de Domínio, o Dicionário de Propósito Geral e a Especificação Sintática/Semântica da linguagem natural em XG.

c) Saídas

As saídas deste bloco funcional são o Dicionário de Dominio, o Dicionário de Propósito Geral e o Analisador Sintático/Semântico, todos devidamente traduzidos para a linguagem Prolog.

d) Processamento

A finalidade deste bloco funcional é a de converter o formalismo XG, explicado no capítulo 4 desta dissertação, em regras prolog executáveis. Assim sendo, a partir do formalismo XG obtém-se um código executável.

5.2.3.1.3. Requisitos do Bloco Funcional: ANALISADOR SINTÁTICO/SEMÂNTICO

a) Introdução

O principal objetivo deste bloco funcional é o de gerar uma forma lógica, que represente adequadamente a semântica da frase interrogativa, fornecida ao sistema pelo usuário.

b) Entradas

As entradas deste bloco funcional são o Dicionário do Domínio e o de Propósito Geral, bem como a frase interrogativa inicial, convertida para a forma de lista de átomos.

O bloco funcional Tradutor XG/Prolog não é uma entrada mas sim um originador do Analisador Sintático/Semântico.

Convém esclarecer que neste ponto os Dicionários de entrada, já estão traduzidos para código em linguagem Prolog e contém também a expressão semântica das palavras.

c) Saídas

A única saída deste bloco funcional é a expressão lógica que representa a semântica da frase fornecida como entrada.

No atual estado deste software-produto os seguintes operadores lógicos são utilizados:

- <==
- &
- meta-operador "number-of"
- V

Estes operadores são aplicados sobre predicações de maneira a gerar, uma expressão lógica consistente e coerente com a interpretação semântica.

d) Processamento

Como este bloco funcional é uma tradução para a linguagem Prolog, da Especificação Sintático/Semântico de um sub conjunto da lingua portuguêsa, sua principal função é a de realizar uma análise sintática da frase de entrada e, no caso de tal frase estar correta, gerar uma expressão lógica que represente a semântica correspondente.

À medida que este bloco funcional necessita consultar os Dicionários do Domínio ou de Propósito Geral durante o processo de análise sintática, são obtidos destes dicionários a classificação léxica como também a interpretação semântica dos átomos. Ao longo do processo de análise sintática estas interpretações vão sendo combinadas, para formar ao final uma forma lógica coerente.

Este tipo de especificação torna bastante paralelos os processos de análise sintática e de interpretação semântica, característica esta, que diverge com vantagens de algumas correntes de processamento sequencial, em que é efetuada primeiramente a análise sintática e depois, a semântica.

5.2.3.1.4. Requisitos do Bloco Funcional: TRANSFORMADOR LóGICA/REGRAS

a) Introdução

O objetivo deste bloco funcional é o de transformar a expressão lógica obtida com o auxílio do Analisador Sintático/Semântico em regras executáveis da linguagem Prolog.

b) Entradas

A Entrada deste bloco funcional é a Forma Lógica que deve representar a semântica da frase (interrogativa) de entrada.

c) Saídas

A saída deste transformador é um conjunto de regras executáveis, denotadas em linguagem Prolog, que devem representar fielmente a expressão lógica correspondente.

d) Processamento

Este bloco funcional deve traduzir corretamente para a linguagem Prolog a expressão lógica da entrada.

Desta forma: P <== Q' deve ser traduzido para ' P if Q ' ;
' P & Q ' deve ser traduzido para ' P and Q ' ;
'(P V Q)' deve ser traduzido para '(P ; Q)' ;
number-of (X,P,N) deve ser traduzid para set-of (X,P,S) and size of (S,N).

Obs.: number-of (X,P,N) é lido como "o número dos X's que satisfazem P é N".

5.2.3.1.5. Requisitos do Bloco Funcional: OTIMIZADOR

a) Introdução

O objetivo deste bloco funcional é tornar mais eficiente a execução do conjunto de regras em prolog anteriormente produzido.

b) Entradas

A entrada do Otimizador é um conjunto de regras em Prolog que representam a expressão semântica da frase de entrada.

c) Saídas

A saída deste bloco funcional é também um conjunto equivalente de regras executáveis em Prolog, porém melhoradas quanto ao tempo de processamento através de uma reordenação adequada.

d) Processamento

Como já foi explicado no item 5.1 deste capítulo, o processo de otimização das regras em Prolog, deve basear-se em atributos específicos dos predicados da Base de Dados a ser consultada.

Estes atributos representam a abrangência do domínio dos seus respectivos predicados e devem ser utilizados para reorganizar as regras em Prolog.

5.2.3.1.6. Requisito do Bloco Funcional: EXECUTOR

a) Introdução

O objetivo deste bloco funcional é de adaptar/formatar as regras executáveis em Prolog à Base de Dados a ser consultada e obter a resposta desejada. Assim sendo, este bloco funcional permite a adaptação deste sistema de processamento de linguagem natural a qualquer padrão de base de dados.

b) Entradas

A entrada do EXECUTOR são as regras otimizadas, denotadas em linguagem Prolog.

c) Saídas

A saída deste bloco funcional é um conjunto de regras executáveis e adaptadas à Base de Dados em questão.

d) Processamento

Este bloco funcional deve adaptar as regras otimizadas em linguagem Prolog, aos formatos, padrões e outras características da particular Base de Dados. Neste software-produto, este bloco funcional apenas executas as regras otimizadas, fornecidas pelo bloco Otimizador. A Base de Dados a ser consultada está num formato compatível com as regras originadas em Prolog. A existência deste bloco funcional é destinada a facilitar a adaptação deste sistema a outros tipos/formatos de Bases de Dados, [5], [7].

5.2.3.2. Requisitos dos Dicionários

Neste item são apresentados os requisitos a serem preenchidos pelos seguintes dicionários do software-produto:

- Dicionário do Domínio
- Dicionário de Propósito Geral
- Especificação Sintático/Semântico

5.2.3.2.1. Dicionário do Domínio

Este dicionário é uma das entradas do bloco funcional Tradutor XG/Prolog e em seguida é utilizado pelo bloco funcional Analisador Sintático/Semântico.

Devem fazer parte deste dicionário todas as palavras "primitivas" que estão relacionadas com o domínio do assunto contido na Base de Dados a ser consultada.

Para cada uma dessas palavras, conforme sua classificação léxica, deve-se ter as seguintes informações:

. classificação léxica da palavra "primitiva"
. interpretação semântica
. variáveis envolvidas na interpretação semântica

A grande vantagem de se utilizar este tipo de especificação é que quando este dicionário for consultado é devolvida não só a classificação léxica, utilizada na análise sintática, mas também a interpretação semântica da mesma fica disponível para ser convertida para a forma de regras executáveis.

5.2.3.2.2. Dicionário de Proposta Geral

Este dicionário também é uma entrada para o bloco funcional Tradutor/XG Prolog, sendo em seguida utilizado pelo bloco funcional Analisador Sintático/Semântico.

Devem fazer parte deste dicionário os termos não dependentes do domínio do assunto sendo indagado. Dentro deste escopo estão os determinantes, conectivos e os pronomes relativos, nesta implemntação. Dentre os determinantes pode-se citar os pronomes interrogativos e os advérbios de negação e de intensidade. Os conectivos abrangem as conjunções coordenadas aditivas e as alternativas.

Da mesma forma que no Dicionário de Domínio, cada uma das palavras contidas neste dicionário deve exibir os seguintes atributos:

. classificação léxica da palavra
. interpretação semântica
. variáveis envolvidas na interpretação semântica

Convém esclarecer que, a interpretação das palavras contidas neste dicionário atuam sobre as interpretações semânticas das palavras, contidas no Dicionário do Domínio. Desta forma

consegue-se gerar expressões lógicas, que representam adequadamente a semântica das frases analisadas.

No final da análise de uma determinada frase a interpretação semântica fica disponível para ser convertida para a forma de regras executáveis.

5.2.3.2.3. Especificação Sintática/Semântica

Este dicionário deve conter uma especificação formal da linguagem a ser analisada, no caso o Português, e também o relacionamento das interpretações semânticas das partes que constituem a frase a ser interpretada.

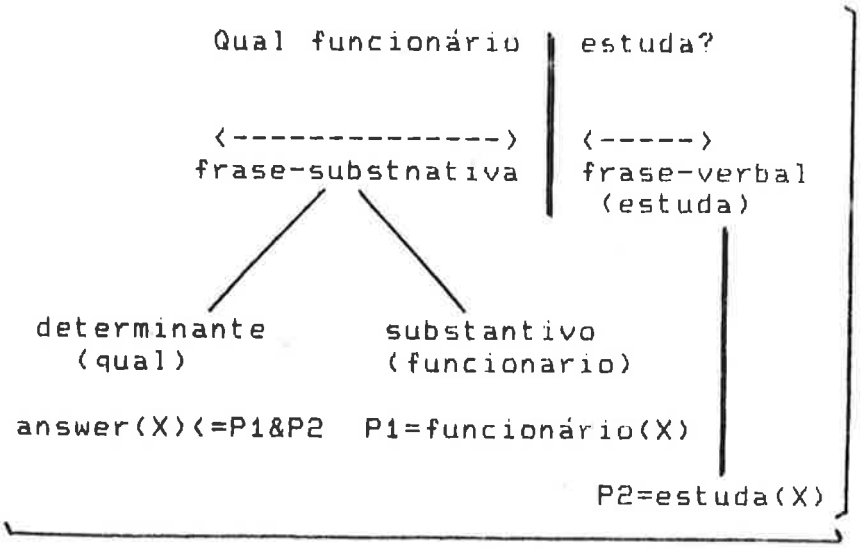
Pode-se entender melhor este procedimento através do seguinte exemplo:

Sentença(P) -> frase-substantiva(P1,P), frase-verbal(P1)

frase-substantiva(P1,P) -> determinante(P2,P1,P), substantivo(P2)

A interpretação destas duas regras é: "A interpretação P de uma sentença corresponde a interpretação da frase substantiva, aplicada sobre a interpretação P1 da frase verbal. Por sua vez, a interpretação P da frase substantiva corresponde a interpretação do determinante, sobre a interpretação P1 da frase verbal, e a interpretação P2 do substantivo."

A seguir é executado tal procedimento sobre uma frase:



Relacionamento das interpretações das partes que constituem a frase.

answer(X) <== funcionário(X) & estuda(X)

Convém lembrar que este dicionário deve ser especificado através do formalismo XG que, conforme foi descrito no capítulo 4, apresenta uma grande capacidade de representar sentenças em linguagem natural.

No caso específico deste software-produto as regras de especificação contidas neste dicionário procuram abranger a formalização de sentenças interrogstivas. Dentro desta característica devem ser estudadas as sentenças interrogativas com frases subordinadas adjetivas, frases negativas e demais contruções relacionadas.

5.2.3.3. Requisitos da Interface com o Usuário

Neste item são apresentadas as características que o software-produto deve apresentar com relação ao diálogo com o usuário, para cada questão por este formulada:

<pergunta em português realizada pelo usuário> (Usuário)
<expressão lógica que representa a semântica da frase interrogativa> (Sistema)
<resposta desejada pelo usuário> (Sistema)

5.3. Descrição do Projeto de Software - DPS [39]

Este item mostra qual a arquitetura e os detalhes implementados do software-produto para satisfazer os requisitos identificados no item 5.2. Este item contém os detalhes necessários para se implementar o software-produto.

5.3.1. Introdução

O propósito desta descrição é apresentar as principais características de projeto adotadas. O nome do software-produto em questão é "INTERFACE PARA BANCO DE DADOS EM LINGUAGEM NATURAL". Todas as funções descritas na Especificação dos Requisitos - item 5.2 - são abordadas neste item do ponto de vista de projeto. O Software-Produto desenvolvido nesta pesquisa foi implementado no ambiente Arity-Prolog [32], utilizando um microcomputador tipo PC. A quantidade de memória utilizada pelo sistema está em torno de 15K (interpretado) e o tempo de desenvolvimento da parte prática foi de 6 meses.

5.3.2. Descrição de Decomposição

Esta descrição registra a divisão do software-produto em entidades de projeto (dados e processos). Ela descreve também os detalhes internos de cada entidade de projeto.

5.3.2.1. Decomposição dos Dados - Dicionários

5.3.2.1.1. Dicionário do Domínio

Este dicionário deve conter todas as palavras "primitivas" que estão relacionadas com o domínio do assunto contido, na Base de Dados a ser consultada.

Para implementar este dicionário foi adotado o seguinte procedimento:

classificação-léxica (argumentos) --> [átomo]

Este software-produto implementa este dicionário para **substantivos**, **adjetivos**. **advérbios** e **verbos** (transitivos, intransitivos e de ligação). As demais classificações podem ser facilmente implementadas dentro do mesmo método de implementação.

Para **substantivo** temos:

substantivo (número, X, átomo(X)) --> [átomo]

singular plural

Ex.: substantivo (singular, X, funcionário(X)) --> [funcionário]

Obs.: Este software-produto trata apenas a flexão em número dos substantivos. A implementação de outros tipos de flexão, é possível criando-se novos atributos: grau e gênero.

Para **adjetivo** tem-se:

adjetivo (número, X, Advérbio, átomo(X, Advérbio)) --> [átomo]

singular plural

advérbio modificando o adjetivo.

Ex.: adjetivo (singular, X, muito, bonita(X, muito)) --> [bonita]

Para advérbio tem-se:

advérbio (-, átomo) --> [átomo]

Ex.: advérbio (-, muito) --> [muito].

Para nomes próprios tem-se:

nome (número, átomo) --> [átomo]

Ex.: nome (singular, maria) --> [maria]

Para verbos transitivos e de ligação temos:

verbo-transitivo (infinitivo, X, Y, predicado(X, Y)) --> [verbo]

↑
tempo

Ex.:verbo-transitivo(infinitivo, X, S, salário(X, S)) --> [ganhar]
verbo-ligação (infinitivo, X,Y, ser(X,Y)) --> [ser]

Obs.: Este software-produto transforma os verbos para o infinitivo. No entanto, poder-se-á tratar também verbos no passado e no futuro. Neste caso o atributo de tempo do verbo teria 3 valores possíveis: "presente", "passado" e "futuro".

Para verbos intransitivos temos:

Verbo-intransitivo (infinitivo, X, verbo(X)) --> [verbo]

Ex.: verbo-intransitivo (infinitivo, X, estudar(X)) --> [estudar]

5.3.2.1.2. Dicionário de Propósito Geral

Este dicionário deve conter todos os determinantes, conectivos e pronomes relativos. Dentre os determinantes pode-se citar os pronomes interrogativos e os advérbios de negação e de intensidade. Os conectivos abrangem as conjunções coordenadas aditivas e as alternativas.

Para os determinantes tem-se:

```
determinante (-,X,P1,P2, predicado(P1,P2)) --> [Determinante]
```

```
Ex.: determinante (-,X,P1,P2, supera(P2,P1)) --> [mais].
     determinante (-,X,P1,P1, not(P1)) --> [não].
     determinante (-,X,P1,P2, answer(X),<== P1&P2) --> [quais].
```

Para os conectivos tem-se:

```
conectivo (átomo) --> [átomo]
```

```
Ex.: ligação (e) --> [e].
     ligação (ou) --> [ou].
```

Com relação aos pronomes relativos tem-se:

```
pronome-relativo (átomo) --> [átomo]
```

```
Ex.: pronome-relativo (que) --> [que]
```

5.3.2.1.3. Especificação Sintático/Semântico

Este dicionário deve conter uma especificação formal da linguagem portuguesa capaz de permitir a interpretação e a

obtenção de respostas à perguntas interrogativas feitas à Base de Dados.

A seguir é mostrado alguns aspectos da especificação formal implementada em XG, que foi explicada no capítulo 4:

```
sentence(N,X,P) --> frase-substantiva(N,X,P1&P2,P),/*P é a semân-
                                                    tica da frase*/
                    frase-verbal(N,X,P1),/*P1 é a semântica da
                                                    frase verbal*/
                    ligação-sentença(N,X,P2)./*P2 é a semântica de
                                                uma outra sentença
                                                ligada através do
                                                conectivo "e" */

sentence(N,X,P) --> frase-substantiva(N,X,P1VP2,P),/*P é a semân-
                                                    tica da frase*/
                    frase-verbal(N,X,P1),/*P1 é a semântica da
                                                    frase verbal*/
                    união-sentença(N,X,P2)./*P2 é a semântica de
                                                uma outra sentença
                                                ligada através do
                                                conectivo "ou" */

ligação-sentença(-,-,true) --> [ ]. /* não há outra sentença */

ligação-sentença(N,X,P) --> open
                            conectivo, /* conectivo "e" */
                            sentence (N,X,P),
                            close.

união-sentença(N,X,P) --> open,
                          or, /* conectivo "ou" */
                          sentença (N,X,P),
                          close.
```

```
frase-substantiva(N,X,P1,P) --> determinante(N,X,P3&P2&P4,P1,P),
                                     /* P - semântica da frase */
                 ↑                substantivo(N,X,P3), /* P3 - se-
            semântica da                 mântica do substantivo */
            frase verbal         frase-sub-relativa(N,X,P2),   /*
                                 semântica da frase subordinada */
                                 ligação-frase-sub-relativa(N,X,P4)
                                 ./* semântica de outra frase  su-
                                   bordinada ligada pelo conectivo
                                   "e" */

frase-substantiva(N,X,P1,P) --> determinante(N,X,P3&(P2VP4),P1,P),
                                     /* P semântica da frase */
                 ↑                substantivo(N,X,P3),/* P3 - semân-
            semântica da                 tica do substantivo */
            frase verbal         frase-sub-relativa(N,X,P2), /* se-
                                    mântica da frase subordinada */
                                 união-frase-sub-relativa(N,X,P4).
                                 /* semântica de outra frase subor-
                                    dinada relativa ligada pelo co-
                                                  nectivo "ou" */

frase-substantiva(N,X,P,P) --> substantivo(N,X,P); /* só tem subs-
                                                        tantivo */
                               nome(N,X). /* só tem nome */

frase-substantivo(N,X,P,P) --> trace. /* necessário para entender
                                         pronomes relativos e co-
                                         nectivos lógicos como "e"
                                         e "ou" */

frase-verbal(N,X,P) --> verbo-trans(N,X,Y,P1), /* P1 -   semântica
                                       do verbo transitivo */
                        frase-substantivo(N1,Y,P1,P)./* semântica
                                          da frase verval */
```

```
frase-verbal(N,X,P) --> verbo-ligação(N,X,P2,P),/* P - interpreta-
                                                   ção da frase verbal */
                        frase-adjetiva(N,X,P1,P2)./ P2 - interpre-
                                                   tação da frase adjetiva */

frase-verbal(N,X,P) --> verbo-intrnasitivo(N,X,P) /* P - interpre-
                                                   tação do verbo intransitivo */

frase-sub-relativa(-,-,true) --> [ ]. /* Não há frase subordinada
                                                   relativa */

frase-sub-relativa(N,X,P1) --> open,
                               rel-marker,
                               sentence(N,X,P1),  /* interpretação
                                     frase subordinada relativa*/
                               close.

ligação-frase-sub-relativa(-,-,truel) --> [ ]. /* não há frase su-
                                                  bordinada relativa
                                                  ligada pelo conec-
                                                  tivo "e" .*/

ligação-frase-sub-relativa(N,X,P1&P2) --> open,
                                          conectivo, /* conectivo
                                                          "e" */
                                          trace,
                                          frase-sub-relativa (N,X,
                                          P1),/*P1 - semântica da
                                              frase subordinada */
                                          ligação-frase-subrelati-
                                          va(N,X,P2), /* P2 -  se-
                                          mântica da outra frase
                                          subordinada ligada pelo
                                          conectivo "e" */
```

```
união-frase-sub-relativa(-,-,false) --> [ ]. /* não há frase  su-
                                                bordinada relativa
                                                ligada pelo conec-
                                                      tivo "ou" */

união-frase-sub-relativa(N,X,((P1VP2)& true)) --> open,
                                                   or,/* conectivo
                                                           "ou" */
                                                   trace,
                                                  relativa(N,X,P1),
                                                   /* P1 - semânti-
                                                    ca da frase su-
                                                       bordinada */
                                                   união-frase-sub-
                                                  relativa(N,X,P2),
                                                   /* P2 - semânti-
                                                   ca da frase  re-
                                                   lativa   ligada
                                                   pelo  conectivo
                                                           "ou" */

frase-adjetiva(N,X,P,P) --> adverbio(-,Y),
                            adjetivo(N,X,Y,P). /* P - interpreta-
                                                  ção do adjetivo modi-
                                                ficado pelo adverbio */

rel-marker...trace --> pronome relativo(X) /* pronome relativo */

conectivo...trace --> ligação(X) /* conectivo "e" */

or...trace --> união(X) /* conectivo "ou" */

open...close --> [ ]. /* aninhamento das frase relativas explicado
                                                  no capítulo 4 */
```

5.3.2.1.4. Banco de Dados

Este banco contém as informações que se deseja acessar pelo usuário. Elas estão organizadas dentro de uma estrutura compatível com a linguagem Prolog.

```
funcionário (paulo).
funcionário (josé).
funcionário (pedro).
funcionário (carlos).
estudar (paulo).
estudar (pedro).
estudar (josé).
salário (josé, 40).
salario (paulo, 35).
salario (pedro, 70).
salario (carlos, 56).
solteiro (paulo).
solteiro (jose).
solteiro (carlos).
admira (maria, paulo).
admira (maria, jose).
admira (maria, carlos).
admira (carlos, maria).
admira (jose, maria).
admira (pedro, maria).
```

5.3.2.2. Decomposição dos Processos

É apresentada a seguir uma descrição do processo de cada bloco funcional identificado na Especificação de Requisitos do Software-Produto.

5.3.2.2.1. Descrição do Processo CONVERSOR

A função deste bloco funcional é a de eliminar os espaços em branco e os caracteres de pontução da frase interrogativa. Além disso deve dar origem a uma lista de átomos e inteiros. Os inteiros são utilizados no caso de tratamento de valores numéricos.

A seguir é detalhada a parte lógica da implementação:

```
            resposta se ler_cadeia,
                        transformar_cadeia_em_lista_de_string,
                        limpar_lista,
                        achar_palavras,
/*devido a implementação
  no Arity-Prolog */---> converter_palavras_do_string_em_átomos_
                        ou-inteiros.

             limpar_lista se
                        limpar_pontuação

             achar_palavras se
                        localizar_palavra,
                        converter_em_string,
                        guardar_numa_lista_nova_palavra,
                        achar_outras_palavras.

    converter_palavras_do_string_em_átomos_ou_inteiros se
                                  converter_strings_em_átomos;
                                  converter_strings_em_inteiros.
```

5.3.2.2.2. Descrição do Processo ANALISADOR LÉXICO

A função deste bloco funcional é inserir no Dicionário de Domínio a classificação léxico-semântica dos átomos da frase de entrada cuja primitiva já existia neste dicionário.

A implementação lógica deste processo é a seguir detalhada:

```
achar-primitiva se

                e-substantivo-primitivo,
                e-nome;
                e-adjetivo;
                e-verbo-intransitivo-primitivo;
                e-verbo-transitivo-primitivo;
                e-verbo-ligação-primitivo;
                e-pronome-relativo;
                e-determinante;
                e-conectivo-de-ligação;
                e-conectivo-de-união;
                e-número.

achar-pimitiva se /* não é átomo primitivo-verificar se é
                     substantivo */

                acha-terminação,
                transforma-terminação,
                acha-novo-átomo,
                verifica-se-é-substantivo-primitivo,
                coloca-no-dicionário-novo-átomo.

acha-primitiva se /* não é átomo primitivo-verifica se é verbo
                     irregular transitivo, de ligação e intran-
                     sitivo */

            é-verbo-irregular,
            localiza-o-verbo-primitivo,
            coloca-no-dicionário-novo-átomo.

acha-primitiva se /* não é átomo primitivo-é verbo */

            localiza-radical-do-verbo,
            verifica-se-está-no-dicionário,
            localiza-terminação-e-transforma,
            verifica-se-é-verbo.
```

```
verifica-se-é-verbo se /* verifica se o infinitivo do verbo está
                                                presente no dicionário */

                    [e-verbo-transitivo;
                     e-verbo-ligação;
                     e-verbo-intransitivo],
                    coloca-no-dicionário-novo-átomo.
```

5.3.2.2.3. Descrição do Processo TRADUTOR XG/PROLOG

A função deste bloco funcional é traduzir toda a notação em XG de parte da gramática da lingua portuguêsa como também dos dicionários do domínio e de propósito geral para regras PROLOG executáveis.

A implementação lógica deste processo é a seguir detalhada, e teve como referência o trabalho desenvolvido por Fernando Pereira na Universidade de Edinburgh. [2]

```
grammar-rule se
                    grammar-head,
                     [` --> '],
                    grammar-body,
                    armazena (grammar-head:- grammar-body).

grammar-head se
                    non-terminal.

grammar-head se  /* tratamento do símbolo `...' na gramática
                                                de extraposição */
                    non-terminal,
                     [`...'], gap,
                    non-terminal.

grammar-body se  grammar-body,
                  [` , '],
                  grammar-body.
```

```
grammar-body se  grammar-body,
                 [' ; '],
                 grammar-body.

grammar-body se  grammar-body-item

grammar-body-item se  [' | '].

grammar-body-item se  non-terminal.

grammar-body-item se  terminal.
```

5.3.2.2.4. Descrição do Processo ANALISADOR SINTÁTICO/SEMÂNTICO

A função deste bloco funcional é gerar uma forma lógica que represente adequadamente a semântica da frase interrogativa fornecida pelo usuário.

A implementação deste processo é uma consequência da aplicação do processo TRADUTOR XG/PROLOG sobre o dicionário ESPECIFICAÇÃO SINTÁTICO/SEMÂNTICO explicado no item 5.3.2.1.3. Basicamente este processo é o dicionário Especificação Sintático/Semântico implementado em linguagem Prolog.

5.3.2.2.5. Descrição do Processo TRANSFORMADOR LóGICA/REGRAS

Este bloco funcional é responsável pela transformação em regras Prolog executáveis da expressão lógica que representa a semântica da frase.

A implementação lógica deste processo é a seguir detalhada:

```
/* entende a expressão lógica L <== number-of(X,Q,N) */
   processa (L <== number-of(X,Q,N) se
                              converte(Q,R)
                              armazena(p(X) :-R),
                              set-of(X,p(X),T),
                              length(T,N),
                              write(N).

/* entende a expressão lógica L <== Q */
   processa (L <== Q) se
                     converter(Q,T),
                     armazena(L :- T),
                     set-of(X,L,Z),
                     write(Z).

/* transforma a expressão lógica ' X & Y ' em ' X, Y ' */
   converter (X&Y,(T1,T2)/ se
                          converter(X,(T1)),
                          converter(Y,T2)).
   converter (R,R).

/* transforma a expressão lógica '(X V Y)' em '(X;Y)' */
   converter ((X V Y), ((T1;T2))) se
                          converter (X,(T1)),
                          converter (Y,(T2)).

/* entende a expressão lógica exceeds (X,Y) em X > Y */
   exceeds(X,Y) se
                     X > Y.

/* entende a expressão lógica igual (X,Y) em X = Y */
   igual (X,Y) se
                     X = Y.
```

```
/* entende a expressão lógica ser (X,Y) */
   ser (X,Y)  se
                Y.
```

5.3.2.2.6. Descrição do Processo OTIMIZADOR

Este bloco funcional tem a função de tornar mais eficiente o conjunto de regras denotadas em linguagem Prolog.

A implementação lógica deste processo é a seguir detalhado:

predicado i (Ni,-,-,...), onde Ni representa a abrangência do domínio. Basicamente este processo coloca em ordem crescente, de abrangência de domínio, os predicados que compõem a parte condicional da regra.

5.3.2.2.7. Descrição do Processo EXECUTOR

Este bloco funcional deve adaptar/formatar as regras executáveis em prolog à Base de Dados a ser consultada e obter a resposta desejada. Neste software-produto como a Base de Dados já está no formato das regras originadas pelo bloco OTIMIZADOR, este processo apenas executa as regras sobre a Base de Dados obtendo desta forma a resposta desejada. Qualquer adaptação futura a outros padrões de Base de Dados deve ser realizada neste processo.

5.3.3. <u>Descrição de Dependência</u>

Esta descrição especifica o relacionamento de subordinação entre as diversas entidades que compõem este software-produto. Esta dependência é representada através de um diagrama HIPO-DB- "Hierarquy Input Process Output - Data Base" desenvolvido pela IBM para documentar a estrutura de sistemas, apresentado na figura 5.8.

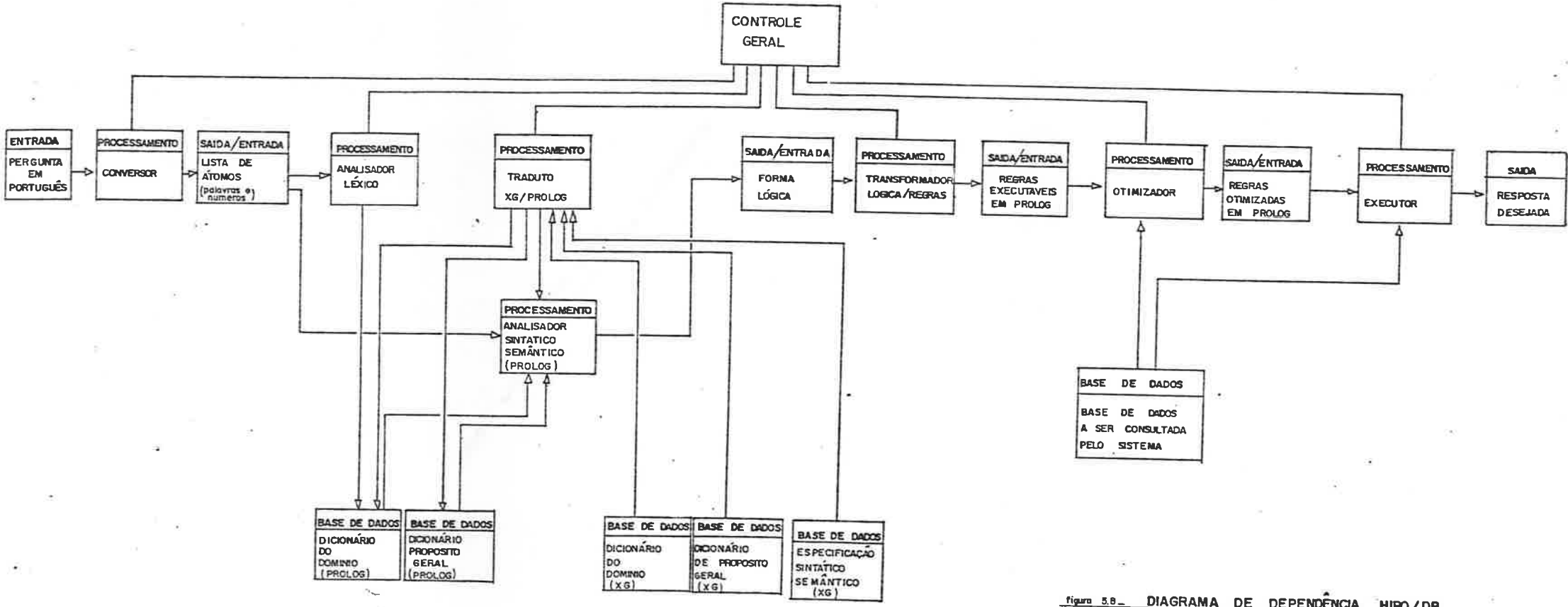


figura 5.8 – DIAGRAMA DE DEPENDÊNCIA HIPO/DB

5.3.4. Descrição da Interface do Software-Produto

Esta descrição apresenta a interface com o software-produto com todas as características necessárias para o usuário saber usar corretamente suas funções. Ela descreve a linguagem de comunicação com o sistema incluindo formato de telas, entradas válidas e saídas resultantes.

```
frase(X).
quais funcionarios estudam ?
quais funcionarios estudam
jose
paulo
pedro

Tempo de obtencao da forma logica = 0 min. 0 seg. 99 cent. seg.

Tempo total de resposta = 0 min. 1 seg. 37 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (estudar(_013D) & true
) & true ->c
yes
?- frase(X).
quantos funcionarios estudam ?
quantos funcionarios estudam
3
Tempo de obtencao da forma logica = 0 min. 0 seg. 38 cent. seg.

Tempo total de resposta = 0 min. 0 seg. 66 cent. seg.

X = answer(3) <= number_of(_013D,(funcionario(_013D) & true & true) & (estudar(_
013D) & true) & true,3) ->c
yes
?- frase(X).
quais funcionarios sao solteiros ?
quais funcionarios sao solteiros
carlos
jose
paulo

Tempo de obtencao da forma logica = 0 min. 0 seg. 76 cent. seg.

Tempo total de resposta = 0 min. 1 seg. 20 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (ser(_013D,solteiro(_0
13D,normal)) & true) & true ->c
yes
?- frase(X).
quais funcionarios admiram maria ?
quais funcionarios admiram maria
carlos
jose
pedro

Tempo de obtencao da forma logica = 0 min. 0 seg. 87 cent. seg.

Tempo total de resposta = 0 min. 1 seg. 26 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (admirar(_013D,maria)
& true) & true ->c
yes
```

```
frase(X).
quais funcionarios ganham igual a 35 ?
quais funcionarios ganham igual a 35
paulo

Tempo de obtencao da forma logica = 0 min. 1 seg. 27 cent. seg.

Tempo total de resposta = 0 min. 1 seg. 54 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (igual(35,salario(_013
D,_60A5)) & true) & true ->c
yes
?- frase(X).
quais funcionarios estudam ou admiram maria ?
quais funcionarios estudam ou admiram maria
carlos
jose
paulo
pedro

Tempo de obtencao da forma logica = 0 min. 1 seg. 97 cent. seg.

Tempo total de resposta = 0 min. 2 seg. 52 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (estudar(_013D) \/ adm
irar(_013D,maria) & true) & true ->c
yes
?- frase(X).
quais funcionarios nao sao solteiros e ganham mais de 30 ?
quais funcionarios nao sao solteiros e ganham mais de 30
pedro

Tempo de obtencao da forma logica = 0 min. 1 seg. 76 cent. seg.

Tempo total de resposta = 0 min. 2 seg. 3 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & true & true) & (not ser(_013D,solteir
o(_013D,normal)) & exceeds(salario(_013D,_9E41),30) & true) & true ->c
yes
?- frase(X).
quais funcionarios, que maria admira, sao solteiros ?
quais funcionarios que maria admira sao solteiros
carlos
jose
paulo

Tempo de obtencao da forma logica = 0 min. 1 seg. 26 cent. seg.

Tempo total de resposta = 0 min. 1 seg. 70 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & (admirar(maria,_013D) & true) & true)
 & (ser(_013D,solteiro(_013D,normal)) & true) & true ->c
yes
?- frase(X).
quais funcionarios, que maria admira e sao solteiros, estudam ?
quais funcionarios que maria admira e sao solteiros estudam
jose
paulo

Tempo de obtencao da forma logica = 0 min. 5 seg. 49 cent. seg.

Tempo total de resposta = 0 min. 5 seg. 87 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & (admirar(maria,_013D) & ser(_013D,sol
teiro(_013D,normal)) & true) & true) & (estudar(_013D) & true) & true ->c
```

```
frase(X).
quais funcionarios ,que ganham mais de 35 e estudam, nao sao solteiros ?
quais funcionarios que ganham mais de 35 e estudam nao sao solteiros
pedro

Tempo de obtencao da forma logica = 0 min. 7 seg. 25 cent. seg.

Tempo total de resposta = 0 min. 7 seg. 58 cent. seg.

X = answer(_0165) <= (funcionario(_0165) & (exceeds(salario(_0165,_C8BD),35) & e
studar(_0165) & true) & true) & (not ser(_0165,solteiro(_0165,normal)) & true) &
 true ->c
yes
?- frase(X).
quais funcionarios, que estudam e que ganham mais de 40, nao admiram maria?
quais funcionarios que estudam e que ganham mais de 40 nao admiram maria

Tempo de obtencao da forma logica = 0 min. 5 seg. 60 cent. seg.

Tempo total de resposta = 0 min. 5 seg. 87 cent. seg.

X = answer(_0165) <= (funcionario(_0165) & (estudar(_0165) & true) & (exceeds(sa
lario(_0165,_D795),40) & true) & true) & (not admirar(_0165,maria) & true) & tru
e ->c
yes
?- frase(X).
quais funcionarios,que maria admira ou nao ganham igual a 35, estudam e
sao solteiros?
quais funcionarios que maria admira ou nao ganham igual a 35 estudam e sao solte
iros
jose
paulo

Tempo de obtencao da forma logica = 0 min. 6 seg. 54 cent. seg.

Tempo total de resposta = 0 min. 6 seg. 98 cent. seg.

X = answer(_00ED) <= (funcionario(_00ED) & ((admirar(maria,_00ED) \/ not igual(3
5,salario(_00ED,_66F1)) & true) & true) & true) & (estudar(_00ED) & ser(_00ED,so
lteiro(_00ED,normal)) & true) & true ->c
yes
?- frase(X).
quais funcionarios que maria admira estudam ou sao solteiros ?
quais funcionarios que maria admira estudam ou sao solteiros
carlos
jose
paulo

Tempo de obtencao da forma logica = 0 min. 13 seg. 40 cent. seg.

Tempo total de resposta = 0 min. 13 seg. 84 cent. seg.

X = answer(_013D) <= (funcionario(_013D) & (admirar(maria,_013D) & true) & true)
 & (estudar(_013D) \/ ser(_013D,solteiro(_013D,normal)) & true) & true ->c
yes
```

5.3.5. Descrição do TRADUTOR-AUTOMÁTICO

Foi realizado também a implementação de um protótipo com o objetivo de verificar a eficiência de todo o formalismo, desenvolvido para a compreensão de sentenças, na utilização de um tradutor automático. Basicamente foi testada a eficiência de se obter uma sentença em português a partir de sua representação semântica, ou seja, a partir da sua "FORMA LóGICA" correspondente.

O Diagrama de Fluxo de Dados deste novo Software-Produto está apresentado na figura 5.9 e é composto de um subconjunto de blocos funcionais desenvolvido para a INTERFACE PARA BANCO DE DADOS EM LINGUAGEM NATURAL.

5.3.6. Descrição da Interface do TRADUTOR-AUTOMÁTICO

Esta descrição apresenta a interface com este TRADUTOR-AUTOMÁTICO com todas as características necessárias para o usuário saber usar corretamente suas funções. Ela descreve a linguagem da comunicação com o sistema incluindo formato de telas, entradas válidas e saídas resultantes.

```
frase(X,Y).
quais funcionarios estudam ?
quais funcionarios estudam
jose
paulo
pedro
Tempo de analise da sentenca = 0min.0seg.22cent. seg.

Tempo de sintese da sentenca = 0min.0seg.16cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (estudar(_0189) & true
) & true
Y = [quais,funcionarios,estudam] ->c
yes
?- frase(X,Y).
quantos funcionarios estudam ?
quantos funcionarios estudam
3Tempo de analise da sentenca = 0min.0seg.28cent. seg.

Tempo de sintese da sentenca = 0min.0seg.11cent. seg.

X = answer(3) <= number_of(_0189,(funcionario(_0189) & true & true) & (estudar(_
0189) & true) & true,3)
Y = [quantos,funcionarios,estudam] ->c
yes
?- frase(X,Y).
quais funcionarios sao solteiros ?
quais funcionarios sao solteiros
carlos
jose
paulo
Tempo de analise da sentenca = 0min.0seg.11cent. seg.

Tempo de sintese da sentenca = 0min.0seg.5cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (ser(_0189,solteiro(_0
189,normal)) & true) & true
Y = [quais,funcionarios,sao,solteiros] ->c
yes
?- frase(X,Y).
quais funcionarios admiram maria ?
quais funcionarios admiram maria
carlos
jose
pedro
Tempo de analise da sentenca = 0min.0seg.27cent. seg.

Tempo de sintese da sentenca = 0min.0seg.16cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (admirar(_0189,maria)
& true) & true
Y = [quais,funcionarios,admiram,maria] ->c
yes
```

```
frase(X,Y).
quais funcionarios que maria admira ?
quais funcionarios que maria admira
carlos
jose
paulo
Tempo de analise da sentenca = 0min.0seg.49cent. seg.

Tempo de sintese da sentenca = 0min.0seg.28cent. seg.

X = answer(_0189) <= (funcionario(_0189) & (admirar(maria,_0189) & true) & true)
 & (true & true) & true
Y = [quais,funcionarios,que,maria,admira] ->c
yes
?- frase(X,Y).
quais funcionarios ganham mais de 40 ?
quais funcionarios ganham mais de 40
carlos
pedro
Tempo de analise da sentenca = 0min.0seg.17cent. seg.

Tempo de sintese da sentenca = 0min.0seg.22cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (exceeds(salario(_0189
,_612D),40) & true) & true
Y = [quais,funcionarios,ganham,mais,de,40] ->c
yes
?- frase(X,Y).
quais funcionarios ganham menos de 60 ?
quais funcionarios ganham menos de 60
carlos
jose
paulo
Tempo de analise da sentenca = 0min.0seg.22cent. seg.

Tempo de sintese da sentenca = 0min.0seg.22cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (exceeds(60,salario(_0
189,_5EE9)) & true) & true
Y = [quais,funcionarios,ganham,menos,de,60] ->c
yes
?- frase(X,Y).
quais funcionarios ganham igual a 35 ?
quais funcionarios ganham igual a 35
paulo
Tempo de analise da sentenca = 0min.0seg.22cent. seg.

Tempo de sintese da sentenca = 0min.0seg.22cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (igual(35,salario(_018
9,_5E01)) & true) & true
Y = [quais,funcionarios,ganham,igual,a,35] ->c
yes
?- frase(X,Y).
quais funcionarios estudam e sao solteiros ?
quais funcionarios estudam e sao solteiros
jose
paulo
Tempo de analise da sentenca = 0min.0seg.49cent. seg.

Tempo de sintese da sentenca = 0min.0seg.61cent. seg.

X = answer(_0189) <= (funcionario(_0189) & true & true) & (estudar(_0189) & ser(
_0189,solteiro(_0189,normal)) & true) & true
Y = [quais,funcionarios,estudam,e,sao,solteiros] ->c
yes
```

```
?- frase(X,Y).
quais funcionarios ,que sao solteiros ou maria admira,estudam?
quais funcionarios que sao solteiros ou maria admira estudam
jose
paulo
Tempo de analise da sentenca = 0min.3seg.95cent. seg.

Tempo de sintese da sentenca = 0min.0seg.71cent. seg.

X = answer(_0189) <= (funcionario(_0189) & (ser(_0189,solteiro(_0189,normal)) \/
 admirar(maria,_0189) & true) & true) & (estudar(_0189) & true) & true
Y = [quais,funcionarios,que,sao,solteiros,ou,maria,admira,estudam] ->c
yes
?- frase(X,Y).
quais funcionarios,que maria admira ou nao ganham igual a 35,estudam e
sao solteiros ?
quais funcionarios que maria admira ou nao ganham igual a 35 estudam e sao solte
iros
jose
paulo
Tempo de analise da sentenca = 0min.4seg.73cent. seg.

Tempo de sintese da sentenca = 0min.1seg.21cent. seg.

X = answer(_0105) <= (funcionario(_0105) & ((admirar(maria,_0105) \/ not igual(3
5,salario(_0105,_677D)) & true) & true) & true) & (estudar(_0105) & ser(_0105,so
lteiro(_0105,normal)) & true) & true
Y = [quais,funcionarios,que,maria,admira,ou,nao,ganham,igual,a,35,estudam,e,sao,
solteiros] ->c
yes
?- frase(X,Y).
quais funcionarios, que maria admira, estudam ou sao solteiros ?
quais funcionarios que maria admira estudam ou sao solteiros
carlos
jose
paulo
Tempo de analise da sentenca = 0min.12seg.63cent. seg.

Tempo de sintese da sentenca = 0min.0seg.72cent. seg.

X = answer(_0161) <= (funcionario(_0161) & (admirar(maria,_0161) & true) & true)
 & (estudar(_0161) \/ ser(_0161,solteiro(_0161,normal)) & true) & true
Y = [quais,funcionarios,que,maria,admira,estudam,ou,sao,solteiros] ->c
yes
?- frase(X,Y).
quais funcionarios,que maria admira e sao solteiros,estudam?
quais funcionarios que maria admira e sao solteiros estudam
jose
paulo
Tempo de analise da sentenca = 0min.4seg.67cent. seg.

Tempo de sintese da sentenca = 0min.0seg.66cent. seg.

X = answer(_0161) <= (funcionario(_0161) & (admirar(maria,_0161) & ser(_0161,sol
teiro(_0161,normal)) & true) & true) & (estudar(_0161) & true) & true
Y = [quais,funcionarios,que,maria,admira,e,sao,solteiros,estudam] ->c
yes
```

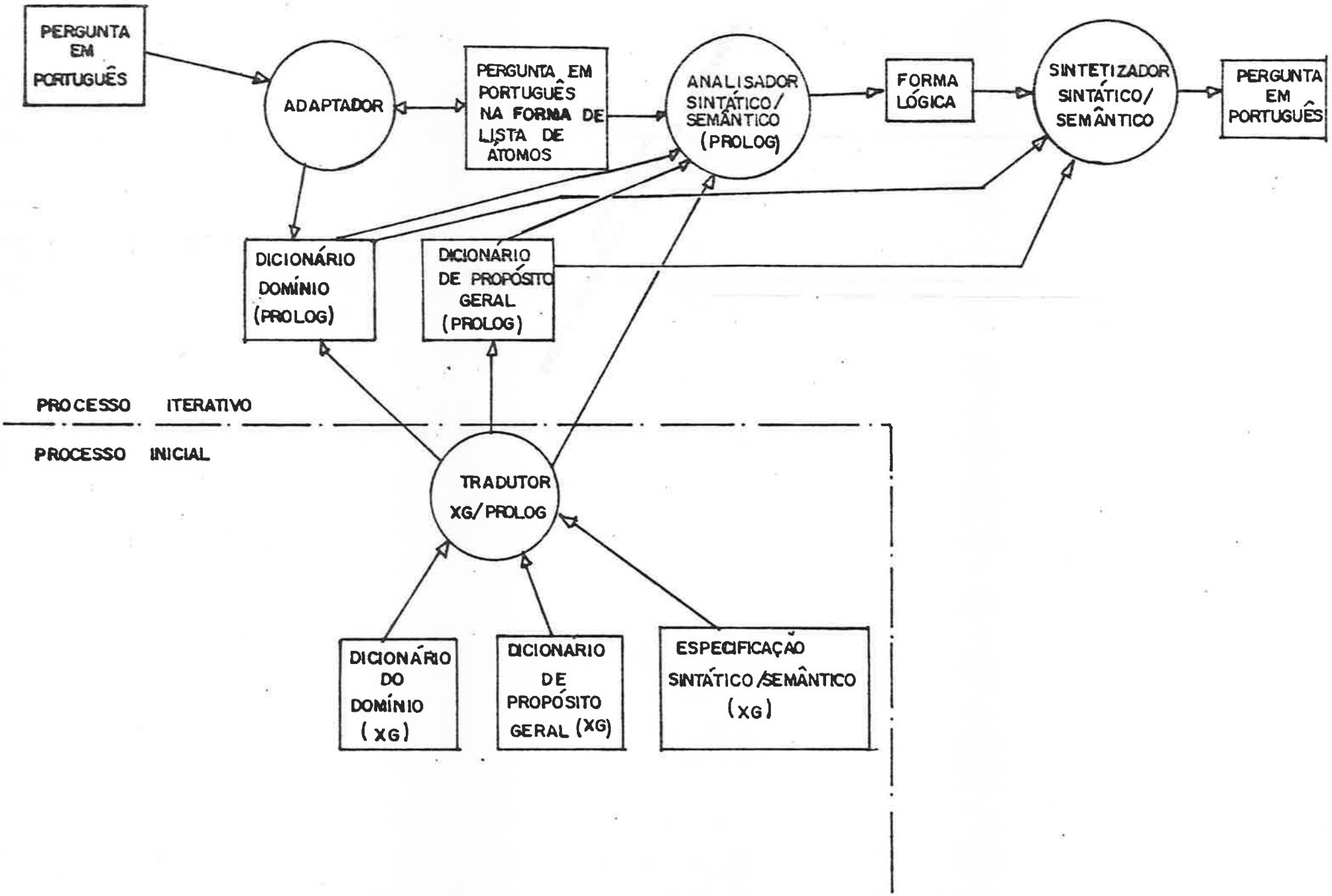


figura 5.9 - DIAGRAMA DE FLUXO DE DADOS DO TRADUTOR

6. AVALIAÇÃO

Neste capítulo é feita uma avaliação do protótipo desenvolvido nesta pesquisa. São a seguir apresentados e analisados, brevemente, os tópicos considerados mais relevantes do ponto de vista do trabalho realizado.

6.1. Interface Homem-Máquina e Aprendizado

Como primeiro tópico a ser considerado, pode-se indicar a interface deste sistema com seu usuário Para um usuário que seja leigo em computação e em linguística, esta interface homem-máquina apresenta uma relativa flexibilidade, visto que, apesar de permitir, quanto a sua abrangência, um número restrito de tipos de sentenças, pode-se dizer que estas são frequentemente utilizadas em interfaces do tipo diálogo pergunta-resposta.

No entanto, convém destacar dois aspectos importantes, quanto às implicações em futuras expansões do sistema, e que podem, de algum modo, tornar este software-produto mais eficaz em futuras aplicações. Um destes aspectos diz respeito a uma capacidade léxica maior do sistema. Devido à modularidade da estrutura deste sistema, como a existência de Dicionários independentes, Dicionário do Domínio e Dicionário de Propósito Geral, é possível, sem grandes dificuldades, incorporar ao sistema novos vocábulos aos dicionários, juntamente com suas interpretações semânticas.

Outro aspecto a se considerar nesta mesma linha é sua facilidade de assimilação de novas construções sintáticas. Novamente, devido à modularidade do sistema, é possível introduzir regras adicionais correspondentes às novas construções, no Dicionário Especificação Sintático/Semântico da língua portuguêsa.

Convém deixar claro que eventuais modificações, efetuadas neste dicionário, exigem do responsável por essas alterações, um mínimo de conhecimento na parte sintática e semântica da lingua

portuguêsa, tal como foi definida nas regras já incorporadas ao sistema, caso contrário estas modificações podem fazer com que o sistema absorva de forma incorreta algumas classes de sentenças, podendo passar, inclusive, a não mais aceitar outras classes de sentenças que o sistema já reconhecia

6.2. Pontuação

O segundo tópico importante, que pode ser, sem grandes dificuldades, incorporado ao sistema, é a capacidade de interpretação sintática da pontuação de um texto. Tal característica pode ser implementada através de dois métodos não exclusivos. De maneira geral poderia ser modificado o Dicionário Especificação Sintático/Semântico de forma a interpretar corretamente sinais de pontução. Neste caso, os sinais de pontuação seriam considerados como conectivos do ponto de vista da análise sintática. De maneira complementar, este trabalho poderia ser realizado no módulo Adaptador através da substituição da pontuação por palavras equivalentes que, na maioria dos casos, seriam conectivos, e que representariam de maneira correta o significado da pontuação dentro da sentença que está sendo analisada. Já no processo de Síntese de sentenças só haverá geração de sinais de pontuação quando isto for previsto no Dicionário Especificação Sintático/Semântico, ou então se houver uma "marca" nas palavras que devem ser substituidas por sinais de pontuação, que tenham a mesma interpretação semântica dentro da sentença.

6.3. Otimização de Regras

Como terceiro tópico verificou-se, durante o desenvolvimento deste trabalho, a conveniência de se criar um módulo denominado Otimizador que funcionalmente dever tornar mais eficientes as regras executáveis, em função da abrangência do assunto que se deseja cobrir. Assim sendo, é de grande importância que se possa

gerar automaticamente um módulo Otimizador a partir de uma Base de Dados que contenha as informações relativas ao assunto em questão. Na realidade, este objetivo é atingido através da criação de um Meta-Otimizador, cujo diagrama de fluxo de dados é apresentado na figura 6.1.

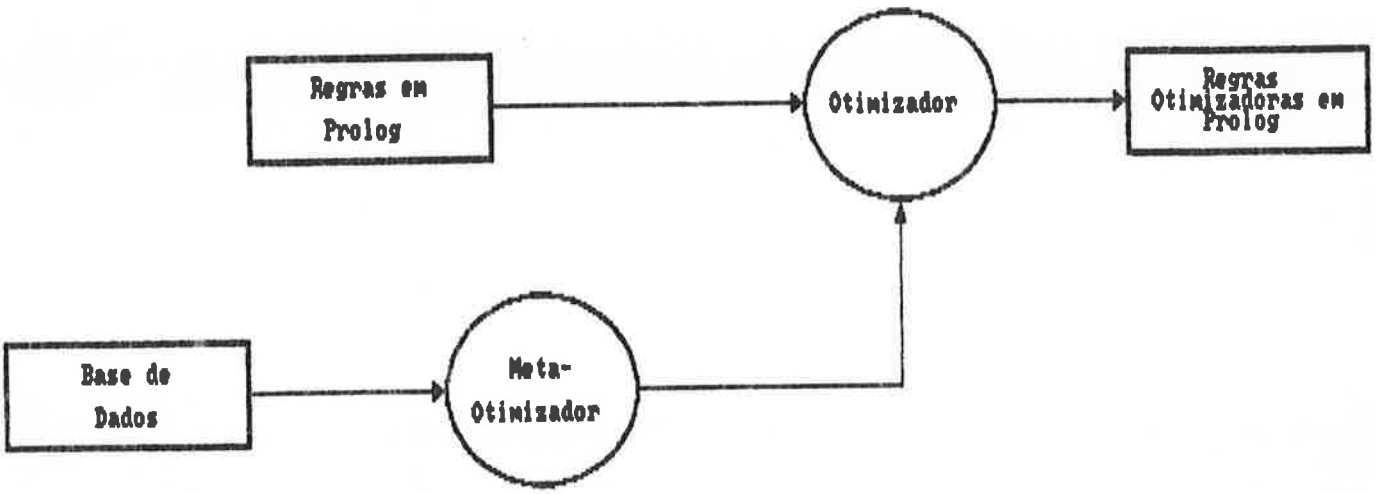


Figura 6.1 - Diagrama do Meta Otimizador.

Este Meta-Otimizador deve realizar uma pesquisa na Base de Dados, com a finalidade de avaliar a abrangência de cada conceito contido nesta Base, e cuja representação no sistema é feita através de predicados lógicos. Desta forma pode-se gerar automaticamente um Otimizador, que terá, para cada conceito, como parâmetros de entrada, indicadores do índice de abrangência do

correspondente predicado lógico. Assim, as regras executáveis podem ser reorganizadas de tal modo que, seja efetuada em primeiro lugar, a verificação relativa dos predicados, que exibam um menor índice de abrangêncida, de acôrdo com o indicador associado. Isto implementa uma heurística dirigida por informações semânticas, cuja finalidade é a de reduzir os tempos de busca, acelerando assim o processamento da frase em função do assunto a que estão relacionados os vocábulos que a compõem.

6.4. Ambiguidades [4], [36]

O quarto tópico diz respeito ao problema de ambuiguidades. Existem dois tipos de ambiguidade: ambiguidade léxica, sintática e semântica. A ambiguidade léxica pode ser resolvida através de um tratamento semântico adequado. Este tratamernto pode ser implementado com as restrições impostas na interpretação semântica das palavras que fazem parte dos dicionários do sistema. Desta forma, na frase "A manga da camisa está rasgada" a interpretação da palavra "camisa" apresentaria um atributo adicional dizendo que "manga" faz parte da "camisa". (faz-parte (manga, camisa)). Dentro deste contexto encontram-se frase do tipo:

. a manga está madura
. a manga está saborosa

A ambiguidade sintática, analogamente, pode ser caracterizada como a possibilidade de mais de uma interpretação sintática da mesma sentença, de acordo com uma dada gramática. Esta ambiguidade sintática pode levar a existência de uma ambiguidade semântica, já que cada sintaxe particular deve corresponder a uma semântica associada. Neste caso, o problema a ser resolvido é uma ambiguidade semântica.

Como exemplo deste tipo de ambiguidade, é apresentada a seguinte frase:

"O cachorro latiu para o gato andando no jardim."

Devido a existência de ambiguidade sintática, as possíveis interpretações desta frase são:

O cachorro latiu para (o gato andando no jardim)

$\Downarrow$

latiu (cachorro, gato) & andando (gato, jardim)

O cachorro latiu para (o gato) andando no jardim.

$\Downarrow$

latiu (cachorro, gato) & andando (cachorro, jardim)

Com relação a ambiguidade semântica, sua resolução é mais complexa. O esclarecimento da interpretação correta da frase é obtido através de um diálogo com o usuário ou através do conhecimento, por parte do sistema, do contexto que está sendo utilizado. Neste sentido existem formalismos tais como a Rede Semântica e a Dependência Conceitual, que associados ao formalismo desenvolvido neste trabalho, apresentariam, inclusive, a capacidade de permitir um maior rigor na interpretação de textos, embora nem todas as frase possam ser assimiladas sem a intervenção do operador.

Como exemplos de ambiguidades semânticas pode-se obter as seguintes frases:

. Esta manga é $\begin{cases} \text{verde} \\ \text{amarela} \end{cases} \longleftrightarrow$ a interpretação correta depende do contexto.

A interpretação de texto requer, na relaidade, uma cooperação mútua entre o conhecimento específico da língua, tanto em relação ao seu uso como à sintaxe, e o conhecimento do mundo em determinadas situações e eventos. A decisão final por uma interpretação mais provável pode ser realizada através do uso de uma heurítica, que permite a escolha de uma interpretação

particular, que apresente um maior índice de correção (peso). Estes pesos devem representar o grau de dependência entre as palavras, e os conceitos gerais envolvidos no texto [15]. Interações com o usuário permitem a determinação da interpretação correta caso o sistema não seja capaz de resolver certas dúvidas menos triviais.

Como exemplo, pode-se citar que a palavra "cassino" deve ter um peso muito maior quando encontrada em um texto relativo ao conceito "jogo" do que em um texto sobre "caça". De forma contrária, a palavra "floresta" deve ter um peso menor em textos sobre "jogo" do que em textos sobre o assunto "caça". Na realidade, estes pesos podem, inclusive, depender da cultura do grupo no qual o texto a ser interpretado se originou. Neste caso particular pode-se citar a existência de um vasto vocabulário, correspondete a diversas tonalidades peculiares de "branco", encontrada na cultura esquimó, e que em outras línguas perde a riqueza original por não encontrar equivalentes.

6.5. Transcrição de Discurso

O quinto tópico de possível aplicação do software-produto desenvolvido nesta pesquisa, é um "Gerador de texto escrito numa linguagem A, a partir de um discurso (fonético) na linguagem B". O esquema funcional desta aplicação é apresentado na figura 6.2.

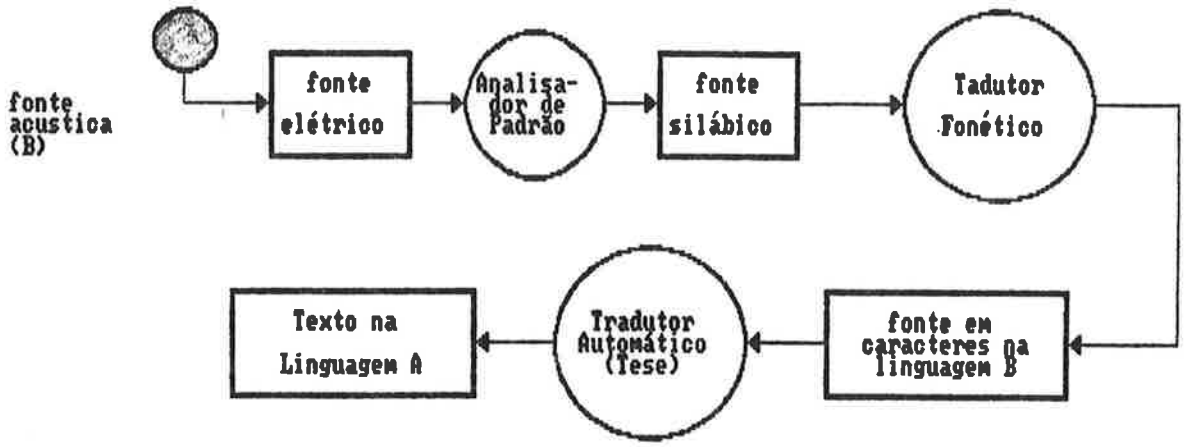


Figura 6.2 - Diagrama do Gerador de Texto a partir de um Discurso.

O Analisador de Padrão destina-se a transformar informações elétricas, que representam um discurso na linguagem B, em um texto silábico que representa, de forma correta, este discurso captado de uma fonte acústica.

O Tradutor Fonético é responsável pela conversão de um Fonte Silábico em um texto de caracteres que comporta palavras redigidas na mesma linguagem com que o discurso fonético foi proferido. Uma das possíveis implementações deste tradutor, é através da utilização de algoritmos eficientes de "pattern-matching". Neste aspecto, palavras são encontradas através do agrupamento de sílabas, e heurísticas, associadas com interações eventuais com o usuário, devem determinar qual das diversas combinações das interpretações obtidas deve ser considerada a mais correta em cada erro.

O Tradutor Automático [34] consiste em uma das mais importantes contribuições desta pesquisa. A principal característica deste sistema, além de sua capacidade léxica e sintática, é a obtenção de uma forma lógica que representa a interpretação do texto, fornecido na linguagem B. Em seguida, esta forma lógica é transformada em um outro texto denotado numa linguagem A. A forma lógica representa a semântica das frase e pode ser considerada como uma FORMA INTERMEDIÁRIA independente da linguagem a ser utilizada, conforme é mostrada na figura 6.3. Os dicionários utilizados pelo TRADUTOR 1 correspondem à linguagem A enquanto que os dicionários utilizados pelo TRADUTOR 2 correspondem à linguagem B.

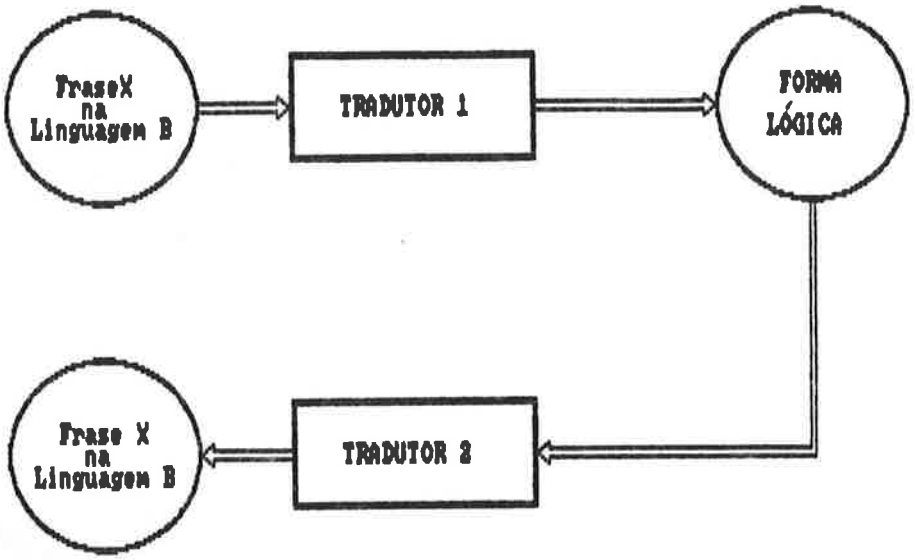


Figura 6.3 - Diagrama de Contexto do Tradutor Automático.

6.6. Revisão de Texto

Como sexto tópico de avaliação da pesquisa realizada neste trabalho, pode-se indicar sua aplicação, de forma racional, na Correção de Textos auxiliada por Computador. As palavras escritas de modo incorreto poderiam, através de heurísticas ou interações

com o usuário, serem substituídas por palavras com ortografia mais próxima possível, que tivesse sentido correto no contexto do texto fornecido.

Em seguida, para essas novas frases, seriam obtidas as formas lógicas que representam a interpretação semântica. Estas formas lógicas seriam verificadas do ponto de vista semântico, ou automaticamente, pelo sistema, ou através de um diálogo com o usuário.

6.7. APLICAÇÃO À ENGENHARIA DE SOFTWARE

O sétimo tópico consiste na aplicação da linguagem natural na Engenharia de Software. Neste aspecto, desde que se tenha um ambiente de Especificação formalizado, pode-se representá-lo através de um modelo adequado, tal qual uma "Hierarquia de Frames". Os "Frames" são estruturas compostas por "slots", que representam as características de uma "frame", podendo inclusive representar por sua vez um novo "frame". Para tornar-se um modelo adequado, esta representação deve ou estar completa ou então ser preenchida, de tal forma que possa espelhar corretamente uma determinada especificação. A figura 6.4 apresenta um diagrama de contexto deste sistema.

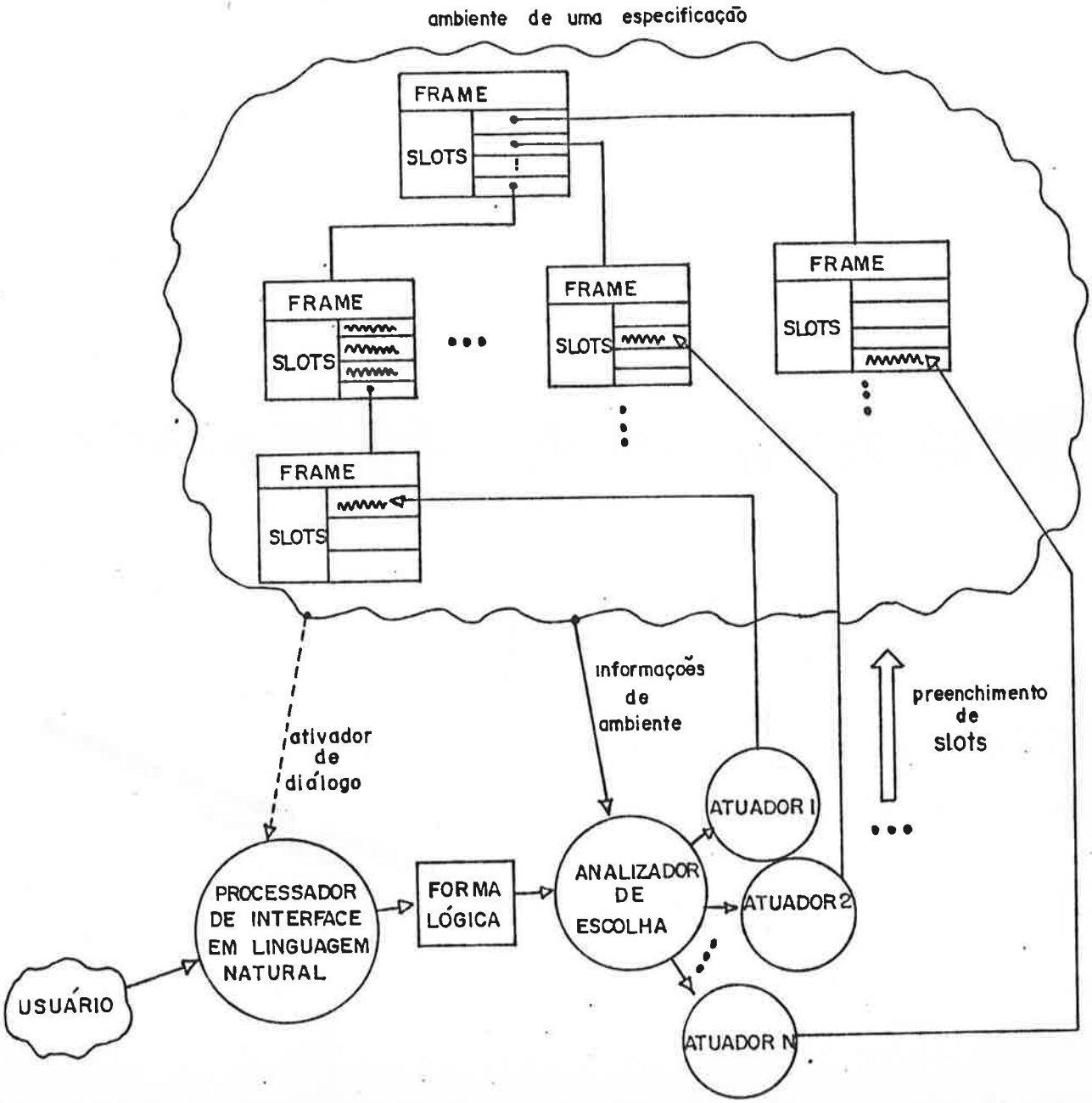


Figura 6.4 - Diagrama de Contexto.

Neste sistema, pode-se executar o processo de preenchimento de "slots" através da introdução por parte do usuário, de informações pertinentes. Estas informaçoes seriam fornecidas pelo usuário como respostas a uma requisição, na forma de diálogo,

promovida pelo sistema, ou então como resultado de uma atitude espontânea de fornecimento de informações, por parte do usuário. Estas infiormações são descritas através de um texto em linguagem natural. As ambiguidades léxicas e algumas características semânticas podem ser resolvidas pelo próprio PROCESSADOR DE INTERFACE EM LINGUAGEM NATURAL, pois este possui as informações de um Ambiente de uma Especificação. Pode-se chamar essas insconstâncias de "Contradições Linguísticas".

No entanto, existe um outro tipo de contradição, estruturada não na linguagem, mas sim na função que o texto procura transmitir, e que é denominada "Contradição Funcional".

Para que este sistema possa reconhecer uma Contradição Funcional é necessário, uma representação semântica adequada do conhecimento contido nos "slots". Assim sendo, torna-se importante, para promover esta tarefa, uma pesquisa mais abrangente com relação a outros formalismos não clássicos.

Como conclusão final desta avaliação, depreende-se que a forma de representação semântica das frases, constitui-se em um fator determinante da eficiência do sistema, dependendo do seu campo de aplicação. Neste aspecto, deve-se esclarecer que a expressão semântica adotada nesta pesquisa, que se constitui numa forma lógica clássica, é bastante adequada do ponto de vista de Interface com Base de Dados.

Para adaptar o software-produto desenvolvido nesta pesquisa, a outras representações semânticas, é necessário realizar modificações no Dicionário Sintático/Semântico, através da alteração dos atributos que representam as interpretações semânticas dos componentes das frases, e de atualizações das interpretações semânticas das palavras, contidas nos Dicionários de Domínio e de Propósito Geral.

6.8. Interface com Banco de Dados Reais [22]

O software-produto desenvolvido nesta pesquisa possibilita o acesso às informações contidas num Banco de Dados em prolog. Convém esclarecer, que é possível realizar este interfaceamento com um Banco de Dados Real, como por exemplo em DBASE, LOTUS, através da implantação adequada do módulo EXECUTOR.

Este módulo deve construir uma pergunta formal para o Banco de Dados considerado e se necessário, otimizar a pesquisa das informações neste banco. Esta pergunta formal deve representar a semântica da requisição do usuário e pode ser gerada a partir das regras em prolog ou da forma lógica, conforme a opção mais eficiente e adequada, conforme é mostrada na figura 6.5.

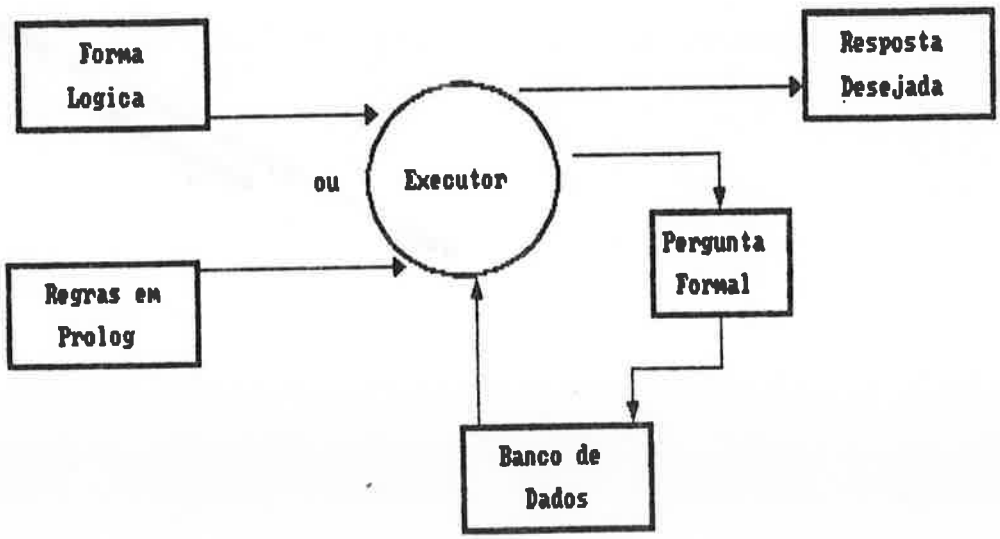


Figura 6.5 - Diagrama de Contexto do EXECUTOR.

Quando já existe uma "Structure Queery Language - SQL" relacionada com um determinado Banco de Dados, o módulo EXECUTOR pode mapear a semântica da frase de entrada na própria SQL ou interrogar diretamente o Banco de Dados.

O processo de geração desta Pergunta Formal envolve ações, tais como a localização das relações e dos domínios necessários à pergunta, além do mapeamento de tais informações em construções compatíveis com o Banco de Dados considerado. Estas construções são denominadas "templates" ou padrões [23].

Convém lembrar que, no ambiente Arity prolog utilizado no desenvolvimento do software-produto, existe condições de transferir registros de arquivos em LOTUS 1-2-3 e em DBASE e também existe uma SQL padronizada neste ambiente.

7. CONCLUSÕES

Neste capítulo são destacados os principais aspectos deste software-produto, bem como sua capacidade de assimilação de futuras expansões e aplicações em diversos campos.

Devido às principais características exibidas por este software-produto, INTERFACE COM BANCO DE DADOS EM LINGUAGEM NATURAL, pode-se concluir que ele atendeu plenamente às expectativas iniciais desta pesquisa. Seus sub-produtos potenciais são inúmeros e de grande utilidade, compondo toda uma linha de pesquisa na área.

Conforme foi mencionado no capítulo 1, esta dissertação está mais dedicada à formalização computacional básica da linguagem natural, no caso concreto a língua portuguesa, do que a problemas específicos abrangentes de seu campo de aplicação. Sob este enfoque convém destacar quatro aspectos importantes deste protótipo: facilidade de uso, potência, modularidade e flexibilidade.

A facilidade de uso deste software-produto, pode ser comprovada através da observação da simplicidade de suas características: tratamento adequado de sentenças que são normalmente utilizadas na língua portuguêsa, e simplicidade da técnica de utilização do protótipo. Quando discutido o aspecto potência, poder-se-á comprovar que as sentenças, que são processadas por este software-produto, constituem-se em sentenças de uso comum na lingua portuguêsa. É fácil constatar a simplicidade de utilização do sistema já que o usuário não necessita qualquer espécie de comando mais complexo para operar e interagir com o sistema.

A potência deste software-produto está relacionada com o seu poder de processar determinadas frases interrogativas complexas,

que incorporem qualquer tipo de encadeamento de frases subordinadas relativas.

Neste aspecto, convém destacar outras características como, por exemplo, a capacidade de manipular partículas tais como "nao", "e" e "ou", a capacidade de processar a flexão em número de substantivos e adjetivos, a capacidade de tratar a concordância verbal e formalização lógica adequada de adjetivos e advérbios, quando utilizados em interface com Base de Dados.

Para ilustrar o aspecto de potência deste software-produto, são apresentadas a seguir algumas sentenças que ele é capaz de processar léxica, sintática e semânticamente:

- Quais (Quantos) funcionários estudam?
- Quais (Quantos) funcionários que Maria admira?
- Quais (Quantos) funcionários admiram Maria?
- Quais (Quantos) funcionários estudam e são solteiros?
- Quais (Quantos) funcionários, que estudam e são solteiros, ganham mais de 40?
- Quais (Quantos) funcionários, que Maria admira e que ganham menos de 70, estudam e são solteiros?
- Quais (Quantos) funcionários, que ganham menos de 70, não admiram Maria?
- Quais (Quantos) funcionários estudam e não são solteiros?
- Quais (Quantos) funcionários estudam ou são solteiros?
- Quais (Quantos) funcionários, que estudam ou ganham mais de 40, admiram Maria?
- Quais (Quantos) funcionários, que são solteiros ou que estudam ganham (igual) 35?
- Quais (Quantos) funcionários estudam ou são solteiros ou ganham mais de 60 ou admiram Maria?
- Quais (Quantos) funcionários, que ganham menos de 70 ou são solteiros, não estudam e admiram Maria?

O processamento correto destas interrogações é apresentado ao usuário através da forma lógica, que representa a interpretação semântica da sentença, e da resposta correta a estas perguntas. Outra característica importante a se destacar no aspecto de Potência, diz respeito a utilização deste software-produto como TRADUTOR AUTOMÁTICO. Neste caso, a partir da forma lógica obtida, e que representa o significado da sentença, é obtida novamente a frase original.

A Modularidade está relacionada com o grau de independência dos processos que compõem este software-produto. Esta característica implica maior facilidade de modificação e adaptação, bem como aumento de sua capacidade de processamento da lingua portuguêsa. Estes são alguns dos fatores que colaboram na obtenção de maior flexibilidade para o sistema.

Os principais módulos deste software-produto e suas funções são:

- Adaptador: converter a sentença de entrada em uma lista de átomos onde cada átomo representa uma palavra, e interpretar léxica e semânticamente os átomos não primitivos.

- Tradutor XG/Prolog: traduzir para a linguagem Prolog toda a notação em XG que define parte da gramática da lingua portuguêsa, como também dos dicionários de domínio e de propósito geral.

- Analisador Sintático/Semântico: gerar uma forma lógica que represente adequadamente a semântica da frase interrogativa, fornecida ao sistema pelo usuário.

- Transformador Lógica/Regras: transformar a expressão lógica em regras executáveis da linguagem Prolog.

- Otimizador: tornar mais eficiente a execução na linguagem Prolog do conjunto de regras, anteriormente produzido.

- Executor: adaptar/formatar as regras executáveis em Prolog à Base de Dados a ser consultada e obter a resposta desejada.

- Dicionário de Domínio: deve conter todas as palavras "primitivas" que estão relacionadas com o domínio do assunto contido na Base de Dados a ser consultada.

- Dicionário de Propósito Geral: deve conter todas as palavras operadoras (por ex. mais, menos, igual, ...), conectivos lógicos (e, não, que, ou, ...) e quantificadores (quais, quantos, ...).

- Especificação Sintático/Semântico: deve conter uma especificação formal da linguagem a ser analisada, no caso o Português, e também o relacionamento das interpretações semânticas, das partes que constituem a frase a ser interpretada.

A flexibilidade deste software-produto pode ser encarada sob três enfoques: léxico, sintático e semântico. A capacidade de expansão léxica compreende, por exemplo, a manipulação da flexão das palavras em gênero e em grau, já que a flexão em número está incorporada no sistema. Como mencionado no Capítulo 5, estas

características podem ser facilmente implementadas, aumentando com isso a capacidade de manipulação léxica do sistema.

Uma maior abrangência da sintaxe está relacionada com a possibilidade de maior formalização da gramática portuguesa. Convém destacar, neste ponto que a lingua inglesa apresenta uma evolução maior neste sentido, em função de inúmero trabalhos já realizados na área, podendo-se destacar o histórico estudo de Chomsky [29]. A partir desta maior formalização, as regras sintáticas podem, sem grandes dificuldades, ser incorporadas no módulo responsável também pela tarefa de análise sintática.

Por outro lado, o aumento da capacidade semântica do sistema está intimamente ligado ao estudo das tranformações que as palavras podem sofrer conforme as conotações posicionais e temporais que se apresentam em um dado contexto. Além deste aspecto, existe um outro campo de grande importância na análise semântica, que corresponde ao estudo dos tipos de representação de conhecimento, que melhor se adaptem a esta finalidade. Neste ponto convém destacar os formalismos das Redes Semânticas, da Dependência Conceitual, dos Frames e das menos exploradas Lógicas não Clássicas.

Na realidade, o software-produto desenvolvido neste trabalho utiliza-se de uma Analisador Semântico dirigido por sintaxe. À medida que, se deseje tornar a análise menos dependente da sintaxe da frase, é possível desconsiderar alguns dos atributos e regras que foram criados especificamente para manipular tais características sintáticas. Convém lembrar que a desconsideração exagerada da estrutura sintática de uma frase pode levar a interpretações ambíguas ou incorretas.

Com relação aos demais projetos realizados na área do processamento de linguagem natural, a maioria dos trabalhos publicados está feito para a lingua inglesa.

Existem algumas características do software-produto desenvolvido nesta dissertação, que nem sempre estão presentes em publicações de outros trabalhos. Como exemplo, pode-se citar o tratamento de conectivos tais como "e" e "ou", bastante complexos quando utilizados em sentenças em linguagem natural, e a manipulação de adjetivos e adverbios. Outro aspecto muito importante a se destacar, é que este sistema engloba, conjuntamente, o tratamento léxico, sintático e semântico, enquanto a maioria dos trabalhos publicados na área, emprega outros tipos de modelamento. Convém destacar também outra característica deste sistema, a manipulação simultânea dos processos de análise sintática e semântica.

Finalizando, é importante destacar que, através do estudo dos formalismos utilizados neste trabalho, e da criação de um protótipo de teste para ser avaliado, possibilitou a obtenção de uma visão não tão informal do processamento da linguagem natural.

Este processo de estudo do processamento da linguagem natural deve prosseguir através de uma pesquisa, no sentido de se obter uma maior formalização sintática e semântica da lingua portuguêsa, além de procurar encontrar modelos mais adequados de representação e de manipulação correta do conhecimento.

Neste sentido é esperada uma aplicação mais objetiva e eficiente da linguagem natural em diversos campos: Entendimento de Texto, Tradução Automática e até mesmo na Engenharia de Software, já que desta maneira, a Visão Descritiva de uma especificação de software pode mostrar-se menos abstrata e um pouco mais normativa.

REFERÊNCIAS BIBLIOGRAFICAS

ARTIGOS

[1] - Barbara, J., Grosz, Appelt, Douglas E., Martin, Paul A., and Pereira, Fernando C.N.; TEAM : An Experiment in the Design of Transportable Natural - Language Interfaces, Artificial Intelligence, 32, 173-243; 1987.

[2] - Biermann, A. and Ballard, B.; Toward Natural Language Computation. American Journal of Computational Linguistic, 6(2): 71-86; 1980

[3] - Bruce, W. Ballard; A phrase - Structured Grammatical Frame Work for Transportable Natural Language Processing, Computational Linguistic, Vol. 10, nr.2, April-June 1984.

[4] - Dahl, V.; Translation Spanish into Logic Through Logic. American Journal of Computational Linguistic, 7, 3, 149-164, 1981.

[5] - Gallaire, H., Minker, J. and Nicolas, J.; Logic and Databases: a deductive approach, ACM Computation Survey. 15(2), 153-185; 1984.

[6] - Heidorn, G.E.; Automatic Programming Through Natural Language Dialogue: A Survey, IBM Journal of Research Development, July 1976.

[7] - Hendrix, G.G., E.D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a Natural Language Interface to Complex Data, ACM Transaction on Data-Base Systems 3, No. 2, 105-147; 1978.

[8] - Mc Cord, M.C.; Using Slots and Modifiers in Logic Grammars for Natural Language. Artificial Intelligence, 18, 3, 327-367; 1982.

[9] - Mc Cord, M.C., Slot Grammar, American Journal of Computational Linguistic, 6, 1980, 31-43.

[10] - Miller, L.A.; Natural Language Programming: Styles, Strategies, and Contrasts, IBM System Journal, Vol. 20, nr. 2, 1981.

[11] - Pereira, F.C.N.; Extraposition Grammars, American Journal of Computational Linguistic, 7, 4, 243-256; 1981.

[12] - Pereira, F.C.N. and Warren, D.H.D.; Definitive Clause Grammar for Language Analysis - a survey of the formalism and a comparison with augmented transition networks, Artificial Intelligence 13, 231-278; 1980.

[13] - Petrick, S.R.; On Natural Language Based Computer Systems, IBM Journal of Research Development, July 1976.

[14] - Plath, W.J.; REQUEST: A Natural Language Question Answering System, IBM Journal of Research Development 20, No. 4, 326-335; July 1976.

[15] - Pollack, Jordan and Waltz, David L. - Interpretation of Natural Language - 189 - February 1986 - BYTE.

[16] - Robison, J.; DIAGRAM: A Gramar for Dialogues. Communications of the ACM - 25(1); 27-47; 1982.

[17] - Sager, N. and Grishman, R.; The Restriction Language for Computer Grammars of Natural Language, Communications of the ACM, 18 : 390-400; 1975.

[18] - Sandewall, E.; Representing Natural Language Information in Predicate Calculus, Machine Intelligence 6, University of Edinburgh - 1971.

[19] - Schank, R.C., Lebowitz, M., Birnbaum, L.; An Integrated Understander, American Journal of Computational Linguistic, Vol.6, Nr. 01, Jan-March, 1980.

[20] - Schank, R.C.; Representation and Understanding of Text: Machine Intellingence 8; 1977.

[21] - Simmon, and Slocum - Generating English Discource from Semantic Network. Communication of the ACM, 15, 10, pg 891 - 1972.

[22] - Stonebraker, M., Wrong, E., Kreps, P. and Held, G., The Design and Implementation of INGRES, ACM Transaction on Database Systems 1, 3 (September), 189-222, 1976.

[23] - Waltz, David L., An English Language Question Answering System for a Large Relational Database, Communications of the ACM, July 1978, Vol.21, nr. 7.

[24] - Warren, David H.D. and Pereira, Fernando C.N.; An Efficient Easily Adaptable Systems for Interpreting Natural Language Queries: American Journal of Computational Linguistic, Vol.8, Nr. 34, July-December, 1982.

[25] - Warren, D.H.D., Pereira, L. and Pereira, F.; Prolog - The Language and its Implementation Compared with Lisp, ACM Symposium on Artificial Intelligence and Programming Languages, SIGPLAN, 8, 1977.

[26] - Warren, D.S., Friedman, J.; Using Semantics in non-context-free parsing of Montague Grammar, American Journal of Computational Linguistic, Vol.8, nr. 34, July-December, 1982.

[27] - Woods, W.A.; Trasition Network Grammars for Natural Language Analysis, Communication of the ACM 13, October, 591-606; 1970.

[28] - Woods, W.A.; Cascaded ATN Grammars, American Journal of Computational Linguistic, Vol.6, nr. 1, January-March, 1980.

LIVROS

[29] - Barr, Avron and Feigenbaum, Edward A.; The Handbook of Artificial Intelligence, Addison - Wesley Publishing Company, 1986, Vol. I.

[30] - Clocksin, W.F. and Mellish, C.S., Programming in Prolog, Springer-Verlag, 1987.

[31] - Dahl, V. and Saint-Dizier, P.; Natural Language Understanding and Logic Programming, North-Holland, 1984.

[32] - Kowalki, Robert; Logic for Problem Solving, North-Holland, 1979.

[33] - Mondin, Battista; Curso de Filosofia, Volume 1, Edições Paulinas.

[34] - Rauch - Hindin, Wendy B.; Artificial Intelligence in Business, Science, and Industry; Volume I: Fundamentals, Prentice-Hall, 1986.

[35] - Sager, Naomi; Natural Language Information Processing, Addison-Wesley Publishing Company, 1984.

[36] - Savadovsky, Pedro; Introdução ao Projeto de Interfaces em Linguagem Natural, Projeto Estra, SID Informática, 1987.

[37] - Tremblay, Sorenson - The Theory and Practice of Compiles - Mc - Graw - Hill Book Company - 1985.

NORMAS

[38] - FDTE nr. 900.000.NI.002 - Diretrizes Básicas para Especificação dos Requisitos de Software - ERS.

[39] - FDTE nr. 900.000.NI.005 - Diretrizes Básicas para Descrição do Projeto de Software - DPS.

APÊNDICE A

DEPENDÊNCIA CONCEITUAL

[19], [20], [29]

A dependência Conceitual é uma teoria desenvolvida por Roger Schank para representar o significado de sentenças em linguagem natural de maneira a facilitar inferências a ser independente da linguagem na qual as sentenças foram originalmentes escritas.

De maneira não similar às redes semânticas, que fornecem uma estrutura na qual os nós, representando informações em algum nível, podem ser alocados, e dependência conceitual apresenta uma estrutura e um conjunto de primitivas sobre as quais a representação de porções particulares de informações podem ser construídas.

A Teoria da Dependência Conceitual compõe-se de um conjunto de quatro categorias de AÇÃO para representar os conceitos:

- Categoria Instrumental
- Categoria Física
- Categoria Mental
- Categoria Global

Antes de explicar em detalhes as primitivas de cada uma das categorias convém esclarecer alguns formalismos desta teoria como suas diversas relações semânticas e seus objetos.

Didaticamente convém apresentar primeiramente os objetos das relações semânticas:

ACT's : Ações
PP's : Objetos
AA's : Modificadores de Ações
PA's : Modificadores de PP's

A seguir são apresentas as diversas relações semânticas que são utilizadas pelas diversas categorias de AÇÃO.

Relação 1 : .PP ⟨▬⟩ ACT, descreve a relação entre um ator e o evento que este ator causa.

Relação 2 : .PP ⟨▬⟩ PA, descreve a relação entre um PP e um PA. O PA neste caso funciona como um qualificador de PP assumindo inclusive valores numéricos.

Relação 3 : .PP ⟨≡⟩ PP, descreve a relação entre dois PP's sendo que um dos quais pertence ao conjunto definido pelo outro.

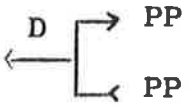
Relação 4 : .PP ↑ PA , descreve a relação entre um PP e um atributo que já lhe foi conferido anteriormente.

Relação 5 : .PP ↑ PP , descreve a relação entre dois PP's, um dos quais tem um tipo particular de informação sobre o outro. Os três tipos mais comuns nesta relação são posse (POSS-BY), localização (LOC) e limitação física (CONT).

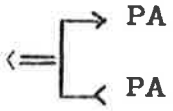
Relação 6 : .ACT ⟨—o— PP , descreve a relação entre uma ACT e um PP que se constitue no objeto desta ação.

Relação 7 : .ACT ⟨—R→ PP, ⟨ PP , descreve a relação entre uma ACT e a fonte e o recipiente da ação.

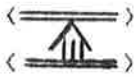
Relação 8 : .ACT ⟨—I— ⇕ , descreve a relação entre uma ACT e o instrumento com o qual esta ação se realizou.

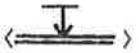
Relação 9 : ACT 

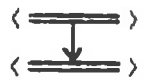
, descreve a relação entre uma ACT e sua fonte e destino físicos.

Relação 10 : PP 

, representa a relação entre um PP e um estado inicial e outro final.

Relação 11 : 
, descreve a relação entre uma conceituação* e um outro que o causou. A conceituação causa coprresponde à parte superior da figura.

Relação 12 : 
, descreve a relação entre uma conceituação e o tempo no qual ela ocorreu.

Relação 13 : 
, descreve a relação entre duas conceituações que ocorrem no tempo da primeira conceituação (parte superior da figura).

Relação 14 :

, descreve a relação entre uma conceituação e o local que ela ocorre.

(*) Entende-se por conceituação ao conjunto, que representa uma idéia, formado por uma relação e seus objetos.

A seguir são apresentadas as diversas primitivas englobadas dentro de cada categoria de ação da Dependência Conceitual.

a) As ações da Categoria Instrumental são:

. ATTEND : ação que tem os orgãos dos sentidos como objeto e a localização física como direção.

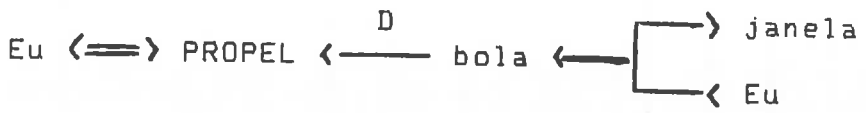
Ex.: { ATTEND olhos I é ver
ATTEND ouvidos I é ouvir

. SPEAK : ação que produz sons e seus objetos são sempre sons.

b) As ações da Categoria Física são:

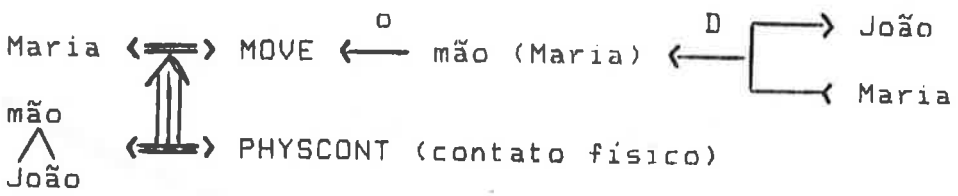
. PROPEL : ação que significa "aplicar uma força a" e seu objeto deve estar sob certas características de tamanho e peso. No exemplo, tal qualificação não será considerada.

Ex.: Eu joguei a bola na janela.



. MOVE : ação que significa "mover uma parte do corpo" e os únicos objetos são as partes do corpo.

Ex.: Maria tocou João com sua mão.



. INGEST : ação que significa "ingerir alguma coisa" e seu objeto deve ser menor do que a boca do ator, ou deve ser dividido em pedaços menores, que caibam na boca.

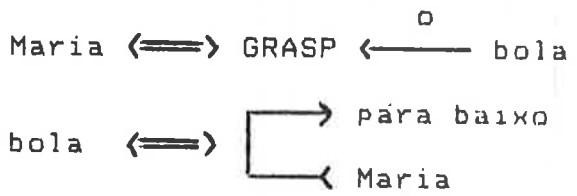
Ex.: Maria comeu peixe.

. EXPEL : ação que significa "tirar alguma coisa de dentro de voce e forçar para fora".

Ex.: Maria chorou.

. GRASP : ação que significa "agarrar" e seu objeto deve estar dentro de um limite de tamanho.

Ex.: Maria derrubou a bola.



c) As ações da Categoria Mental, que representam os caminhos através dos quais os pensamentos chegam a nossa mente, são:

MTRANS : ação que representa uma mudança no controle mental de uma conceituação e engloba verbos tais como relembrar, perceber e comunicar. O objeto desta ação não deixa o controle do dono, mas é copiado no controle do recipiente. O dono e o recipiente nem sempre são pessoas diferentes, mas são sempre processadores mentais diferentes.

Existem cinco processadores que são utilizados:

1. Processador Consciente - PC : local onde tudo é correntemente processado.

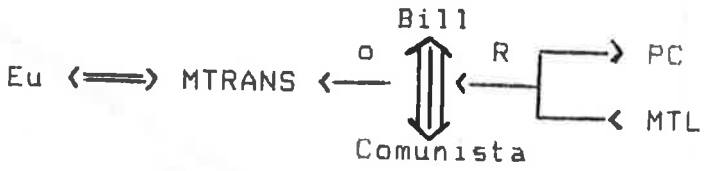
2. Memória de Tempo Longo - MTL : local onde as ações de esquecimento e associação do subconsciente acontecem. Deve ser tratado como um elemento passivo devido ao baixo nível de atividade.

3. Memória Imediata - MI : local onde o contexto está sendo processado.

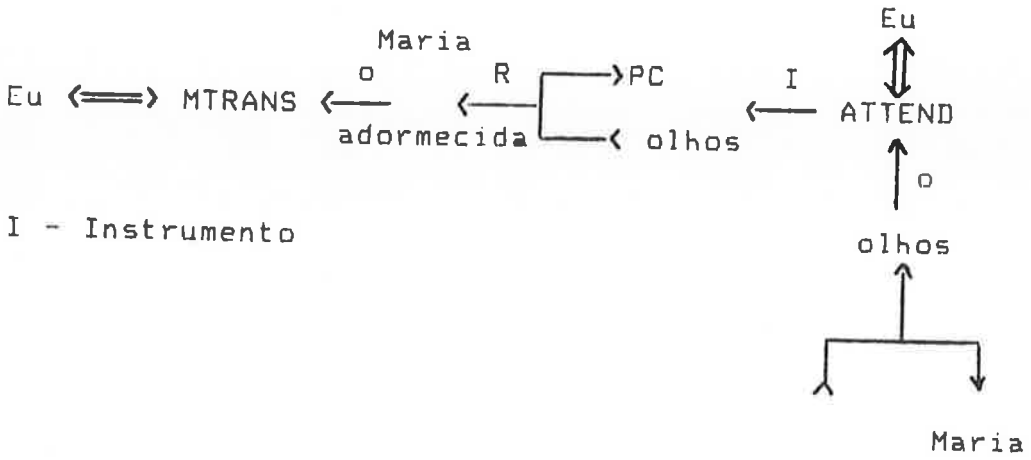
4. Orgão dos Sentidos (Visão, Audição, Olfato, Paladar e Tato) : processadores convertendo os dados sensoriados em dados descritos através de uma conceituação.

5. Corpo : este processador manipula sensações internas, tais como dor, excitação, tristeza, etc...

Ex: . Eu lembrei que Bill era comunista.

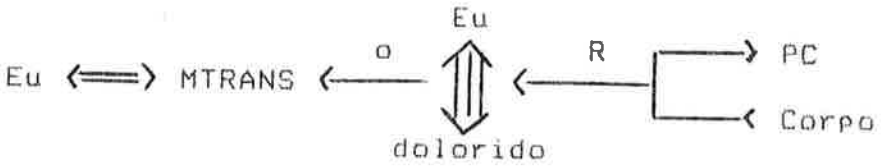


. Eu vi Maria dormindo.

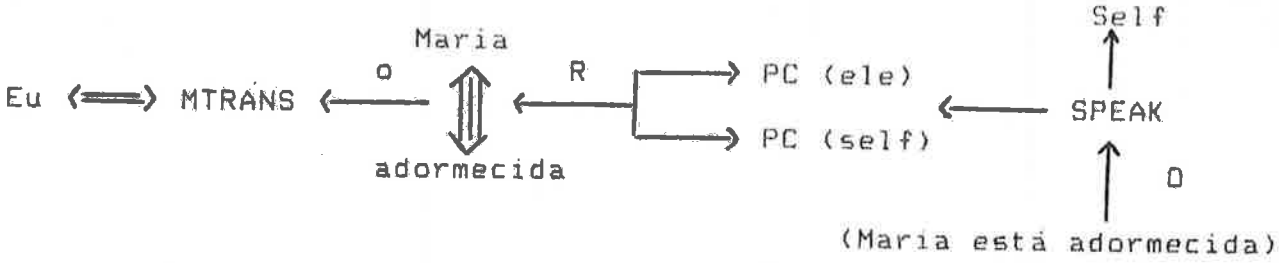


I - Instrumento

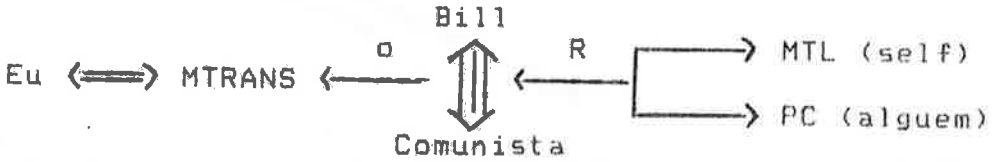
. Eu sinto dor.



. Eu disse a ele que Maria estava adormecida.

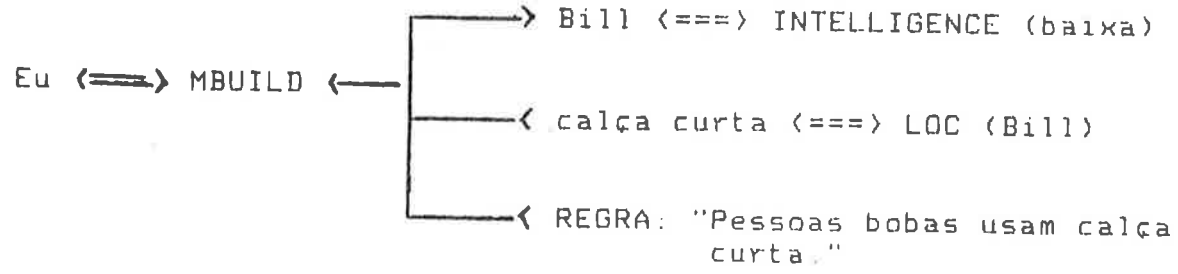


. Eu pensava que Bill era comunista

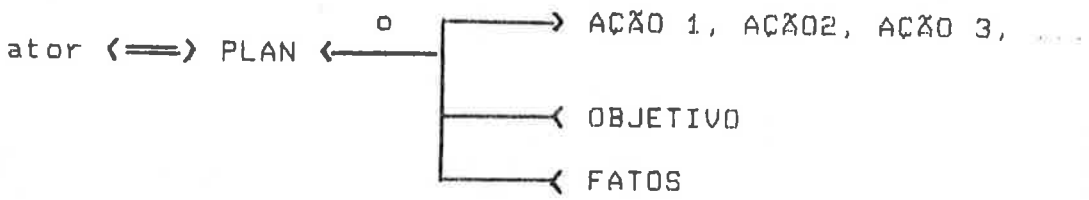


MBUILD: ação que significa combinação de pensamento e não inclue nenhuma informação do sucesso ou falha da operação.

Ex: Eu decidi que Bill era bobo porque ele usava calça curta.



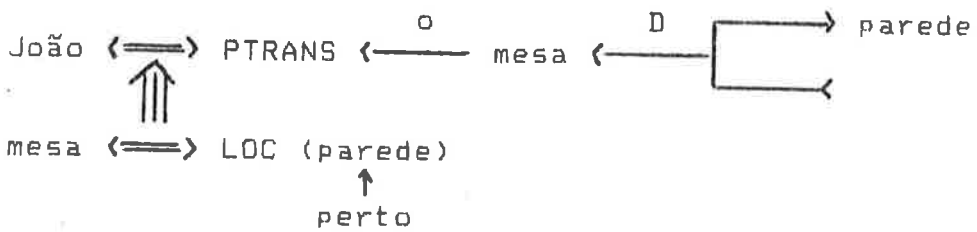
PLAN: ação que tem como entrada um objetivo e um conjunto de planos possíveis que podem ser requeridos para atingir o objetivo. O PLAN decide entre os planos possíveis e produz como saída uma sequencia de ações que o ator tenciona realizar para conseguir o seu objetivo.



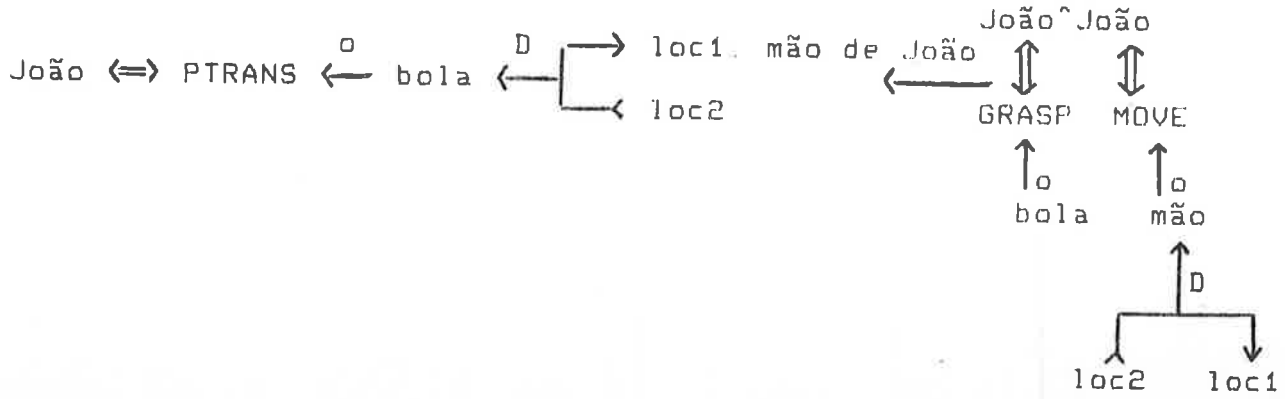
d) A Categoria Global exprime ações que representam uma mudança de estado, consequência de uma variável de ação física. Suas ações são:

PTRANS: ação que exprime a mudança na localização física de um objeto.

Ex: . João moveu a mesa para perto da parede.

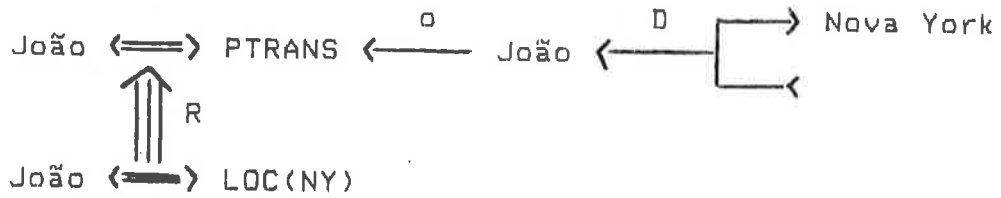


. João pegou a bola.



Obs: loc1 é mais alto do que loc2.

. João foi para Nova Iork.



ATRANS: ação que opera sobre relações abstratas e seu instrumento físico raramente é especificado. O ATRANS tem como objeto a relação abstrata entre dois objetos reais.

Ex: . Maria deu o livro para João.

. Maria emprestou o livro para João.

. João deu seu carro à Bill.

Esta explicação da Dependência Conceitual teve como principal objetivo mostrar a complexidade envolvida no processo de entendimento de texto. O número de variáveis e conceitos se multiplica de maneira bastante grande para englobar um conjunto razoável de idéias.

APÊNDICE B

REDE DE TRANSIÇÃO ESTENDIDA - ATN ("Augmented Transition Network") [27], [28]

O modelo desenvolvido por Woods e apresentado no item 3.2 constitue-se na chamada Rede de Transição Estendida - ATN.

Esse novo modelo adiciona recursos ao modelo de rede de transição, tais como, a possibilidade de mover fragmentos da estrutura da sentença, copiar e remover fragmentos da estrutura da sentença e realizar ações sobre seus constituintes, dependendo do contexto em que ocorram. Tais ações são implementadas adicionando a cada arco da transição uma condição arbitrária que deve ser satisfeita para que o arco possa ser percorrido.

A linguagem de especificação de uma rede ATN é apresentada a seguir em notação BNF:

```
<rede de transição> -> (<conjunto de arcos><conjunto de arcos>*)
<conjunto de arcos>  -> (<estado><arco>*)
<arco> -> (CAT <nome da categoria><teste><ação>*<ação final>)
          (PUSH <estado><teste><ação>*<ação final>)
          (TST <rótulo arbitrário><teste><ação>*<ação final>)
          (POP <forma><teste>)
```

A palavra <teste> representa um teste arbitrário que deve ser satisfeito para que o arco seja percorrido.

As palavras CAT, PUSH, TST e POP constituem os diversos tipos de arcos, e são explicados abaixo:

CAT: o arco pode ser percorrido se o símbolo de entrada for um membro da categoria léxica indicada no arco (e se o <teste> for satisfeito);

PUSH: o arco causa um desvio para o estado indicado; deve-se retornar à rede principal a forma da estrutura.

TST: arco que permite um teste arbitrário para determinar se ele pode ser percorrido;

POP: uma arco "dummy" que indica sob quais condições o estado pode ser considerado um estado final e a estrutura que deve ser retornada neste caso.

```
<ação> -> (SETR  <registro> <forma>)|
          (SENR  <registro> <forma>)|
          (LIFTR <registro> <forma>)
```

que corresponde à ação de construção de estruturas.

Estas ações causam a igualdade do valor indicado na <forma> ao conteúdo do <registro>.

SETR: a igualdade é realizada no nível corrente de computação na rede.

SENDR: a igualdade é realizada no próximo nível inferior de computação. Utilizada para enviar informações ao nível inferior de computação.

LIFTR: a igualdade é realizada no próximo nível superior de computação. Utilizada para retornar informações adicionais ao nível superior de computação.

```
<forma> -> (GETR <registro>)|
            *|
           (GETF <característica>)|
           (BUILDQ <fragmento> <registro>*)|
           (LIST <forma>*)|
           (APPEND <forma> <forma>)|
           (QUOTE <estrutura arbitrária>)
```

corresponde a uma função arbitrária do conteúdo do registro, representada em alguma linguagem de especificação funcional (tal como o LISP).

GETR: uma função cujo valor é o conteúdo do registrador indicado;

* : é uma forma cujo valor é geralmente a palavra de entrada corrente;

GETF: uma função que determina o valor de uma característica específica para a palavra de entrada corrente;

BUILDQ: uma forma de construção de estrutura bastante útil que manipula uma estrutura, <fragmento>, com alguns nós especiais, e retorna como resultado a substituição destes nós pelos conteúdos dos registros indicados. Especificamente, para cada ocorrência do simbolo ' + ' na estrutura dada como primeiro argumento, BUILDQ o substitue pelos conteúdos de cada um dos registros listados. Além disso, BUILDQ substitue a ocorrência do símbolo ' * ' no fragmento pelo valor corrente de ' * ';

LIST: faz uma lista dos valores dos seus argumentos;

APPEND: junta duas listas numa única lista;

QUOTE: produz como valor a forma do seu argumento.

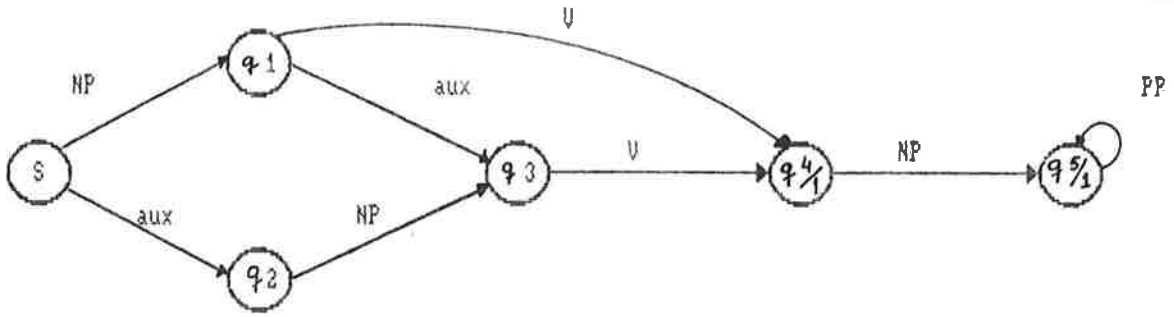
<ação final> -> (TO <estado>)|
(JUMP <estado>)

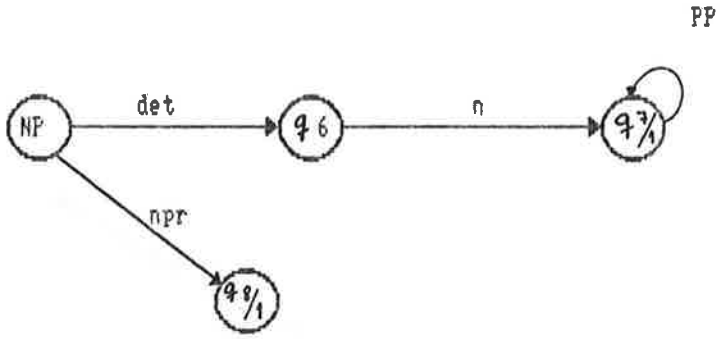
especifica o estado para o qual o controle é passado como resultado da transição.

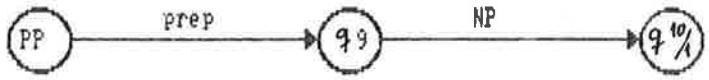
TO: indica que o ponteiro de entrada deve ser avançado, ou seja, o próximo estado vai analisar a próxima palavra de entrada;

JUMP: indica que o ponteiro de entrada não deve ser avançado, ou seja, o próximo estado vai continuar analisando a mesma palavra.

A seguir é apresentado um exemplo da aplicação do formalismo ATN em um subconjunto da gramática inglesa.








```
(S/   (PUSH NP/ T
          (SETR SUBJ *)
          (SETR TYPE (QUOTE'DCL)) <———— "DCL uma declaração"
          (TO Q1))
      (CAT AUX T
          (SETR AUX *)
          (SETR TYPE (QUOTE Q)) <———— "Q indica uma `querry'"
          (TO Q2)))
```

```
(Q1  (CAT  V  T
          (SETR AUX NIL)
          (SETR  V   * )
          (TO Q4))
     (CAT AUX  T
          (SET AUX *)
          (TO  Q3)))

(Q2  (PUSH  NP/T
          (SETR SUBJ  *)
          (TO  Q3)))

(Q3  (CAT  V  T
          (SETR  V  *)
          (TO Q4)))

(Q4  (POP (BUILDQ(S + + + (VP+)) TYPE SUBJ AUX V) T)
     (PUSH NP/T
          (SETR VP (BUILDQ (VP (V+) *) V))
          (TO  Q5)))

(Q5  (POP (BUILDQ (s + + + +) TYPE SUBJ AUX VP T)
     (PUSH PP/T
          (SETR VP (APPEND (GETR VP) (LIST *)))
          (TO  Q5)))
```

Agora é mostrado o funcionamento desta rede para a sentença de entrada: "Does John like Mary?"

1. O processo inicia-se no estado S/ analisando a primeira palavra da sentença, "Does". Já que esta palavra é um auxiliar, o arco (CAT AUX ...) é seguido. Em seguida são executadas as seguintes ações:

   - (SETR AUX *): coloca a palavra corrente "Does" no registro chamado AUX.

- (SETR TYPE (QUOTE Q)) coloca o símbolo "Q" (pergunta) no registro chamado TYPE.

- (TO Q2): causa a rede entrar no estado Q2 analisando a próxima palavra "John".

2. A partir do estado Q2 executa-se uma "PUSH" e uma representação da estrutura da frase substantiva (NP/T) irá retornar. Tal representação será o valor do registro ' * '. ("NP John"). Em seguida são executadas a seguintes ações:

   - (SETR SUBJ *): causa a alocação do valor "(NP John)" ao registro SUBJ.

   - (TO Q3): causa a rede ir para o estado Q3 analisando a próxima palavra "like".

3. Do estado Q3, o verbo "like" permite a transição para o estado Q4 realizando as seguintes ações:

   - (SETR V *): coloca a palavra corrente "like" no registro chamado V.

   - (TO Q4): causa a rede entrar no estado Q4 analisando a próxima palavra "Mary".

4. O estado Q4 pode ser um estado final, mas tal procedimento não satisfaz esta sentença. é percorrido neste caso o arco (PUSH ...) retornando o valor "(NP Mary)" no registro *. Em seguida são executadas as seguintes ações:

   - (SETR VP (BUILDQ (VP (V+) *) V)): monta a estrutura

     "(VP(V like)(NP Mary))" que será alocada no registro VP

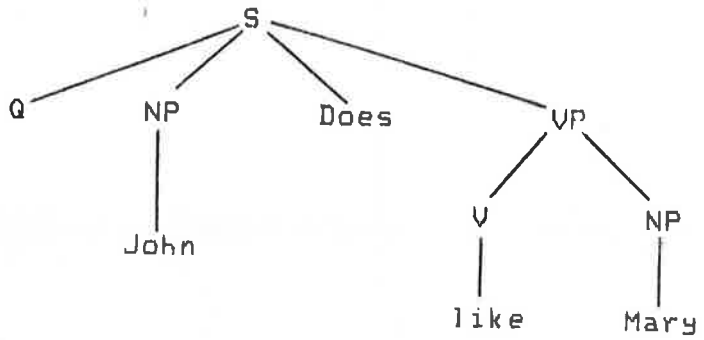
   - (TO Q5): causa a rede ir para o estado Q5 analisando a próxima palavra. Neste caso já se atingiu o caso final.

Assim sendo neste ponto o conteúdo dos registros são:

TYPE : Q
SUBJ : (NP John)
AUX : Does
VP : (VP (V like)(NP Mary))

5. Já que o estado Q5 é um estado final, é percorrido o arco "POP" e obtem-se a seguinte estrutura:

(S Q (NP John) Does (VP (V like)(NP Mary))) que representa a seguinte árvore:



Desta forma, o ATN apresenta o mesmo poder que uma gramática transformacional, mas mantém muitas das possibilidades do modelo da gramática livre de contexto.

Pode-se acrescentar também que uma das características mais interessantes no uso de registros em ATN é a habilidade de tomar decisões a respeito da estrutura da sentença e mudá-la posteriormente sem a necessidade de "back-tracking".

APÊNDICE C

EXECUÇÃO SIMBÓLICA DO FORMALISMO XG [11]

Neste apêndice é apresentada a execução simbólica do formalismo XG visando demonstrar sua capacidade de análise. Ao longo desta execução, está indicado, entre " [] ", o número da regra utilizada e que são apresentadas a seguir.

As regras em formato XG e em predicados lógicos são:

```
(XG)
 Sentence ---> noun-phrase, verb-phrase

(Predicados Lógicos)

(1) Sentence (S0,S,X0,X): -

                        noun-phrase (S0,S1,X0,X1)
                        verb-phrase (S1,S,X1,X)
```

```
(XG)

 noun-phrase --> proper-noun.

(Predicados Lógicos)

(2) noun-phrase (S0,S,X0,X): -

                    proper-noun (S0,S,X0,X)
```

```
    (XG)

     noun-phrase --> determiner, noun, relative.

    (Predicados Lógicos)

(3) noun-phrase (S0,S,X0,X): -

                    determiner(S0,S1,X0,X1),

                    noun (S1,S2,X1,X2),

                    relative (S2,S,X2,X).
```

```
    (XG)

     noun-phrase --> determiner, noun, prep-phrase

    (Predicados Lógicos)

(4) noun-phrase (S0,X0,X): -

                    determiner(S0,S1,X0,X1),

                    noun (S1,S2,X1,X2),

                    relative (S2,S,X2,X)
```

```
    (XG)

     noun-phrase --> trace.

    (Predicados Lógicos)

(5) noun-phrase (S,S0,X,X0):-

                    trace (S,S0,X0,X).
```

```
    (XG)
     verb-phrase --> verb, noun-phrase
    (Predicados Lógicos)
(6) verb-phrase (S0,S,X0,X):-
                    verb (S0,S1,X0,X1)
                    noun-phrase (S1,S,X1,X)
```

```
    (XG)
     verb-phrase --> verb.
    (Predicados Lógicos)
(7) verb-phrase (S0,S,X0,X):-
                    verb (S0,S,X0,X)
```

```
    (XG)
     relativs --> [ ]
    (Predicados Lógicos)
(8) relative (S0,X0,X,X)
```

```
    (XG)
     relative --> open, rel-marker, sentence, close.
    (Predicados Lógicos)
(9) relative (S0,S,X0,X):-
                    open (S0,S1,X0,X1).
                    rel-marker (S1,S2,X1,X2)
                    sentence (S2,S3,X2,X3)
                    close (S3,S,X3,X).
```

```
(XG)

 open ... close --> [ ].

(Predicados Lógicos)

(10) open (S0,S0,X,x(gap,nt,close,X)).
                    e
(11) close (S0,S0,X0,X):-

                    virtual (close,X0,X)
```

```
(XG)

 rel-marker ... trace --> rel-pronoun.

(Predicados Lógicos)

(12) rel-marker (S0,S,X0,x(gap,nt,trace,X)):-

                    rel-pronoun (S0,S,X0,X)

                    e

(13) trace (S0,S0,X0,X):-

                    virtual (trace,X0,X).
```

```
(XG)

 prep-phrase --> preposition, noun-phrase.

(Predicados Lógicos)

(14) prep-phrase (S0,S,X0,X):-

                    preposition (S0,S1,X0,X1),

                    noun-phrase (S1,S,X1,X).
```

O vocabulário a seguir utilizado foi montado para poder analisar a seguinte frase, anteriormente analisada:

The boy that the girl that liker chocolate kissed laugh.

```
(XG)
 determiner --> [the].
      (Predicados Lógicos)
(15) determiner (S0,S,X0,X):-
                    terminal (the,S0,S,X0,X).
```

```
(XG)
 determiner --> [ ].
(Predicados Lógicos)
(16) determiner (S0,S0,X,X).
```

```
(XG)
 noun -> [boy].
 noun -> [girl].
 noun -> [chocolate].
(Predicados Lógicos)
(17) noun (S0,S,X0,X):-
            terminal (boy,S0,S,X0,X)
(18) noun (S0,S,X0,X):-
            terminal (girl,S0,S,X0,X).
(19) noun (S0,S,X0,X):-
            terminal (chocolate,S0,S,X0,X).
```

```
     (XG)
      rel-pronoun -> [that].
     (Predicados Lógicos)
(20) rel-pronoun (S0,S,X0,X):-
                        terminal (that,S0,S,X0,X).
```

```
     (XG)
      verb -> [likes].
      verb -> [kissed].
      verb -> [laugh].
     (Predicados Lógicos)
(21) verb (S0,S,X0,X):-
                    terminal (likes,S0,S,X0,X).
(22) verb (S0,S,X0,X):-
                    terminal (kissed,S0,S,X0,X).
(23) verb (S0,S,X0,X):-
                    terminal (laugh,S0,S,X0,X).
```

O predicado "terminal", como já havia sido explicado anteriormente, é representado da seguinte forma:

```
(24) terminal (T,S0,S,X,X):-
                        gap (X),
                        connects (S0,T,S).
(25) terminal (T,S,S,x(c,t,T,X),X).
```

O predicado "gap" é representado como:

(26) gap (x (gap,T,S,X).

ou

(27) gap ([ ]).

O predicado "virtual" é representado por

(28) virtual (NT,x(_,nt,NT,X),X).

Os seguintes predicados indicam a posição das palavras nas sentenças:

(29) connects (1,the,2).
(30) connects (2,boy,3).
(31) connects (3,that,4).
(32) connects (4,the,5).
(33) connects (5,girl,6).
(34) connects (6,that,7).
(35) connects (7,likes,8).
(36) connects (8,chocolate,9).
(37) connects (9,kissed,10).
(38) connects (10,laugh,11).

É importante frisar que, para maior clareza, foram evitados todos os caminhos com "back-tracking", selecionando de imediato a regra adequada.

Nesta execução simbólica foi analisada a seguinte frase:

The boy that the girl that likes chocolate kissed laugh

Foram utilizadas as 38 regras apresentadas. A pergunta que deve ser feita para se iniciar o processo é:

```
? - sentence (1, 11, [ ], [ ]).

(1) sentence (1,11,[ ],[ ]) :-

                    noun-phrase (1, 10, [ ], X0)
                    verb-phrase (10, 11, X0, [ ])

(3) noun-phrase (1,10,[ ],X0) :-
                       determiner (1,2,[ ],X2),
                       noun (2,3,X2,X3),
                       relative (3,10,X3,X0).

(15) determiner (1,2,[ ],X2) :-
                      terminal (the,1,2,[ ],X2).

                                 [ ]
(24) terminal (the,1,2,[ ],X2) :-
                        gap ([ ]), <-- (27)
                        connects (1,the,2) <-- (28)

(17) noun (2,3,[ ],X3) :-
                 terminal (boy,2,3,[ ],X3).

                                 [ ]
(24) terminal (boy,2,3,[ ],X3) :-
                          gap ([ ]), <-- (27)
                          connects (2,boy,3) <-- (29)

(9) relative (3,10,[ ],X0) :-
                  open (3,3,[ ],X1),
                  rel-marker (3,4,X1,X2),
                  sentence (4,10,X2,X3),
                  close (10,10,X3,X0).

(10) open (3,3,[ ],X1)
                 x (gap,nt,close,[ ]).

                                    x (gap,nt,trace,X4)
```

```
(12) rel-marker (3,4,x (gap,nt,close,[ ]), X2) :-
                        rel-pronoun (3,4,x (gap,nt,close,[ ]),X4).

(20) rel-pronoun (3,4,x (gap,nt,close,[ ]),X4) :-
                      terminal (that,3,4,x (gap,nt,close,[ ]),X4).

                                                x(gap,nt,close,[ ])
(24) terminal (that,3,4,x (gap,nt,close,[ ]),X4) :-
                                gap (x (gap,nt,close,[ ])), <- (26)
                                connects (3,that,4). <-- (30)

(1) sentence (4,10,x (gap,nt,trace,x (gap,nt,close,[ ])),X3) :-
      noun-phrase (4,9,x (gap,nt,trace,x (gap,nt,close,[ ])),X5),
      verb-phrase (9,10,X5,X3).

(3) noun-phrase (4,9,x(gap,nt,trace,x(gap,nt,close,[ ])),X5) :-
        determiner (4,5,x(gap,nt,trace,x(gap,nt,close,[ ])),X6),
        noun (5,6,X6,X7),
        relative (6,9,X7,X5).

(15) determiner (4,5,x(gap,nt,trace,x(gap,nt,close,[ ])),X6) :-
       terminal (the,4,5,x(gap,nt,trace,x(gap,nt,close,[ ])),X6).

                                  x(gap,nt,trace,x(gap,nt,close,[ ]))
(24) terminal (the,4,5,x(gap,nt,trace,x(gap,nt,close,[ ])),X6) :-
                  gap(x(gap,nt,trace,x(gap,nt,close,[ ]))), <- (26)
                  connects (4, the, 5). <-- (31)

(18) noun (5,6,x(gap,nt,trace,x(gap,nt,close,[ ])),X7) :-
      terminal (girl,5,6,x(gap,nt,trace,x(gap,nt,close,[ ])),X7).

                                  x(gap,nt,trace,x(gap,nt,close,[ ]))
(24) terminal (girl,5,6,x(gap,nt,trace,x(gap,nt,close,[ ])),X7) :-
                 gap(x(gap,nt,trace,x(gap,nt,close,[ ]))), <- (26)
                 connects (5, girl, 6). <- (32)
```

```
(9) relative (6,9,x(gap,np,trace,x(gap,nt,close,[ ])),X5):-
                open (6,6,x(gap,nt,trace,x(gap,nt,close,[ ])), X8),
                rel-marker (6,7,X8,X9),
                sentence (7,9,X9,X10),
                close (9,9,X10,X5)

                                                        X8
(10)open(6,6,x(gap,nt,trace,x(gap,nt,close[ ])),x(gap,nt,close,X11)
                                  x(gap,nt,trace,x(gap,nt,close,[ ]))


(12) rel-marker  (6,7,x(gap,nt,close,x(gapnt,trace,x(gap,nt,close,
  [ ])), X9):-
      x(gap,nt,trace,X12)
     rel.pronoun       (6,7,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,
     close,[ ])),X12).

(20)   rel-pronoun    (6,7,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,
       close,[ ])),X12):-
       terminal  (that,6,7,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,
       close,[ ])),X12).

(24)  terminal   (that,6,7,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,
      close,[ ])),X12):-
gap(x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,[ ])))), <- (26)
connects (6,that,7). <-- (33)

(1)  sentence   (7,9,x(gap,nt,trace,x(gap,nt,close,x(gap,nt,trace,
     x(gap,nt,close,[ ])))),X10):-
noun-phrase    (7,7,x(gap,nt,trace,x(gap,nt,close,x(gap,nt,trace,x
     (gap,nt,close,[ ])))),X13),
verb-phrase (7,9,X13,X10).

(5) noun-phrase (7,7,x(gap,nt,trace,x(gap,nt,close,x(gap,nt,trace,
    x(gap,nt,close,[ ])))),X13:-
                 trace (7,7,X14,X13).
```

```
(13) trace (7,7,X14,X13):-
                    virtual (trace,X14,X13).

(28) virtual (trace,x(gap,nt,trace,x(gap,nt,close,x(gap,nt,trace,x
     (gap,nt,close,[ ]))),X13).

(6) verb-phrase (7,9,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,
    [ ]))),X10):-
verb (7,8,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,[ ]))),X15),
noun-phrase (8,9,X15,X10).

(21) verb (7,8,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,[ ]))),
     X15):-
terminal  (likes,7,8,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,
     [ ]))),X15).

(24) terminal  (likes,7,8,x (gap,nt,close,x  (gap,nt,trace,x (gap,
  nt,close,[ ])), X15):-
gap (x(gap,nt,close,x(gap,nt,trace,x(gap,nt.close,[ ]))), (- (26)
connects (7, likes, 8). (- (35)

(3) noun-phrase  (8,9,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close
    [ ])),X10):-
            determiner (8,8,X16,X17),
            noun (8,9,X17,X18),
            relative (9,9,X18,X10).

(16) determiner (8,8,X16,X17).

(19) noun (8,9,X17,X18):-
                terminal (chocolate, 8, 9, X17, X18).

(24) terminal (chocolate,8,9,X17,X18):-
                        gap (X17), (- (26)
                        connects (8,chocolate,9). (- (36)

(8) relative (9,9,X18,X10).
```

```
(11)   close     (9,9,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,
       [ ]))),X5):-
virtual          (close,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,
                 [ ]))), X5).

(28) virtual  (close,x(gap,nt,close,x(gap,nt,trace,x(gap,nt,close,
              [ ])),X5).

(6) verb-phrase (9,10,X5,X3):-
          verb (9,10,X5,X19)
          noun-phrase (10,10,X19,X3).

(22) verb (9,10,X5,X19):-
                 terminal (kissed,9,10,X5,X19).

(24) terminal (kissed,9,10,X5,X19).-
                       gap(X19), <- (26)
                       connects(9,kissed,10). <- (37)

(5) noun-phrase (10,10,x(gap,nt,trace,x(gap,nt,close,[ ])), X3),
                trace(10,10,x(gap,nt,trace,x(gap,nt,close,[ ])),X3)

(13) trace (10,10, x(gap,nt,trace,x(gap,nt,close,[ ])),X3:-
                            virtual(trace, X20, X3).

(28) virtual (trace,x(gap,nt,trace,x(gap,nt,close,[ ])),X3)

(11) close (10,10,x(gap,nt,close,[ ]),X0):-
                     virtual (close,x(gap,nt,close,[ ]),X0).

(28) virtual (close,x(gap,nt,close,[ ]),X0).

(7) verb-phrase (10,11,[ ],[ ]):-
                 verb (10,11,[ ],[ ]).

(23) verb (10,11,[ ],[ ]):-
           terminal (laugh,10,11,[ ],[ ]).
```

```
(24) terminal (laugh,10,11,[ ],[ ]):-
                gap ([ ]), <- (27)
                connects (10,laugh,11). <- (38)
```

Concluindo, a sentença analisada, "The boy that the girl that liker chocolate kissed laugh" é aceita pela gramática de acordo com o processo do formalismo XG detalhado.

O exemplo anterior mostra claramente como funciona a limitação dos parenteses. Símbolos são colocados numa lista de extraposição através de regras com mais de um símbolo no lado esquerdo da regra e são removidos desta lista através do predicado 'virtual' através da política "last-in-first-out", ou seja, a lista de extraposição é uma pilha. Na realidade esta propriedade da lista de estraposição é exatamente a característica necessária para balancear os parentenses auxiliares durante os passos intermediários da derivação.